

## Dissertation

# Delegated Authenticated Authorization in the Life Cycle of Smart Objects in the Internet of Things

Stefanie Gerdes

A thesis to be submitted to fulfill the requirements for the degree of  
*Doktor der Ingenieurwissenschaft*  
– Dr.-Ing –

1. Gutachter Prof. Dr.-Ing. Carsten Bormann  
2. Gutachter Prof. Dr. Matthias Wählisch

Bremen, den 6. August 2021

**Gerdes, Stefanie**

gerdes@informatik.uni-bremen.de

Delegated Authenticated Authorization in the Life Cycle of Smart Objects in the Internet of Things

Datum des Promotionskolloquiums: 19 Juli 2021

Fachbereich 3 – Mathematik und Informatik

Universität Bremen, August 2021

## **Erklärung**

Ich versichere, diese Arbeit ohne fremde Hilfe angefertigt zu haben. Es wurden keine anderen als die genannten Quellen und Hilfsmittel verwendet. Alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, sind als solche kenntlich gemacht.

Bremen, 6. August 2021



## Abstract

Equipping everyday objects with microprocessors opens up a new range of applications. These “smart” objects often have sensors and actuators that allow them to monitor their environment and interact with it. Interconnecting smart objects with the help of Internet protocols, thereby creating an *Internet of Things*, enables the devices to communicate with each other and their users. If smart objects are integrated into all aspects of everyday life, they will be entrusted with vast amounts of data. Due to limited hardware resources, smart objects are hard-pressed to use common security mechanisms. Also, their application scenarios differ from typical Web scenarios: smart objects often need to communicate unsupervised and must protect their user’s security objectives on their own.

To establish trust in the new applications, users must be in control of their smart objects, the data handled by them, and their effect on the physical world. This thesis aims at enabling smart objects to enforce their users’ security decisions and participate in the protection of data. To achieve this, we revisit the objectives and design of authentication and authorization solutions. The authenticated authorization model introduced in this thesis identifies the fundamental requirements for authorization and task delegation that effective security solutions must satisfy. From the fundamentals, we derive the tasks that an endpoint must at least be able to perform to communicate securely. The model thus assists solution designers in finding gaps and vulnerabilities in security specifications. It includes the protection of a security objective that was previously missing from the literature, the *data destination verifiability*; it is needed to avoid attacks that make endpoints believe they are the intended receiver of a message, such as man-in-the-middle attacks.

Based on our model, we develop the *task delegation* architectural style. It supports less powerful, *constrained* devices by coupling them with a less-constrained *authorization manager*, to which challenging authentication and authorization tasks are offloaded. We implement the architectural style with the Delegated CoAP Authenticated Authorization Framework (DCAF). As the, to the best of our knowledge, only protocol that implements a separate authorization manager for the client side, it supports unsupervised constrained clients as well as constrained servers, and facilitates RESTful communication across organization boundaries. DCAF’s design offers solutions for important problems such as secure key distribution, simplified time synchronization, and the revocation of authorization and authentication data. By coupling DCAF with common security solutions, smart

objects can securely be integrated into the big Internet. Thus, DCAF enables a true Web of Things.

## Zusammenfassung

Mit Mikroprozessoren ausgestattete Alltagsgegenstände eröffnen neue Anwendungsbereiche. Diese "smarten" Objekte (engl. smart objects) sind oft mit Sensoren und Aktuatoren bestückt, die ihnen die Beobachtung und Interaktion mit ihrer Umwelt erlauben. Werden Smart Objects durch Internet-Protokolle zu einem *Internet der Dinge* verbunden, können sie miteinander und ihren Nutzern kommunizieren. Wenn die Geräte in alle Bereiche des täglichen Lebens integriert werden, verarbeiten sie erhebliche Mengen an Daten. Smart Objects haben aufgrund begrenzter Hardware-Ressourcen Schwierigkeiten, verbreitete Sicherheitsmechanismen zu verwenden. Auch unterscheiden sich ihre Anwendungen oft grundlegend von typischen Web-Szenarien: Smart Objects kommunizieren meist autonom und müssen Sicherheitsziele ihrer Nutzer daher selbständig schützen.

Um Vertrauen in die neuen Anwendungen zu gewinnen, müssen Nutzer über ihre Smart Objects und die darauf verarbeiteten Daten bestimmen und deren Einfluss auf die physische Welt kontrollieren können. Die vorliegende Arbeit zielt darauf ab, Smart Objects zu befähigen, die Sicherheitsentscheidungen ihrer Nutzer durchzusetzen und so an dem Schutz ihrer Daten mitzuwirken. Um dies zu erreichen, werden Ziele und Ausgestaltung etablierter Authentisierungs- und Autorisierungslösungen überdacht. Das in dieser Arbeit vorgestellte Modell zur authentisierten Autorisierung identifiziert die grundlegenden Anforderungen an Autorisierung und Delegation, die eine effektive Sicherheitslösung erfüllen muss. Von diesen Grundsätzen werden die Aufgaben abgeleitet, die ein Endpunkt mindestens erfüllen muss, um sicher zu kommunizieren. Das Modell hilft Protokoll-Designern, Lücken und Verwundbarkeiten in Spezifikationen zu finden. Es berücksichtigt ein Sicherheitsziel, das bisher in der Literatur nicht beachtet wurde: die Prüfbarkeit des Bestimmungsortes der Daten (Datenzielprüfbarkeit). Sie ist nötig, um Angriffe aufzudecken, bei denen einem Endpunkt suggeriert wird, er sei der beabsichtigte Empfänger einer Nachricht: beispielsweise Man-in-the-Middle-Angriffe.

Auf der Grundlage unseres Modells wird der Architekturstil *Task Delegation* entwickelt. Er unterstützt leistungsschwache Geräte, indem es sie mit einem *Autorisierungsmanager* koppelt, der ihnen Authentisierungs- und Autorisierungsaufgaben abnimmt. Der Architekturstil wird durch das Delegated CoAP Authorization Framework (DCAF) realisiert. Als das einzige uns bekannte Protokoll, das einen client-seitigen Autorisierungsmanager vorsieht, unterstützt es autonome

leistungsschwache Clients und Server und ermöglicht so die REST-basierte Kommunikation über Organisationsgrenzen hinweg. DCAFs Design bietet Lösungen für wichtige Probleme wie vereinfachte Zeitsynchronisierung und den Rückruf von Authentisierungs- und Autorisierungsdaten. Durch die Kopplung DCAFs mit verbreiteten Sicherheitslösungen können Smart Objects sicher in das große Internet integriert werden. So ermöglicht DCAF ein echtes Web der Dinge.

## Acknowledgements

I would like to thank my supervisor, Prof. Dr.-Ing. Carsten Bormann, for the many challenging but valuable discussions, and for teaching me to open my eyes and open my eyes again to see to the bottom of a problem. I would also like to thank Prof. Dr. Matthias Wählisch for agreeing to be my second reviewer and giving me valuable feedback for my work. Writing this thesis would not have been possible without Prof. Dr.-Ing. Ute Bormann, who first sparked my interest in computer networks and who supported me during this work.

I am very grateful for the support I received from the TZI and the Universität Bremen who granted me a scholarship to finish my work. I would particularly like to thank Prof. Dr.-Ing. Michael Lawo, Prof. Dr.-Ing. Rolf Drechsler, Sabine Veit and Regine Janssen.

My work benefited from discussions with my colleagues at the Arbeitsgruppe Rechnernetze. I would especially like to thank Dr.-Ing. Olaf Bergmann who designed libcoap and tinydtls and whose deep knowledge of writing implementations for constrained devices was very helpful to me. His support, encouragement and advice were invaluable for my work. Dr. Karsten Sohr provided helpful comments and encouragement for my work on authenticated authorization. I am also grateful for the many fruitful discussions with Klaus Hartke.

Much of my research was conducted in the context of the IETF, in particular the CoRE and the ACE working groups. I want to thank the participants of these working groups for the interesting discussions and valuable input.

I enjoyed working with Jens Trillmann who used the model for authenticated authorization to analyze the OAuth 2.0 framework, and who provided valuable comments and feedback. I am also grateful to Tobias Hartwich for his DCAF prototype and for his work on DCAF revocation and sliding windows. I would like to thank Tim Laurinat for comments and proofreading.

I very much appreciate the help of my friends and family who patiently tried to understand what my work is about and offered advice and encouragement. I am particularly grateful to my wonderful husband for his infinite patience and unwavering support. Infinite thanks to my mom and dad for encouraging me to choose my own way and supporting me in my decisions.

# Contents Overview

<b>Acronyms</b>	<b>xxi</b>
<b>I. Background and Requirements</b>	<b>1</b>
1. Introduction	2
2. Authorization in Communication Scenarios	18
3. Characteristics of Smart Objects	26
4. Authorization in Constrained Environments	46
<b>II. Authenticated Authorization Model</b>	<b>67</b>
5. Authenticated Authorization Model	68
6. Comparison with the BAN Logic	131
7. Example: Transport Layer Security in the Web	136
8. Conclusion	148
<b>III. Architecture Model and Protocols</b>	<b>151</b>
9. Architecture	152
10. Solution Design Considerations	169

<b>11. ACE DTLS Profile for Constrained Environments</b>	<b>185</b>
<b>12. Delegated CoAP Authenticated Authorization Framework</b>	<b>209</b>
<b>13. DCAF Evaluation</b>	<b>276</b>
<b>14. Authorization Transitions</b>	<b>328</b>
<b>15. Integrating Constrained Devices into the Big Internet</b>	<b>354</b>
<b>IV. Conclusion and Outlook</b>	<b>365</b>
<b>16. Conclusion</b>	<b>366</b>
<b>17. Outlook</b>	<b>383</b>
<b>V. Appendix</b>	<b>385</b>
<b>A. Comparison between ANAZEM and the BAN Logic</b>	<b>386</b>
<b>B. DTLS Profile Analysis</b>	<b>402</b>
<b>C. DCAF Alternative Architectures</b>	<b>431</b>
<b>D. DCAF Data Structure</b>	<b>436</b>
<b>Bibliography</b>	<b>439</b>

# Contents

<b>Acronyms</b>	<b>xxi</b>
<b>I. Background and Requirements</b>	<b>1</b>
<b>1. Introduction</b>	<b>2</b>
1.1. Motivation . . . . .	6
1.2. Research Questions . . . . .	11
1.3. Contributions . . . . .	13
1.4. Outline . . . . .	16
<b>2. Authorization in Communication Scenarios</b>	<b>18</b>
2.1. Security Objectives . . . . .	19
2.2. Stakeholders . . . . .	21
2.3. Internet Threat Model . . . . .	21
2.4. Granularity of Authorization . . . . .	22
2.5. OAuth 2.0 . . . . .	23
2.6. Conclusion . . . . .	25
<b>3. Characteristics of Smart Objects</b>	<b>26</b>
3.1. Device Limitations . . . . .	27
3.1.1. User Interfaces . . . . .	27
3.1.2. Energy . . . . .	28
3.1.3. Network Communication . . . . .	28
3.1.4. Clocks . . . . .	30
3.2. Life Cycle . . . . .	32
3.3. Protocol Stack Elements . . . . .	33
3.3.1. IEEE 802.15.4 . . . . .	33

3.3.2.	IP . . . . .	34
3.3.3.	6LoWPAN . . . . .	34
3.3.4.	Concise Binary Object Representation . . . . .	35
3.4.	Representational State Transfer . . . . .	35
3.5.	Constrained Application Protocol . . . . .	37
3.5.1.	Observe . . . . .	41
3.5.2.	Discovery . . . . .	41
3.5.3.	Security . . . . .	42
3.6.	Requirements Resulting from Hardware Limitations . . . . .	42
3.7.	Conclusion . . . . .	44
<b>4.</b>	<b>Authorization in Constrained Environments</b>	<b>46</b>
4.1.	Use Cases for Authorization in Constrained Environments . . . . .	47
4.1.1.	Container Monitoring . . . . .	48
4.1.2.	Sports and Entertainment . . . . .	51
4.1.3.	Building Automation . . . . .	52
4.1.4.	Smart Metering . . . . .	54
4.2.	Discussion . . . . .	55
4.2.1.	Constrained and Less-Constrained Communication . . . . .	56
4.2.2.	Granularity . . . . .	57
4.2.3.	Autonomous Communication . . . . .	58
4.2.4.	Denial of Service . . . . .	58
4.2.5.	Offline / Cloud . . . . .	60
4.2.6.	Perimeter Security . . . . .	60
4.2.7.	Multiple Stakeholders . . . . .	61
4.2.8.	Mutual Authorization . . . . .	61
4.3.	Requirement Summary . . . . .	62
4.3.1.	Environment Requirements . . . . .	62
4.3.2.	Authorization Rule Requirements . . . . .	63
4.3.3.	Authorization Rule Update Requirements . . . . .	64
4.3.4.	Lifecycle Requirements . . . . .	64
4.4.	Conclusion . . . . .	64

<b>II. Authenticated Authorization Model</b>	<b>67</b>
<b>5. Authenticated Authorization Model</b>	<b>68</b>
5.1. Related Work . . . . .	72
5.2. Authorization . . . . .	75
5.2.1. Data Destination Verifiability . . . . .	76
5.2.2. Authorization Fundamentals . . . . .	78
5.2.3. Authorization-Related Tasks . . . . .	82
5.2.4. List of Authorization Tasks . . . . .	93
5.3. Authentication . . . . .	96
5.4. Task Delegation . . . . .	98
5.4.1. Delegation Fundamentals . . . . .	99
5.4.2. Delegation Tasks . . . . .	101
5.4.3. List of Delegation Tasks . . . . .	107
5.5. Securing Application Data . . . . .	110
5.6. Omission of Tasks . . . . .	110
5.7. Minimal Set of Tasks . . . . .	114
5.8. Satisfying the Fundamentals . . . . .	116
5.9. Model Completeness . . . . .	119
5.10. Detection of Typical Vulnerabilities of Protocols . . . . .	120
5.10.1. Simple Bank Fraud . . . . .	121
5.10.2. Pay-Per-View Television . . . . .	121
5.10.3. Wide Mouthed Frog . . . . .	122
5.10.4. Challenge-Response . . . . .	123
5.10.5. Denning-Sacco . . . . .	123
5.10.6. Middleperson Attack . . . . .	125
5.10.7. CCITT . . . . .	125
5.10.8. Needham-Schroeder Protocol . . . . .	126
5.10.9. Decryption and Signature . . . . .	127
5.10.10. Addressed Tasks and Fundamentals . . . . .	128
5.11. Conclusion . . . . .	129
<b>6. Comparison with the BAN Logic</b>	<b>131</b>
6.1. Discussion . . . . .	132

6.2. Conclusion . . . . .	133
<b>7. Example: Transport Layer Security in the Web</b>	<b>136</b>
7.1. Client-side Authorization . . . . .	137
7.1.1. Delegating the Attribute Validation . . . . .	138
7.1.2. Validating the CA's Authorization . . . . .	141
7.2. Server-Side Authorization . . . . .	143
7.3. Secure Communication . . . . .	144
7.4. TLS Example Conclusion . . . . .	146
<b>8. Conclusion</b>	<b>148</b>
<b>III. Architecture Model and Protocols</b>	<b>151</b>
<b>9. Architecture</b>	<b>152</b>
9.1. Terminology Translations . . . . .	153
9.2. Architectural Style . . . . .	154
9.3. Architecture Model . . . . .	156
9.3.1. Actors . . . . .	157
9.3.2. Architecture Variants . . . . .	158
9.3.3. Intermediaries . . . . .	159
9.4. Evaluation of Architectural Properties . . . . .	162
9.4.1. REST . . . . .	163
9.4.2. Hardware Requirement Evaluation . . . . .	164
9.4.3. Environment Requirements . . . . .	165
9.4.4. Authorization Rule Requirements Evaluation . . . . .	166
9.4.5. Authorization Rule Update Requirements Evaluation . . . . .	167
9.5. Conclusion . . . . .	168
<b>10. Solution Design Considerations</b>	<b>169</b>
10.1. Cryptographic Algorithms . . . . .	169
10.1.1. Pre-shared Keys . . . . .	171
10.1.2. Asymmetric Keys . . . . .	172
10.1.3. Forward Secrecy . . . . .	175

10.1.4. Encryption and Message Authentication . . . . .	176
10.1.5. Object Security . . . . .	177
10.1.6. Discussion . . . . .	179
10.2. Validity of Authorization Information . . . . .	180
10.3. Nonces and Time . . . . .	182
10.4. Conclusion . . . . .	183
<b>11. ACE DTLS Profile for Constrained Environments</b>	<b>185</b>
11.1. DTLS Profile Overview . . . . .	186
11.1.1. Obtaining SAM Information . . . . .	187
11.1.2. Obtaining the Access Token . . . . .	188
11.1.3. Pre-Shared Key Mode . . . . .	189
11.1.4. Raw Public Key Mode . . . . .	190
11.1.5. Resource Access . . . . .	190
11.2. DTLS Profile Evaluation and Improvements to the ACE Framework	190
11.2.1. Server Side Delegation . . . . .	192
11.2.2. Client Side Delegation . . . . .	198
11.3. Conclusion . . . . .	206
<b>12. Delegated CoAP Authenticated Authorization Framework</b>	<b>209</b>
12.1. Protocol Design Overview . . . . .	211
12.1.1. Server Discovery . . . . .	213
12.1.2. Security Associations . . . . .	214
12.1.3. Mutual Authenticated Authorization Between CAM and SAM218	
12.1.4. Requirements for the Underlying Security Solution . . . . .	220
12.1.5. Authorization Information . . . . .	221
12.1.6. Keying Material . . . . .	223
12.1.7. Freshness and Validity . . . . .	224
12.2. Protocol Flow . . . . .	232
12.2.1. SAM Discovery . . . . .	233
12.2.2. Access Request Message . . . . .	237
12.2.3. Ticket Request Message . . . . .	239
12.2.4. Access Ticket Generation . . . . .	242
12.2.5. Ticket Grant Message . . . . .	250

12.2.6. Ticket Transfer Message . . . . .	251
12.2.7. Transmitting the Access Ticket . . . . .	255
12.2.8. Validating the Ticket Face . . . . .	258
12.2.9. Security Association Setup between C and S . . . . .	259
12.2.10. Authorized Communication . . . . .	260
12.2.11. Updating the Access Ticket . . . . .	261
12.2.12. Resource Registration . . . . .	264
12.2.13. Key Derivation . . . . .	265
12.3. Alternative Architectures . . . . .	265
12.4. Revocation . . . . .	266
12.5. Group Communication . . . . .	268
12.6. Server-Initiated Ticket Request . . . . .	269
12.7. RESTful Design . . . . .	270
12.7.1. Statelessness . . . . .	271
12.7.2. Loosely Coupled Components . . . . .	271
12.8. Implementation . . . . .	272
12.9. Conclusion . . . . .	273
<b>13. DCAF Evaluation</b>	<b>276</b>
13.1. Authorization Fundamentals Evaluation . . . . .	276
13.1.1. Unauthorized Resource Request Exchange . . . . .	278
13.1.2. Access Request Evaluation . . . . .	278
13.1.3. Ticket Request Evaluation . . . . .	280
13.1.4. Ticket Face Generation Evaluation . . . . .	284
13.1.5. Client Information Generation Evaluation . . . . .	287
13.1.6. Ticket Grant Message Evaluation . . . . .	288
13.1.7. Ticket Transfer Message Evaluation . . . . .	290
13.1.8. Ticket Transmission Evaluation . . . . .	295
13.1.9. Trust Establishment Evaluation . . . . .	298
13.1.10. Authorization Fundamentals Summary . . . . .	298
13.2. Quantitative Evaluation . . . . .	300
13.2.1. Quantity of Network Communication . . . . .	300
13.2.2. Memory Footprint . . . . .	309
13.2.3. Quantitative Evaluation Summary . . . . .	317

13.3. Requirements Evaluation . . . . .	318
13.3.1. Hardware Limitation Requirements . . . . .	319
13.3.2. REST . . . . .	321
13.3.3. Environment Requirements . . . . .	322
13.3.4. Authorization Rule Requirements . . . . .	324
13.3.5. Authorization Rule Update Requirements . . . . .	325
13.4. Evaluation Conclusion . . . . .	326
<b>14. Authorization Transitions</b>	<b>328</b>
14.1. Authorization Tasks . . . . .	330
14.2. Related Work . . . . .	331
14.3. Design of Authorization Transitions . . . . .	333
14.3.1. Authorization Manager Data Resources . . . . .	333
14.3.2. Planned Authorization Permission Transfer . . . . .	335
14.4. Authorization Transitions in the Life Cycle . . . . .	341
14.4.1. Manufacturing . . . . .	341
14.4.2. Commissioning . . . . .	342
14.4.3. Operation . . . . .	346
14.4.4. Maintenance . . . . .	346
14.4.5. Decommissioning . . . . .	348
14.4.6. Handover . . . . .	349
14.5. Fallback Mechanism . . . . .	350
14.6. Software Updates . . . . .	351
14.7. Conclusion . . . . .	352
<b>15. Integrating Constrained Devices into the Big Internet</b>	<b>354</b>
15.1. Cloudy IoT . . . . .	355
15.2. Intermediaries . . . . .	356
15.3. OAuth 2.0 Integration . . . . .	357
15.4. OpenID Connect Integration . . . . .	361
15.5. Conclusion . . . . .	363

<b>IV. Conclusion and Outlook</b>	<b>365</b>
<b>16. Conclusion</b>	<b>366</b>
16.1. Defining Characteristics of Smart Objects and their Environments .	368
16.2. Required Tasks . . . . .	369
16.3. Architectural Style and Architecture . . . . .	371
16.4. Design of the Authenticated Authorization Solution . . . . .	374
16.4.1. DTLS Profile . . . . .	375
16.4.2. Delegated CoAP Authenticated Authorization Framework .	376
16.5. Life Cycle . . . . .	379
16.6. Integration . . . . .	380
16.7. Standardization . . . . .	382
<b>17. Outlook</b>	<b>383</b>
<b>V. Appendix</b>	<b>385</b>
<b>A. Comparison between ANAZEM and the BAN Logic</b>	<b>386</b>
A.1. Message 1 . . . . .	387
A.2. Message 2 . . . . .	388
A.2.1. Validating the Claim in Message 2 . . . . .	390
A.2.2. Freshness and Validity of S's Claim in Message 2 . . . . .	390
A.2.3. Validating S's Authorization to Send Message 2 . . . . .	392
A.2.4. Destination Validation . . . . .	393
A.2.5. Evaluating the Claim . . . . .	394
A.3. Message 3 . . . . .	394
A.3.1. Validating S's Claim . . . . .	395
A.3.2. Freshness and Validity of S's Claim . . . . .	395
A.3.3. Validating S's Authorization . . . . .	396
A.3.4. Destination Validation for S's claim . . . . .	397
A.3.5. Evaluating S's Claim in Message 3 . . . . .	397
A.3.6. Validate that S's Claim Relates to A . . . . .	398
A.4. Message 4 . . . . .	399
A.4.1. Validate that S's Claim Relates to B . . . . .	400

<b>B. DTLS Profile Analysis</b>	<b>402</b>
B.1. Unauthorized Resource Request . . . . .	402
B.2. SAM Information Message . . . . .	403
B.3. Token Request . . . . .	404
B.4. Access Token Generation . . . . .	407
B.4.1. SAM Validates C's Authorization . . . . .	408
B.4.2. SAM Performs the D0 and D1 for S . . . . .	411
B.5. Client Information . . . . .	414
B.6. Access Token Response . . . . .	418
B.7. Access Token Transmission . . . . .	422
B.8. Access Token Validation . . . . .	423
B.9. S validates SAM's authorization . . . . .	426
B.10. Authorized Communication . . . . .	427
<b>C. DCAF Alternative Architectures</b>	<b>431</b>
C.1. Combined CAM and SAM . . . . .	431
C.2. Combined Client and CAM . . . . .	433
C.3. Combined Server and SAM . . . . .	434
<b>D. DCAF Data Structure</b>	<b>436</b>
<b>Bibliography</b>	<b>439</b>

# Acronyms

- 6LoWPAN** IPv6 over low-power wireless area networks (RFC 4944). 33, **34**, 310
- ACE** Authentication and Authorization in Constrained Environments. 15, 16, **46**, 147, 179, 180, 182, 184–197, 199–203, 208, 217, 218, 226, 234, 242, 265, 291–294, 296–300, 308, 309, 312–314, 318, 359, 363, 365, 367, 368, 371, 374, 376, 394, 398, 400, 401, 403–408, 410–421, 424, 427
- AEAD** Authenticated Encryption with Associated Data. 140, **170**, 171, 195, 214, 238, 273, 278, 280, 285, 366, 409
- AES** Advanced Encryption Standard. 240
- AIF** Authorization Information Format (draft-ietf-ace-aif-00). **214**, 230, 231, 257, 286, 288, 303, 350, 351, 428, 429
- AM** Authorization Manager. **149**, 150, 151, 153, 158–161, 187, 196, 203, 207–209, 211, 212, 215, 216, 218, 219, 222–225, 228, 257, 259–262, 265, 267, 270, 277, 283, 300, 309, 312, 315, 316, 318–321, 323–333, 335–344, 347–349, 351, 352, 365, 368–372, 422
- ANAZEM** Authenticated Authorization Evaluation Model. 13–16, 66, **69**, 70–72, 87, 116–119, 122, 125–131, 142, 143, 146, 268–270, 286, 289, 291, 317, 343, 348, 358, 362–364, 366, 368, 370, 375, 378, 379, 382–384, 386–388, 394
- CA** Certificate Authority. **133**, 134–138, 141, 167, 168, 173, 213
- CAM** Client Authorization Manager. **152**, 196, 204, 205, 207, 209–216, 218–220, 222, 223, 225–227, 230–233, 235, 236, 238, 241–247, 256, 259–262, 267, 269–286, 289, 290, 292, 294, 298–300, 302, 309, 311, 312, 314, 316, 349, 350, 352, 353, 370, 371, 424–429

**CBOR** The Concise Binary Object Representation (RFC 7049). 33, **34**, 35, 172, 182, 234, 239, 247, 265, 293, 302, 427–429

**CoAP** The Constrained Application Protocol (RFC 7252). 26, 27, **37**, 38–44, 147, 152, 153, 159, 171, 180, 198, 206, 213, 214, 218, 226, 231, 232, 235, 237, 241, 257, 259, 265, 298, 301, 304, 306–312, 320, 323, 324, 334, 339, 345, 352, 373, 376, 408, 411, 412, 428

**COP** Client Overseeing Principal. **151**, 194, 198, 201, 205, 208–210, 213, 218–220, 227, 232, 233, 241, 244, 246, 252, 261, 269, 271, 272, 274, 277, 282, 283, 286, 289, 309, 314, 315, 349, 352, 370, 371, 376, 424, 425, 428

**CoRE** Constrained RESTful Environments. **37**, 38, 41, 228

**COSE** CBOR Object Signing and Encryption (RFC 8152). **172**, 182, 238, 240, 251, 302, 348, 406, 409, 410

**CWT** CBOR Web Token (RFC 8392). **182**, 226, 231, 233, 246, 404, 405

**DCAF** Delegated CoAP Authenticated Authorization Framework. 14–16, 180, 189, 199, 202, **203**, 204, 205, 207–211, 213–218, 222, 224–227, 230, 232, 233, 236–238, 242, 246, 248, 252, 255, 257, 259, 261, 263–294, 296–305, 307–318, 320, 323–327, 331, 332, 337, 339, 343–345, 348–355, 359, 367–374, 376, 395, 422, 427, 429

**DDoS** distributed denial of service. 7, **58**

**DoS** Denial of Service. 54, **57**, 58, 61, 188, 189, 191, 200, 201, 225, 249, 250, 262, 264, 315, 341, 369, 402, 414, 416, 425

**DTLS** Datagram Transport Layer Security (RFC 6347). 15, 16, 42, 68, 114, 152, 153, 163, 164, 166, 168–171, 176, 179–186, 189, 198–200, 203, 205, 208, 230, 246, 248–250, 252, 254–256, 261, 269, 289, 292, 294, 301, 304, 306–311, 315, 318, 345, 347, 359, 363, 367, 369, 371, 373, 374, 394, 395, 397, 398, 401, 402, 404–409, 412–414, 417, 418, 420, 421

**FS** Forward Secrecy. 169

**HMAC** Keyed-Hash Message Authentication Code (RFC 2104). **170**

**HTTP** Hypertext Transfer Protocol. 5, 23, 37, 38–40, 44, 56, 57, 147, 152, 153, 171, 197, 345

**IANA** Internet Assigned Numbers Authority. 34, 40

**IETF** Internet Engineering Task Force. 6, 15, 37, 46, 147, 179, 184, 360, 367

**IoT** Internet of Things. 5, 6, 7, 10, 13, 14, 16, 26–29, 34, 37, 38, 42–46, 54–60, 64, 146, 147, 164, 166, 168, 179, 196–199, 201, 204, 206, 221, 267, 309, 315, 322, 333, 342, 343, 345, 346, 348, 354, 355, 360, 361, 364, 367, 369, 372, 373

**JWT** JSON Web Token (RFC 7519). 182

**LwM2M** Lightweight Machine-to-Machine. 38

**MAC** Message Authentication Code. 135, 170, 171, 172, 238, 251–253, 258, 278, 288, 401, 402, 405, 406, 409–412, 417, 418

**OVP** overseeing principal. 21, 46–50, 60, 67, 74, 76–78, 80–84, 87–89, 92, 94, 96–99, 103, 111–113, 116, 119, 125, 160, 208, 209, 211, 212, 216, 283, 316, 323, 352

**PEP** Policy Enforcement Point. 80, 149

**PKIX** X.509 Public Key Infrastructure (RFC 5280). 131, 133, 134, 143

**PSK** pre-shared key. 248, 402, 404, 405, 407, 408

**REST** Representational State Transfer. 26, 27, 35, 36, 37, 39, 41, 43, 44, 46, 58, 147, 148, 153, 156–158, 161, 162, 180, 263, 264, 310, 312, 313, 315, 320, 324, 369

**RPK** Raw Public Keys (RFC 7250). 169, 182, 183, 191, 193, 216, 232–235, 238, 241, 252, 254, 255, 272, 278, 296, 396, 397, 400, 402, 405, 407, 408, 428

**RTC** real-time clock. 30, 173, 218, 219, 221, 225, 360

**SAM** Server Authorization Manager. 152, 180–200, 203–205, 208, 210–213, 216–223, 225–232, 235–246, 248, 250–252, 254–263, 269–283, 285–290, 292–294, 296–300, 303, 304, 309, 311, 314, 316, 325, 349–353, 367, 368, 370, 371, 395–421, 423–429

**SITR** Server-Initiated Ticket Request. 262, 263, 314, 425

**SOP** Server Overseeing Principal. **151**, 152, 186, 187, 191, 196–198, 208–210, 213, 218, 221, 227, 235–237, 241, 242, 244, 269, 273, 274, 276–278, 288, 314, 315, 349, 350, 352, 425, 426, 429

**TLS** Transport Layer Security (RFC 5246). 16, 117, **131**, 132–135, 137–143, 152, 153, 163, 164, 166, 167, 170, 171, 176, 195, 197, 198, 208, 213, 248, 249, 345, 398, 401, 406, 407, 409, 412

**UMA** User Managed Access. **146**, 147

**URI** Uniform Resource Identifier (RFC 3986). **37**, 39, 41, 132, 197, 214, 229–231, 300, 395, 429

## **Part I.**

# **Background and Requirements**

*A robot must obey the orders given it  
by human beings*  
– Isaac Asimov

# 1. Introduction

Information processing devices are created to execute tasks for human beings to facilitate certain processes, and even provide capabilities to perform tasks that were not possible before (cf. Moor 1985). With the rise of the computing age, ethical questions arose about how a computer should act. The fear of the capabilities of computers manifested in scary scenarios of robots conquering the world and enslaving mankind. These visions belong to the main themes of science fiction literature in the second half of the 20th century. In 1942, Isaac Asimov, one of the most famous science fiction writers, introduced the three laws of robotics. In his stories, these rules are implanted into the brains of robots and define their behavior (Asimov 1942):

1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

These rules were designed to ensure that robots behave correctly and, above all, prevent humans from coming to harm. Although these laws are fictional, they represent common expectations of the behavior of machines. They would also be quite useful in today's information processing devices, especially if these are able to interact with the physical world: the devices should pursue the interests of humans, act according to their users' wishes, and prevent them and their assets from coming to harm. Unfortunately, especially the first and third law cannot easily be applied: devices would need to have a vast knowledge about the consequences of their actions. Current systems do not have this kind of knowledge,

---

and this is not likely going to change in the foreseeable future. The second law is easier to accomplish but less useful on its own. The question therefore is: how do we get an information processing device to act according to its users' wishes and protect them and their assets?

These considerations reflect in security mechanisms. According to Lampson, the original reason for building security mechanisms into computing systems was "to keep one user's malice or error from harming other users" (Lampson 1971, p. 1). To achieve this, the systems must not blindly obey orders from every human being but must act in their users' interest: the users' data and devices must be protected from other users.

Authentication and authorization are essential for the protection of data in computing systems (Abadi et al. 1993). To put it simply, **authorization is about what an entity is allowed to do, while authentication validates who an entity is**. The combination of these mechanisms enables a machine to decide **whether a certain entity is authorized**, e.g., if a communication partner is allowed to get a certain piece of information.

Authorization approaches originally focused on protecting the local access to stored data in multiuser systems (cf. Saltzer and Schroeder 1975, pp. 1279–1280). These access control mechanisms aim at limiting the actions that legitimate users (or programs that act in behalf of those users) are allowed to perform on an object such as a file in a file system (Sandhu and Samarati 1994). To achieve this, users must define access rules for their objects that state how a subject (e.g. a system entity such as a process) is allowed to access the object (Graham and Denning 1972; Sandhu and Samarati 1994).

**To be effective, authorization requires authentication** (cf. Sandhu and Samarati 1994, p. 40; Ferraiolo et al. 2003, p. 3). Authentication mechanisms originally aimed at identifying users, i.e., human beings, in a system: Only after the user has been identified, access control rules could be enforced.

With the increasing importance of communication between computers the need for security mechanisms for access over the network came into focus. Protecting

the communication between multiple endpoints created new challenges for authentication mechanisms, e.g., replay and masquerading attacks, and introduced the need for mutual authentication (Sandhu and Samarati 1994, p. 1).

Authorization also needed to cope with new challenges; the number of parties that are participating in a communication or are interested in the transmitted data increased. **These parties are often unknown to each other prior to the communication, and often do not have a common central authority that everyone trusts** (Li et al. 2003, p. 129). Also, in scenarios where multiple parties are communicating, the question of ownership for a piece of data is more difficult to answer. In distributed systems, authorization mechanisms may need to consider the (probably contradicting) authorization rules of multiple entities for authorization (Woo and Lam 1992a).

In the last decades, the usage behavior for computer systems has changed. Users no longer have a single powerful general-purpose machine but rather use several smaller devices that were specifically designed for the location and the task they need to perform. This development culminates in equipping everyday objects such as temperature regulating devices, weather stations or intelligent light switches with integrated microprocessors. These *smart objects* are developed to be integrated into the environment of their users with the purpose to—unseen and autonomously—make life easier for them. Interconnected smart objects act as a link between the physical world, the cyber world, and their users. Their ability to interact with their environment is one of the most remarkable properties of smart objects. Their impact on the physical world makes it even more important that these devices act in the interest of their users and other stakeholders.

The idea of interconnected things is not new. Some application areas, e.g., building automation, have a long history of communicating objects such as intelligent light switches or heating controls. However, those solutions are mostly proprietary: machines often do not interoperate well (or at all) with those of other manufacturers. Moreover, many of these devices require specialized networks, while almost all other technical communication has transitioned to Internet-based systems.

The Internet is designed for flexibility and thus allows for easy and quick innovation. The concept of *permissionless innovation*, a term coined by Vint Cerf,

---

demands that no permission is required for introducing new features, since top-down approaches will likely hinder innovation (Cerf et al. 2013, p. 7). The World Wide Web shapes the Internet as we know it today. It was designed as “an interactive world of shared information through which people could communicate with each other and with machines” (Berners-Lee 1996). The Hypertext Transfer Protocol (HTTP) is “by far the most widely used application protocol” (Shelby and Bormann 2009). One of the most important features of the Web is its principle of minimal constraints: as few things should be specified as possible to achieve flexibility (Berners-Lee 1996). This allows for loosely coupled architecture components which can evolve independently (*separate evolution*). New protocols and services can be added quickly (Shelby and Bormann 2009, p. 1), and new features are developed in a “rapid pace” (Berners-Lee 1996).

Since the success of the Internet and the Web can be attributed to these characteristics, smart objects can be expected to also benefit from Web-like communication patterns. By connecting smart objects and thus forming an *Internet of Things (IoT)*, the devices can use “communication services that use Internet protocols”<sup>1</sup>. The common factor for these protocols is the use of the Internet Protocol (IP). Thereby, *things* can interact with each other and their users and provide a whole new range of applications such as home automation<sup>2</sup>, health monitoring<sup>3</sup> and sports equipment<sup>45</sup> and even toothbrushes for improving dental health<sup>6</sup>. Smart cities aim at improving the life of citizens by gathering and distributing environment information (Woetzel et al. 2018), increasing energy efficiency (Ellsmoor 2019), and enhancing traffic control<sup>7</sup>.

In the industrial area, smart objects are envisioned to spawn the *fourth industrial revolution* (“Industry 4.0”): machines are interconnected, use sensor data to assess their environment, and are able to make simple decisions autonomously (Marr 2016; Kumar 2020). This development comprises smart factories with intercon-

---

<sup>1</sup><https://www.iab.org/activities/workshops/smartobjects/>

<sup>2</sup><https://www.eq-3.com/products/homematic.html>

<sup>3</sup><https://www.freestylelibre.us/>

<sup>4</sup><https://shop.liveathos.com/>

<sup>5</sup><https://www.wilson.com/en-us/explore/labs/connected-football-system>

<sup>6</sup><https://www.kolibree.com/en/>

<sup>7</sup><https://enterprise.verizon.com/products/internet-of-things/smart-cities-and-communities/>

nected industrial control systems (ICSes) where machines can even take action on their own.

As things that participate in the IoT were often designed for a specific application scenario and purpose, they will come in various sizes and shapes. Their hardware will vary in terms of storage space, RAM, processing power and networking capabilities. Some devices will have no displays or user interfaces. Some will be battery-powered and need to save as much energy as possible. General-purpose devices that are able to use common operating systems—such as desktop computers and smartphones—are usually not considered to be IoT things (FTC 2015). **In this work, we use the term Internet of Things to refer to interconnected devices with varied hardware capabilities that use the Internet protocol to communicate and typically have the ability to interact with their environment.** These devices often need to act without human intervention; interacting with them is inconvenient for their users because of missing interfaces, long response times, and the large number of devices users may own.

In 2018, the number of IoT devices in use are estimated to have reached between 7 and 10 billion devices (IoT Analytics 2018). According to Gartner, the enterprise and automotive IoT market grows to 5.8 billion endpoints in 2020 (Gartner 2019). McKinsey expects that by 2020 consumers will spend more than €12 billion annually on IoT devices and software (Berger-de Leon et al. 2018). The relevance of this development reveals itself in standardization efforts ranging from industry consortia, e.g., OMA Specworks<sup>8</sup>, to standardization organizations such as the World Wide Web Consortium (W3C)<sup>9</sup> and the Internet Engineering Task Force (IETF)<sup>10</sup>.

## 1.1. Motivation

It becomes apparent that we can expect smart objects to become a part of all aspects of everyday life. They thus will be entrusted with vast amounts of data.

---

<sup>8</sup><https://omaspecworks.org/>

<sup>9</sup><https://www.w3.org/>

<sup>10</sup><https://www.ietf.org>

Without appropriate protection, attackers may gain control over data and devices that are important to our lives. Depending on the application scenario, this might entail severe consequences. Users need to be able to control their devices according to their needs, i.e., they must be enabled to decide about the actions of their smart objects and to define what happens to the data that is handled by their devices. To achieve this, an authorization mechanism is required.

Many IoT devices currently offer no or only little security. Attackers can use the devices' vulnerabilities as an entrypoint to the network (see, e.g., Larson 2017), or even to our homes (Young 2020). Also, they may take over the users' devices. E.g., in 2020, a security researcher showed how to infiltrate a coffee maker and demand a ransom from its owner (Goodin 2020). Vulnerabilities in IoT devices may also affect the privacy of users, as e.g., attacks on security cameras, baby monitors and smart doorbells show (Tenable Research 2018; Bitdefender 2019; Marrapese 2020). Worryingly, even the companies that sell the devices spy on their customers' usage behavior as was discovered for the "smart" vibrator We-Vibe (Bouboushian 2016).

Since smart objects influence the physical world, attacks on these devices may even threaten the safety of customers. E.g., in January 2016, thermostats of the company NEST failed to work because of an error in a software update that drained the battery. The devices shut down and could no longer be controlled by their owners, causing the temperature in the affected homes to drop (Bilton 2016). An example for an even more direct attack on human safety was shown by security researchers who developed an exploit for a car's information system (Greenberg 2015). The exploit enabled the researchers to remotely take over the driving car so that the driver was no longer able to control it. In 2017, researchers found a weakness in an automated car wash: they developed an exploit that enabled them to remotely control components which can physically damage a car or even harm a human being (Rios 2017). Even medical devices can threaten the health of their users; in 2017, a vulnerability in a cardiac device was discovered which enabled attackers to manipulate pacing or administer a shock (FDA).

Compromised IoT devices can also be abused for distributed denial of service (DDoS) attacks: attackers take over a great number of devices and then use these

so-called botnets to attack specific targets. A prominent example for such an attack concerned the company Dyn that controls extensive parts of the Internet's Domain Name System (DNS) infrastructure. In 2016, it was the target of a large-scale DDoS attack from the Mirai botnet (Antonakakis et al. 2017).

Security experts, companies and governments are worried about the missing security of IoT devices as, e.g., the list of security and privacy guidelines shows that were published in recent years (cf., e.g., Schneier 2017).

The special characteristics of the smart objects themselves and of the environments where they are used ask for a reconsideration of common security mechanisms. Even though deceptively named "smart" objects, the devices often lack hardware resources such as processing power, RAM and storage space, and thus have problems using security protocols that are common for the *big Internet*<sup>11</sup>. Devices that can barely run the code for the applications for which they are designed will be hard-pressed to store large cryptographic libraries or perform complex security tasks.

These problems cannot be expected to be solved by hardware developments in the near future. The expected large number of devices and the fact that smart objects are often designed to perform only single, simple tasks entail that microprocessors evolve differently in this area. In the 1960s, the evolution in the development of microprocessors eventually led to Gordon Moore's observation that "the complexity for minimum component costs has increased at a rate of roughly a factor of two per year" (Moore 1965), a rate which he expected to remain constant for at least ten years. With occasional slight updates of the exponent, this prediction (Moore's Law) turned out to be more or less accurate until very recently: as expected, the processing power increased exponentially while the size of the devices shrunk. Smart objects are expected to be influenced by Moore's Law in a different way: "With many sensor network applications requiring extremely low-cost devices, hardware development is unlikely to yield any extensive improvements in resources for the foreseeable future" (Dutta and Dunkels 2012, p. 70). The processing power does increase eventually, but much

---

<sup>11</sup>In this thesis, we will use the term *big Internet* to refer to the Internet as it is used by less-constrained devices using common Internet protocols.

more slowly than for general-purpose computers. Instead, further development is focused on making cheaper, smaller and more energy-saving devices (Bormann et al. 2012, p. 63; Waldrop 2016, p. 147). **This dissertation will argue that even so-called *constrained devices* that are barely able to securely speak IP need to be supported by security mechanisms if they are supposed to contribute to achieving the security objectives of their users.**

But not only their potential limitations distinguish smart objects from general-purpose computers. Their application areas and environments also differ from Web scenarios. Smart objects will often be deployed in places without mains power and need to operate from a primary battery. To save energy, the devices need to sleep as much as possible and must avoid using their radio interface. Some devices may lack user interfaces and displays. Others may simply act as peripherals for these devices, e.g., remote controls. Interacting with the devices will often be tedious for users because of lacking user interfaces and long response times. Also, users may not always be present or may find it more convenient if the devices act on their own. Especially if users own large numbers of devices, interacting with all of them is infeasible. Smart objects are therefore often required to communicate and act autonomously without human intervention at the time of the communication. **This thesis postulates that smart objects need to be enabled to safeguard the interests of their users on their own.**

Application scenarios may involve multiple stakeholders whose security objectives may differ. The most obvious case is if the two communicating devices belong to different owners. Let us consider an example where a customer wants to buy medicine from an online pharmacy. While the pharmacy must make sure that it is actually a certain user that requests to buy something, the customer will also be interested in the confidentiality of her request. The stakeholder's interests may even conflict; e.g., while the customer attempts to protect her privacy, the pharmacy may wish to know every detail about her behavior to provide personalized advertising. **In this thesis, we postulate that the security objectives of all stakeholders whose devices are participating in the communication must be considered.**

To achieve the security objectives, devices need to establish *security associations* with peers before exchanging data with them, i.e., they need to know the data and

tasks with which their communication partners can be entrusted. **A device that has a security association with a peer is able to authenticate the peer and knows its authorizations.** In order to implement security associations, the devices need an authorization mechanism that provides for authorization rules, and an authentication mechanism to be able to map those rules to the entity with which they are communicating.

The setup and management of security associations is a challenging task, even in the big Internet. Common modern cryptographic mechanisms follow Kerckhoffs's principle which states that the security of a cryptographic mechanism must not depend on concealing the security mechanism, but only on the protection of the key (Kerckhoffs 1883, p. 12). The setup of a security association requires the secure distribution of keying material to the communicating peers, and to securely provide the semantic information relating to the keys, e.g., that the key belongs to a certain entity.

In the IoT, security associations provide additional challenges. The devices' users are often not present at the time of the communication to instruct the device. At the same time, the device might be less capable to perform tasks on its own because of its hardware limitations. Smart objects therefore need to rely more on other devices and need to establish security associations with them. To determine the authorization of an entity, they often also require a security association with their own users. The limitations of smart objects make storing large numbers of security credentials infeasible. Also, sending and receiving data is time and energy consuming, which makes the transmission of complex security structures such as X.509 certificates, which are often used in the Web, too expensive. And security associations are not static: during its lifetime, a smart object will likely need to communicate with various other devices. It must therefore be able to dynamically setup, manage and end security associations. Some phases in the device's life cycle are especially challenging, e.g., when a newly bought device is introduced to the network or an old device is decommissioned. **This dissertation will argue that the security of user data requires a model for the setup and management of security associations that covers the whole life cycle of smart objects.**

Although the IoT is envisioned to use Internet protocols, the limitations of the de-

vices restrict the protocols that can be used. To become true Internet components, IoT devices must be enabled to securely communicate with devices in the big Internet. **This thesis will analyze the secure integration of smart objects into the big Internet while considering the communication among things as well as between things and less constrained devices.**

## 1.2. Research Questions

This thesis focuses on safeguarding the interests of users in the Internet of Things. Our guiding research question is: **How can users be empowered to be the authority for their smart objects, control their behaviour in both the cyber world and the physical world, and decide what happens to the data handled by them in the Internet of Things?** From this broad research question we derive the following subordinate questions:

- **Which authentication and authorization problems can be identified by examining common characteristics of smart objects and the application scenarios where they are used?** To understand the requirements for security solutions in the Internet of Things, we analyze the characteristics of constrained devices and of typical application scenarios.
- **Which authentication and authorization tasks must a device at least be able to perform to participate in the protection of its users' data?** Hardware constraints limit the capabilities of constrained devices to perform challenging authentication and authorization tasks. But certain security tasks must be performed to protect data during communication. We identify the fundamental requirements that an authorization solution must satisfy to be secure and therefore effective. From these fundamentals, we derive the minimal set of tasks that endpoints must be able to perform so they can participate in the protection of their users' data.
- **How can architectural design principles that made the Web successful be incorporated in an architecture for the Web of Things which supports smart objects in the protection of their users' data?** From the architectural

design principals of the Web and the authorization and delegation fundamentals we derive an architectural style that is optimized for supporting constrained devices.

- **How must an authorization solution be designed to implement the tasks and the architecture?** The limitations of smart objects ask for light-weight security solutions. Moreover, since users may not be able to intervene at the time of the communication, smart objects must enforce their users' security objectives, concerning the data that they are handling, on their own, despite their limited system resources. This thesis investigates the lower bounds for system resources that need to be available on a smart object to perform authentication and authorization, and aims at providing a scalable, energy-efficient solution. Moreover, concepts for authentication and authorization are introduced that enable smart objects to cope with absent users.
- **How can secure authorization transitions be realized to achieve continuous protection of smart objects in their whole life cycle?** Implementing security associations is challenging even for less constrained devices in the big Internet. To ensure the security of data, smart objects must be able to establish, manage and terminate security associations with other devices and their users. This dissertation analyzes authorization transitions in the whole life cycle of smart objects and identifies the necessary steps to secure them.
- **How can constrained devices using light-weight IoT Security solutions communicate securely with devices in the big Internet?** To achieve true integration of smart objects, secure communication between things and general-purpose computers is required. Smart objects often have difficulties to implement common security protocols, but deploying IoT security solutions on devices in the big Internet is expensive and time-consuming, and not always a feasible solution. In this work, approaches for secure communication between smart objects and less constrained devices are analyzed, and an authentication and authorization solution is developed that considers the limitations of smart objects on one side and fits to existing authentication and authorization solutions on the other.

## 1.3. Contributions

The research in this thesis centers on achieving the security objectives of the users of smart objects, and keeping them in control of their data and devices. This overview is structured in theoretical, methodological and practical contributions. As theoretical contributions we consider adding fundamental knowledge to the scientific knowledge base. As methodological contributions we regard new methods for deriving knowledge from facts when applied to specific scenarios. For the practical contributions of this thesis, the theoretical findings are used to develop an actual solution that can be applied in application scenarios.

### Theoretical Contributions

- *Problem space analysis:* This dissertation provides an analysis of the problem space for securing smart objects in the Internet of Things: Representative real life use cases are collected and investigated. Special characteristics of the devices and their application scenarios are identified. Also, the users' security objectives in the respective application scenarios are determined.
- *Development of an authenticated authorization model:* We develop the Authenticated Authorization Evaluation Model (ANAZEM). It depicts the fundamental requirements of authorization processes. An authorization solution must satisfy all specified fundamentals to offer effective security in the whole authorization process. The tasks that endpoints must perform to participate in the protection of their owners' security objectives are identified and described.
- *Identifying data destination verifiability as a security objective:* In this thesis, a security objective is identified that is clearly needed for secure communication but is not yet mentioned in the literature: the data destination verifiability. Security solutions that do not consider the protection of this objective are vulnerable to man-in-the-middle attacks.
- *Development of the task delegation architectural style:* From the authenticated authorization model we derive the task delegation architectural style that couples each constrained device that is unable to perform authentication or

authorization tasks on its own with a less-constrained device. The most general resulting architecture is the four-corner architecture, where each constrained device has its own less-constrained device.

- *Evaluating the suitability of common authorization solutions:* An evaluation of existing authorization solutions for their suitability for the Internet of Things is presented. In particular, the dissertation investigates to which degree and in what configurations well-known authentication and authorization protocols such as OAuth 2.0 (RFC 6749) are applicable for constrained environments, and what tradeoffs have to be made.
- *Continuous protection of authorization transitions in the whole life cycle of smart objects:* For the protection of security associations in the whole life cycle of a smart object the relationships between a constrained device, a supporting less constrained device, and their user is analyzed. A model for authorization management is presented that is designed to secure authorization transitions accordingly.
- *Integration into the big Internet:* IoT devices will often be deployed in existing network infrastructures. To become real Internet components and avoid a silo solution for the IoT, constrained devices must be able to communicate with devices in the big Internet. We show how the Delegated CoAP Authenticated Authorization Framework (DCAF) can be integrated with existing solutions for the big Internet using prominent examples.

### **Methodological Contributions**

- *Development of a method for evaluating the quality of security solutions concerning their effectiveness:* We present a methodology to apply ANAZEM to security solutions. The model can thus be used to detect gaps and vulnerabilities.

### **Practical Contributions**

- *Development of DCAF:* With DCAF, a framework for authentication and authorization is proposed that supports constrained as well as less constrained devices. It implements the task delegation architectural style, and defines

how constrained devices perform security tasks that they need to execute themselves, while off-loading tasks that can be delegated. The protocol comprises a mechanism for distributing keys to the smart objects. DCAF fully enables clients as well as servers to participate in the protection of their owners' security objectives. The ideas for DCAF originated from discussions with Carsten Bormann and Olaf Bergmann. For this work, we refined the original DCAF version with the assistance of ANAZEM and thereby enhanced the security of the solution. The new version clarifies the necessary security associations, and the requirements on underlying security protocols. It also includes concepts such as the required contents of DCAF tickets and messages, validity options that enable constrained devices to determine the freshness and validity of the ticket, and a revocation mechanism (to name a few).

- *Development of the Datagram Transport Layer Security (DTLS, RFC 6347) profile for the Authentication and Authorization in Constrained Environments (ACE) framework:* A profile for the IETF ACE framework was developed to explore its suitability for constrained environments. The DTLS profile is based in parts on concepts from DCAF and is a joined effort of the ACE working group.
- *Design of a DCAF prototype:* An open source prototype of DCAF was developed that provides authentication, authorization and key distribution for smart objects. The source code can be found at <https://dcaf.science>. The prototype was developed in cooperation with Olaf Bergmann. Our primary contribution is the implementation of the DCAF protocol logic.

Much of the research for this thesis was conducted in the context of standardization efforts concerning authentication and authorization for constrained environments in the IETF. Details concerning the state of standardization can be found in section 16.7.

## 1.4. Outline

The thesis is divided into four parts. The first part provides necessary background information. It begins with an introduction to fundamental security concepts including authentication, authorization and security objectives (see chapter 2). In chapter 3 we examine the special characteristics of smart objects and introduce resulting requirements for an authorization solution. Chapter 4 then analyzes authorization in constrained environments by introducing typical use cases and deriving security requirements.

Authorization is only effective if the authorization process is continuously secure. In part II we introduce our authenticated authorization evaluation model ANAZEM. We identify the fundamental requirements that an authorization process must meet to be secure, and thereby describe how an authorization process with perfect continuous security must be designed. We introduce the tasks that must be performed to satisfy the fundamentals, thus generating a checklist that can be used to detect gaps and vulnerabilities in protocols and implementations (see chapter 5). In chapter 6, we analyze the Kerberos protocol with ANAZEM and compare the result to an analysis with the BAN logic. Chapter 7 shows how endpoints using Transport Layer Security (TLS, RFC 5246) in common Web scenarios perform authorization, authentication and delegation tasks. Chapter 8 summarizes our findings in this part.

The third part presents our architecture model and implementation. From our authenticated authorization model we derive the task delegation architectural style. It can be implemented with varied architectures, the most general of which is the four-corner architecture, where each constrained device has its own less-constrained device that assists with difficult security tasks (see chapter 9). In chapter 10, we discuss design decisions for authorization solutions in the IoT and analyze the suitability of various security mechanisms to satisfy the authorization and delegation fundamentals. Chapter 11 introduces our DTLS profile for the ACE framework. We conducted a security analysis of the profile using ANAZEM and thereby detected several gaps and vulnerabilities in the profile and the ACE framework. Due to our feedback, many of these problems were fixed in later versions of the framework. We will show that the ACE framework

and the DTLS profile do not fully support autonomous clients. Chapter 12 introduces the Delegated CoAP Authenticated Authorization Framework that fully implements the task delegation architectural style. We provide an evaluation of DCAF in chapter 13 that comprises a security analysis using ANAZEM, and discusses if and how DCAF meets the requirements we identified in the background part. To provide for an authentication and authorization solution for the whole life cycle of a smart object, chapter 14 introduces methods for secure authorization management using DCAF. Chapter 15 analyzes how constrained devices can be integrated into the big Internet.

We conclude our work with part four, which discusses the findings of the dissertation and provides an outlook on future work.

In the PDF version of this thesis, references are hyperlinks: by following them you can view the respective entries in the bibliography. Likewise, the requirements, and the fundamentals and tasks are represented by links that lead to their respective definitions in the background and model part. Abbreviations used in this thesis are linked to the respective entry in the glossary. A digital version of this thesis is available at <https://dx.doi.org/10.26092/elib/937>.

Throughout this work, we use footnotes to refer to project or organization pages. The respective URIs were accessed on September 27, 2020.

*Want to know a secret? Promise not to tell?*

– Snow White and the Seven Dwarfs

## 2. Authorization in Communication Scenarios

According to Menezes et al., an *authorization* entitles a subject to do or to be something (cf. Menezes et al. 1996, p. 3). The entity may thereby be allowed to, e.g., access a certain object in a specific way, to perform certain tasks, or to act as an administrator of an endpoint. Authorization mechanisms enable users to specify what they want to happen with their data and devices by defining authorization *policies* that reflect their security objectives. A policy typically consists of sets of *authorization rules* that state who is allowed or not allowed to see or modify data, and enable endpoints to enforce the policies of their owners.

As stated above, authorization approaches originally aimed at controlling the local access to stored information. With the evolution of network computing since the early 1980s, distributed systems have started to replace former mainframe architectures. The introduction of the World Wide Web in 1989 finally allowed people to easily share content not only with other users on the same system but with people all over the world.

The Web's main architectural components are servers that host the content and clients that access this content. Data owners who want to define who may use or modify their data need an authorization mechanism. Common authorization mechanisms for the Web such as the OAuth 2.0 framework (RFC 6749) focus on protecting the content stored on servers and granting access only to authorized clients. The server must determine whether the entity that attempts to access the information actually is authorized. While this scenario seems to be similar to the protection of data in a file system, security solutions must cope with new challenges, since the whole communication between server and client is performed

over the Internet and thus is prone to attacks. To distinguish authorized from unauthorized entities, an authentication mechanism is required. The authentication of communication partners is difficult since clients and servers may not know each other prior to the communication. Also, the Internet has no central authority that everyone trusts (Li et al. 2003, p. 129).

Finally, after the communication partners authenticated each other and the server validated the client's authorization, the content data must be securely transported between client and server: depending on the users' security objectives, others may not be allowed to read or modify this data.

The following sections will introduce basic concepts and terminology for authorization in communication scenarios. Section 2.1 introduces the security objectives that users may have for their data and devices. In section 2.2 we describe various types of relationships between devices and stakeholders. The Internet threat model that describes security characteristics of communication channels between endpoints in the Internet is explained in section 2.3. Section 2.4 introduces terminology concerning the granularity of authorization. In section 2.5, we give a short introduction to the OAuth 2.0 framework as the prevalent authorization mechanism for the Web.

## 2.1. Security Objectives

Human beings want their data to be protected from unauthorized use. That means that data is not disclosed or modified, and the access to it is not impeded, contrary to the owner's wish (Saltzer and Schroeder 1975, p. 1280). These security requirements manifest in *security objectives*. According to FIPS PUB 199, the main security objectives are (cf. FIPS PUB 199, p. 2):

- *Confidentiality*: data is not disclosed to unauthorized entities,
- *Integrity*: data is not manipulated by unauthorized entities and
- *Availability*: entities have unhindered access to the data or service they are authorized to use.

Additional objectives can be derived from these main security goals. *Privacy* is related to confidentiality. It refers to the right of a person to decide which personal information is shared with others (RFC 4949) and therefore encompasses legal aspects. The *Authenticity* of data comprises integrity and freshness (Anderson 2008, p. 14), and allows to determine the origin of data.

Authorization solutions are critical to attain the security objectives confidentiality and integrity (Ferraiolo et al. 2003, p.3). Misconfigured or wrongly designed authorization solutions can also result in availability breaches: Users might no longer be able to use data and services as they are supposed to.

Authentication mechanisms are required for authorization and thus also influence the confidentiality, integrity and availability of data and services. Moreover, they can assist in achieving additional security objectives such as *accountability* (tracing an action back to the executing entity) (RFC 4949) and *third-party verifiability* (proving to a third party, e.g., in court, that an action was performed by a certain entity). These security objectives do not require authorization and are not in scope of this work but might be needed in certain scenarios. To achieve them, the unique identification of an endpoint might be required. In other cases it may be more important to find out who is responsible for the endpoint's actions.

The authorization process itself has its own security objectives. Information that influences the authorization must be protected. If authorization data is transmitted, its integrity and authenticity must be validated. Additionally, it might be necessary to protect the confidentiality of the data, e.g., for privacy reasons. We will analyze the requirements for a secure authenticated authorization process in chapter 5.

Security objectives influence which security mechanism is required. The importance and weight of each security objective and the resulting protection requirements differ depending on the specific use case. Security objectives may even be conflicting, e.g., the confidentiality of data is difficult to achieve without interfering with the availability of data. We will take a closer look at the use cases relevant for this work in section 4.1.

## 2.2. Stakeholders

Access control approaches typically use the term “owner” to refer to the human being that can define access rights for an object (cf. e.g., Sandhu and Samarati 1994; Saltzer and Schroeder 1975). But the individual that defines the authorization policies for a communication endpoint or a data object on that endpoint is not necessarily the owner of the endpoint. Owners, users and administrators may be distinct individuals or organizations. In communication scenarios, at least two endpoints are involved in the communication and each of these endpoints may have a different owner.

There are various types of relationships between endpoint and human being: An endpoint is part of the belongings of its *owner* and is utilized by *users*. *Administrators* are in charge of the configuration and maintenance of the endpoint. Application scenarios may involve various *stakeholders*, i.e. people who are affected by the actions of endpoints. Moreover, the owner of the data on the endpoint may be different from the owner of the device. The security requirements of each of these roles may need to be considered.

In security literature, entities that are participating in a communication are often called *principals* (Anderson 2008, p. 12; RFC 4120, p. 15). The purpose of authorization mechanisms is to prevent unauthorized use (see also section 2.1). **We will use the term *overseeing principal (OVP)* to refer to an individual or a company that is the authority for an endpoint or the data on this endpoint, and directly or indirectly defines security policies for it.** Overseeing principals define what is authorized; their security objectives, that manifest in security policies, must be enforced. **Accordingly, we define the main authorization directive as: Overseeing principals are the main authorities for their data and devices: their decision about if and how these assets are used must be followed.**

## 2.3. Internet Threat Model

The security objectives described above are threatened by attacks. Dolev and Yao distinguish two types of attackers (Dolev and Yao 1983). *Passive attackers* only

listens in on the exchanged messages while *active attackers* can obtain and modify any message that is transmitted, and may also generate own messages.

Rescorla summarizes the assumptions about possible attacks on messages that are exchanged over the Internet in the Internet Threat Model (Rescorla 2001, pp. 1–2)<sup>1</sup>. The communication channel between endpoints is assumed to be more or less under the control of the attacker. Information transported over this channel can be manipulated or intercepted and attackers might insert forged or remove legitimate packets.

Successful attacks on an endpoint are assumed to compromise the communication between this endpoint and its peers. It is very difficult (if not impossible) for protocols to protect against attacks where an attacker has gained control over one endpoint. Protocols therefore need to assume the endpoints to be secure since there is nothing they can do about it if this is not the case. However, if one endpoint is compromised, this should not affect other conversations of its peers.

In 2013, Edward Snowden disclosed information from classified files of the U.S. National Security Agency (NSA) about the extent of its global surveillance programs. These disclosures revealed that more attention needs to be paid to a type of threat called pervasive monitoring, where a passive attacker performs widespread surveillance by monitoring content and metadata across the Internet (RFC 7258). This type of threat stresses the need for protecting the confidentiality of data that may not be that sensitive in itself but becomes sensitive if linked to other data.

## 2.4. Granularity of Authorization

Application scenarios may require various levels of granularity for authorization. In some cases, distinct authorization rules are granted to each user. However, this is not always necessary. If the principals' security objectives do not include the confidentiality of a piece of data, it can be disclosed to everyone. Likewise, if the

---

<sup>1</sup>The term *threat model* is often associated with the analysis and assessment of potential threats to a system. Throughout this work we will use this term as it is defined by Rescorla.

integrity is not required by the principals, pieces of data can be accepted from every source. We use the term *unrestricted authorization* to refer to cases where all entities are allowed to have unconditional access. For unrestricted authorization no authorization mechanism is required.

A very basic authorization mechanism may only distinguish between known and unknown entities; every authenticated entity then has the same entitlements. This strategy, that is sometimes referred to as *binary authorization* (cf. Blaze et al. 1999, p. 187), may fall short where multiple stakeholders are involved; stakeholders may require various pieces of information from an endpoint, may have various legal claims on the information and may have various types of relationships with the (other) overseeing principals. This results in a need for *fine-grained authorization* where distinct authorization rules can be assigned to different parties.

## 2.5. OAuth 2.0

The OAuth 2.0 authorization framework (RFC 6749) was developed to grant third-party applications limited access to Web content without the need to disclose user credentials. It was built for Web applications and is designed to be used with HTTP. The third-party applications are represented by *clients* that request the content stored in *resources* which are hosted by *resource servers*. The overseeing principal of a resource is called the *resource owner*. To gain access to a resource, the client must obtain an *access token* from an *authorization server* and then present this token to the resource server. The authorization server must issue access tokens only if it has an *authorization grant* representing the resource owner's approval. An abstract OAuth 2.0 protocol flow is given in figure 2.1<sup>2</sup>.

In the big Web, OAuth 2.0 is used by service providers such as Google<sup>3</sup>, Dropbox<sup>4</sup>, GitHub<sup>5</sup>, Twitter<sup>6</sup>, and LinkedIn<sup>7</sup>. OpenID Connect provides an identity layer

---

<sup>2</sup>Based on RFC 6749, p. 7

<sup>3</sup><https://developers.google.com/identity/protocols/OAuth2>

<sup>4</sup><https://www.dropbox.com/developers/reference/oauth-guide>

<sup>5</sup><https://developer.github.com/v3/oauth/>

<sup>6</sup><https://dev.twitter.com/oauth>

<sup>7</sup><https://developer.linkedin.com/docs/oauth2>

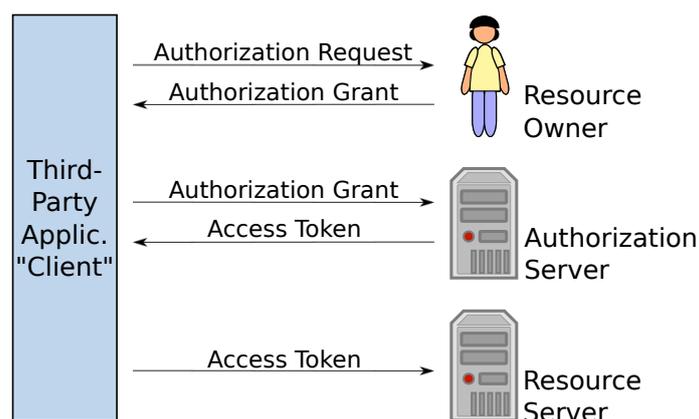


Figure 2.1.: Abstract Protocol Flow in the OAuth Framework

on top of OAuth 2.0: Identity providers offer an authentication service where authorization servers vouch for attributes of end-users such as their name, age or e-mail address (Sakimura et al. 2014).

OAuth 2.0 is a very extensible framework with few required components which are partially or fully undefined in RFC 6749. If used on its own, OAuth 2.0 will likely produce “a wide range of non-interoperable implementations” (RFC 6749, p. 12). The OAuth 2.0 framework does not specify a concrete protocol. Profiles and extensions must be defined to achieve interoperability. A wide range of extensions were developed for OAuth 2.0<sup>8</sup>. There is no overview document available and the documents do not always describe how they are supposed to be used in the OAuth 2.0 framework. That makes it difficult for developers who have no experience with OAuth 2.0 to decide which extensions are needed and how they must be integrated into the framework to achieve a secure authorization process. The complexity and underspecification of OAuth 2.0 leads to vulnerabilities in implementations (Chen et al. 2014; Hammer 2012).

We will analyze the suitability of OAuth 2.0 for the Internet of Things in chapter 11.

<sup>8</sup><https://tools.ietf.org/wg/oauth/>

## 2.6. Conclusion

In this chapter we introduced basic concepts and terminology for authorization in communication scenarios. We showed that authorization mechanisms are required to protect the integrity and confidentiality of data and that they need authentication mechanisms to be effective. We introduced the overseeing principals as the main authorities for their data and devices, who decide which security objectives must be achieved. Securing data in the Internet is a difficult problem since communication channels are assumed to be under the control of attackers who can read and manipulate the traffic.

OAuth 2.0 is used to protect the access to servers in the big Internet. But it is underspecified and requires extensions and profiles to be useful. Its complexity and lack of specification makes it difficult for implementers to develop a secure and interoperable solution. If and how OAuth 2.0 can be used for securing the Internet of Things needs to be analyzed.

### 3. Characteristics of Smart Objects

Smart Objects are typically designed for a specific purpose. In contrast to general-purpose devices that have very similar hardware characteristics, these devices come in various sizes and shapes. The large number of devices demands for low-cost hardware and low operating costs. The devices' hardware will therefore often be customized for their purpose. This leads to limited hardware capabilities in various dimensions, i.e., in one or more of the categories processing power, memory, storage space and (available) energy.

The device constraints limit the communication capabilities and influence the network characteristics. Smart objects are often employed in low-power networks, and the characteristics of these access networks, e.g., their small packet sizes, further limit the networking capabilities of the devices.

Because of their various limitations, some smart objects may be hard-pressed to deploy a full protocol stack as used in the big Internet. Throughout this thesis, we will refer to devices that have difficulties to use unmodified Internet protocols because of hardware limitations as *constrained devices*. We will call devices that are able to use common Internet protocols without modifications *less-constrained devices*.

As described in chapter 1, the success of the Internet and the web can be attributed to certain concepts, most importantly the Representational State Transfer (REST) design principles. We therefore focus our work on applying REST design patterns to IoT security solutions. To achieve this, we use the Constrained Application Protocol (CoAP, RFC 7252) that implements REST for constrained devices. A comparison with other protocols is out of scope for this work.

This chapter will summarize the special characteristics of smart objects and introduce key concepts and protocols for the data transfer in constrained environments. Section 3.1 discusses the various possible hardware constraints that smart objects may have and shows how they influence the ability of the devices to communicate and measure time. In section 3.2, we will give an overview of the life cycle of a smart object. Typical protocols for constrained devices that form the protocol stack are introduced in section 3.3. Section 3.4 introduces the main REST concepts. The Constrained Application Protocol that implements REST for constrained environments is introduced in section 3.5. In section 3.6, we list the requirements for authorization solutions in the IoT caused by hardware constraints and by the protocol stack of smart objects. Section 3.7 summarizes the findings of this chapter.

## 3.1. Device Limitations

Smart objects may have various hardware limitations. One of the limiting factors is the memory and storage space of the devices. RFC 7228 defines three classes of devices (RFC 7228, p. 8). Class 0 devices with a RAM size of under 10 KiB<sup>1</sup> and a storage size of less than 100 KiB are very constrained. They can barely speak IP and will likely not be able to securely speak IP without assistance. Class 1 devices have a RAM size of at least in the order of 10 KiB and a storage size of at least 100 KiB. These will not easily use a full protocol stack with common Internet protocols but are capable to use protocols that were specifically designed for them (RFC 7228, p. 9). Class 2 devices have about 50 KiB RAM and 250 KiB storage space and are likely able to employ a full protocol stack.

### 3.1.1. User Interfaces

Smart objects are not only constrained in terms of storage space. Many of these devices will be used in application scenarios where they communicate autonomously (machine-to-machine) with other devices. Smart objects will often lack

---

<sup>1</sup>KiB: Kibibyte. 1 KiB = 1024 bytes (ISO/IEC 80000-13)

user interfaces and displays to save acquisition cost and energy. Also, smart objects may be installed in places where they are not easily accessible, e.g., in walls. Network communication may be the only means to interact with such a device.

Application scenarios in the IoT that are using machine-to-machine communication differ from the typical Web scenario where a user interacts directly with the browser on her device to communicate with a server in the Web.

### 3.1.2. Energy

Because of the expected large number of devices, their energy consumption influences the operational expenditure. If processors are deployed into every day objects in a large scale, the technology should be energy saving. Constrained devices might be mobile or installed at places where it might be very difficult to lay a cable. The devices will therefore often be battery-powered and only have a limited energy supply. Changing or recharging batteries is very inconvenient and might not be possible at all for some devices and applications.

To save energy, the device's components must be switched off as often as possible (Dutta and Dunkels 2012, p. 72). E.g., the microprocessor sleeps when it is not needed, and is woken up when an event occurred. The approach of saving energy by turning a component's power on and off is called *duty-cycling* (Dutta et al. 2005, p. 3), the duty-cycle being the time where the component is turned on. Duty-cycles are usually stated in percent: a 2% duty-cycle means that the component is switched off 98% of the time.

### 3.1.3. Network Communication

Networking components can only receive packets when they are switched on. Energy-efficient networking is therefore difficult. At the same time, data transmission requires a large amount of power compared to data processing. In the Telos sensor node, an active microprocessor uses 5.40 mW. Writing to flash consumes 45.3 mW, while radio transmission takes 58.5 mW and radio listen 65.4

mW (Dutta and Dunkels 2012, p. 71). Margi et al. show that for a common embedded development board with a MSP430 microcontroller and a CC2420 radio transceiver, the transmission of a 12 byte block costs more than 10 times the energy of encrypting it (Margi et al. 2010).

Anastasi et al. introduce various approaches for conserving energy in networking components (Anastasi et al. 2008). For the *scheduled rendezvous* approach devices switch on their networking component in regular intervals to check if other nodes want to communicate with them. This requires a time synchronization between nodes to ensure that they are awake at the same time. For the *on-demand* approach devices are woken up when they need to communicate. One way to achieve this is to supply devices with a second radio interface that listens on an extra wakeup channel. This device might require less power. In *asynchronous schemes* neighbouring devices have overlapping duty cycles within a specified number of cycles. Dutta and Dunkels call this *sampled* communication, because the receiver in most cases listens for a specified amount of time for communication. Both scheduled rendezvous and asynchronous schemes introduce more latency into the network. While asynchronous schemes do not require synchronization, they are less energy-efficient than scheduled rendezvous approaches.

The goal of all approaches is that networking components sleep most of the time. Dutta and Dunkels state that for an asynchronous scheme they call *sampled operation* it is “possible to operate radios at 1–2% duty cycles”, which will reduce the radio energy consumption by 98–99% (Dutta and Dunkels 2012, p. 79).

The development in wireless networking technologies for accessing the big Internet, e.g., Wi-Fi (IEEE 802.11) aims at increasing the bandwidth to allow users to transport larger amounts of data. For an IoT device, data transmission is very expensive since it costs a lot of energy. Therefore, devices must avoid to transport lots of data. Instead of Wi-Fi, constrained devices often use low-power access networks (see also section 3.3). Low-power radio transmitters are more cost-efficient and require less energy than Wi-Fi. They are therefore a better fit for constrained environments. While 802.11b has a data throughput of 2 to 11 Mbit/s (ISO/IEC 80000-13, see), low-rate wireless personal area networks (WPANs) have a throughput of less than 0.25 Mbit/s (Gutierrez et al. 2001, p. 13). Because of the

low transmission power, many low-power radio transmitters only have a range of about 10 m.

Wireless communication is always prone to packet loss. For constrained environments this problem is made worse because the networking components sleep most of the time.

### 3.1.4. Clocks

Most microprocessors are synchronous circuits: they have a system clock that enables the devices to, e.g., divide processor time, synchronize processes or determine the order and frequency of events. A very basic clock generator consists of an inductor and a capacitor and does provide only an imprecise frequency. Microprocessors used today often have a crystal oscillator with a quartz crystal that creates a signal with a quite precise frequency (Oklobdzija et al. 2003, pp. 9–10). The frequency stability of Quartz crystals depends on the environment temperature. At a temperature of 25 °C, quartz crystals often have a frequency stability between 10 and 100 parts per million<sup>2</sup> (ppm) (see, e.g., Crescent 2017; ECM Electronics 2017). This leads roughly to a deviation of 0.9 to 8.6 s per day. The frequency stability decreases over time; it is often denoted as at most 5 ppm per year. The variation of the clock signal regarding the reference edge of the clock signal, i.e., the time difference between the expected clock tick and the actual clock tick, is described by the *deviation*.

The presence of a clock generator enables the system to use timers and counters that are realized by counting clock ticks. Most operating systems have a function to display the uptime that shows how long the system is already running.

As described above, components can be switched off to save energy. The more components are switched off, i.e., the deeper the sleep, the more energy can be saved. For deeper sleep modes, the system clock itself may be turned off. During such a deep sleep period, counters (including the uptime counter) and timers no longer work and cannot be used to wake up the system.

---

<sup>2</sup>ppm:  $10^{-6}$

Microprocessors may also have a real-time clock (RTC) which is designed to keep the current time. Those clocks have their own integrated battery and are thus able to keep the time even if the system is separated from its main energy source. These clocks are usually are not turned off during sleeping. Although RTCs are not very expensive, low-budget devices do not always have real-time clocks.

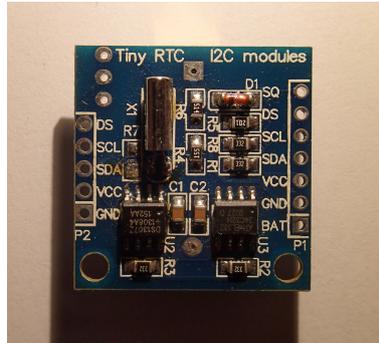


Figure 3.1.: Real Time Clock Chip

For some applications it is necessary that multiple devices have a common timescale. E.g., a measurement server might require to know the exact time when sensor values were measured on various different sensor nodes. Synchronous sleep schemes for communication also require a common timescale (see section 3.1.3). The time on the devices may differ and the clocks may diverge even more over time. Time synchronization solutions aim at synchronizing the local clocks on the devices.

Common time synchronization protocols for the big Internet may not be easily applicable to constrained environments because the constrained nodes may need to turn off their clocks and/or networking components most of the time. Also, time synchronization generates additional traffic which depletes energy resources.

The requirements concerning the preciseness of the clocks may differ for distinct applications. These graduations are out of scope for this work. For this thesis, a rough classification of the devices' capabilities to measure time suffices. A more fine-grained classification of time-keeping strategies for constrained devices can be found in the Terminology for Constrained-Node Networks (draft-bormann-lwig-7228bis-06, pp. 16–18). Since constrained devices may sleep most

of the time, an important distinction is their ability to measure time during sleep. Another significant factor is the use of time synchronization mechanisms. They compensate for the time deviation of the device's clock.

We distinguish between the following capabilities:

- a very basic clock generator (imprecise relative time while awake),
- a crystal oscillator (quartz) that continues counting during sleep,
- a real-time clock and a crystal oscillator, combined with time synchronization (wall-clock time).

## 3.2. Life Cycle

As described above, smart objects will often be integrated into the environment of the user, where they provide their service unseen and autonomously. They will often be installed in places where they are difficult to reach physically, e.g., in walls, on active volcanoes (Werner-Allen et al. 2006), or even in the human body. In these scenarios, they are not easily replaced and more difficult to maintain. Software-updates often need to be performed remotely. Some of the devices may have very long lifetimes.

The life cycle of a smart object can be roughly divided into six phases (see also RFC 8576; Gerdes et al. 2015d): manufacturing, commissioning, operation, maintenance, handover and decommissioning (see figure 3.2). The smart object is created in the manufacturing phase. It is then given to the customer, who introduces it to her network in the commissioning phase. In the operation phase, smart objects fulfill their purpose in life and interact with their principal and with other devices. This phase is sometimes alternated with a maintenance phase if the smart object needs repair, if it needs to be reconfigured or if a software update is required. Some nodes are given away or sold during their lifetime and need to be decommissioned and recommissioned in the handover phase. When the end of the life cycle is reached, the smart object is removed from the network in the decommissioning phase.

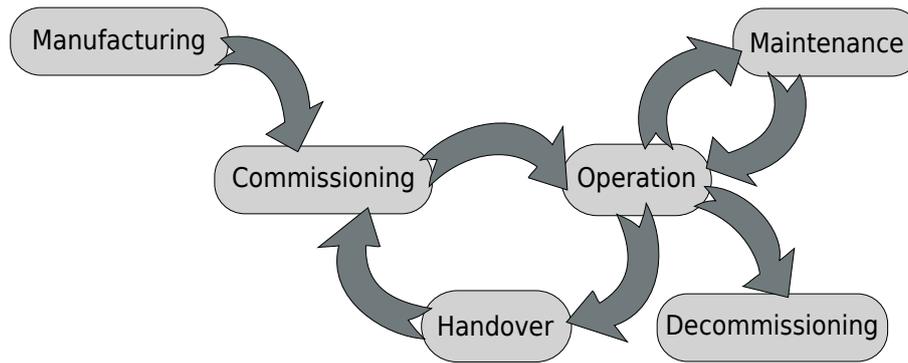


Figure 3.2.: Smart Object Life Cycle

### 3.3. Protocol Stack Elements

Because of the special characteristics and requirements of smart objects, constrained devices will have difficulties to use the Internet protocols that are common in the big Internet. Low-cost, low-power devices with limited system resources benefit from a protocol stack that is designed for their needs.

The main goals for these protocols are saving energy and system resources. Low-power network protocols such as IEEE 802.15.4 aim at supporting devices with very low cost and use low transmit power to save energy. The Internet Protocol, version 6 (IPv6, RFC 8200) has an address space which makes it possible to connect large numbers of devices. IPv6 over low-power wireless area networks (6LoWPAN, RFC 4944) enables the use of IPv6 on wireless constrained devices.

As a data format, the Concise Binary Object Representation, (CBOR, RFC 7049) is not itself part of the protocol stack, but it can be used by protocols to encode content, and was designed for devices with very limited hardware resources.

#### 3.3.1. IEEE 802.15.4

The Institute of Electrical and Electronics Engineers (IEEE) designed the IEEE standard 802.15.4 (IEEE Std. 802.15.4-2006) especially for low-cost systems that are mostly battery-powered. The standard defines the Physical Layer (PHY) and Media Access Control Layer (MAC). It is used by, e.g., Zigbee.

Wireless communication at 2.4 GHz, 915 MHz and 868 MHz is supported. Low power consumption is provided by supporting low duty cycles under 1%, and by using data rates between 20 and 250 kbit/s. IEEE 802.15.4 frames on the MAC sublayer have a maximum size of only 127 bytes.

### **3.3.2. IP**

The Internet Protocol is designed to allow for internetworking. Its main purpose is the addressing of hosts across network borders and to facilitate routing. The Internet Protocol, version 4 (IPv4, RFC 0791) is still widely used in the big Internet, despite the ongoing effort to replace it with IPv6. The main problem of IPv4 is the small address space. The pool of available IPv4 addresses is practically depleted. In 2011, the Internet Assigned Numbers Authority (IANA) assigned the last blocks of IPv4 addresses to the Regional Internet Registries (RIRs) (NRO 2011). The expected large number of devices in the IoT cannot be addressed with IPv4. IPv6 addresses have a length of 128 bit and are therefore significantly longer than the 32-bit IPv4 addresses. IPv6 therefore is much better suited for the Internet of Things.

### **3.3.3. 6LoWPAN**

The goal of the 6LoWPAN standards is to enable low-power, low-rate devices to efficiently use IPv6 (Shelby and Bormann 2009, p. 6). To achieve this, an adaptation layer is defined (RFC 4944) that allows to run IPv6 over IEEE 802.15.4 and similar (e.g., Bluetooth Low Energy, see RFC 7668). One of the main challenges that are targeted by the adaptation layer is that IPv6 requires a minimum packet size of 1280 bytes (RFC 4919, p. 7) while underlying protocols have much smaller frames (see section 3.3.1). 6LoWPAN therefore provides fragmentation and re-assembly below IP.

Uncompressed IP headers leave little room for application payload which might lead to excessive fragmentation and reassembling. To mitigate this problem, 6LoWPAN defines a header compression mechanism (RFC 6282). Figure 3.3 shows the resulting protocol stack for 6LoWPAN.

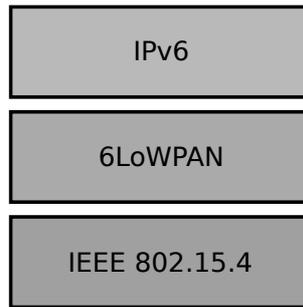


Figure 3.3.: 6LoWPAN Protocol Stack

### 3.3.4. Concise Binary Object Representation

The Concise Binary Object Representation, (CBOR, RFC 7049) is a data format that is designed to support systems with very limited memory and processing power (RFC 7049, p. 4). A primary design goal of CBOR is to allow for encoders and decoders with a very small code size. Also, CBOR is designed for compact serialization and thus allows for small message sizes.

CBOR defines eight major types for data items including unsigned integer, byte string, arrays and maps. The major type is given in the initial byte of the data item. The length of a data item either derives from the data type or is specified by additional bytes that follow the major type. An exception are indefinite-length arrays and maps; their end is indicated by a break stop code.

A special characteristic of CBOR is that it allows for the tagging of items: a data item can be preceded by a tag to give it additional semantics. Examples for tags are standard date/time, URI, and positive bignum for large numbers. The tags indicate how the data item is to be interpreted.

## 3.4. Representational State Transfer

As described above, smart objects can be expected to benefit from Web-like communications patterns. According to Fielding and Taylor, the success of the World Wide Web can be attributed to a large part to its software architecture (Fielding

and Taylor 2002, p. 115) which provides scalability of component interactions, generality of interfaces and independent deployment of components. The Representational State Transfer (REST) architectural style (Fielding 2000) describes an abstract model of the Web architecture.

According to Berners-Lee, “the goal of the Web was to be a shared information space through which people (and machines) could communicate” (Berners-Lee 1996). To provide easy access to the content, REST uses a simple *client-server* architecture. The client initiates the communication by sending a request to the server that hosts the content. The server can provide its content to several clients and clients can communicate with various servers. The components are *loosely coupled*: they can evolve freely as long as the interfaces remain the same.

To allow for loosely coupled components, the REST model focuses on describing interfaces between components instead of the components themselves (Fielding and Taylor 2002, p. 116). To achieve *uniform interfaces*, REST demands interfaces to have four characteristics: there are resources which are identifiable, resources are manipulated by representations, messages are self-descriptive, and hypermedia is used as “the engine of application state” (Fielding 2000, p. 82).

Content is stored in *resources*. Each resource has a *resource identifier* by which it can be accessed and manipulated. The *resource state* may change over time (Fielding 2000, p. 89). The state of a resource at one point in time can be captured in a *resource representation*. For interaction with a resource, resource representations containing the current or intended state of the resource are exchanged between components (Fielding and Taylor 2002, p. 126).

To make information widely available, the process of accessing resources on the servers must be scalable. To achieve this, REST aims at eliminating any need for the server to store information about the client between requests (Fielding 2000, p. 100), thereby enabling the server to save system resources. Fielding calls this the *stateless constraint* (Fielding 2000, p. 78). Application state is only stored by the client. All information required by the server must be transmitted in the messages (Fielding 2000, p. 98): the request must contain the resource identifier, the methods, i.e., the actions that are attempted to be performed, and the media types that define the contents.

A core characteristic of the Web is *hypermedia* which provides application control information together with the content (Fielding 2000, p. 68). Information items of all kinds of types and from various sources can be represented and inter-linked. Relationships between contents from different sources are represented by hyperlinks. Since the same simple interface is used regardless of the information source, REST-based systems have a low entry-barrier (Fielding 2000, p. 67).

Another key concept in REST is *caching*; cacheable data can be stored for a certain period of time for later use, which might reduce network traffic and improve efficiency, scalability and user-perceived performance (Fielding 2000, p. 80). However, caching has the disadvantage that cached data might become stale and differ significantly from the resource state on the server. Data in responses to a request must be labeled as cacheable or non-cacheable.

REST distinguishes two types of intermediaries: proxies and gateways. Proxies are selected by the clients (Fielding 2000, p. 97). They forward requests, after translating them if necessary, to servers. Gateways or reverse proxies are provided by the network or the origin server, and act as normal servers toward the client. They accept client requests, translate them if necessary, and forward them to servers. Intermediaries can act as caches, resolvers or tunnels (Fielding 2000, p. 93). Caches can be used by clients or servers to store cacheable responses. Resolvers translate resource identifiers if necessary. Tunnels forward communication across connection boundaries such as firewalls.

### **3.5. Constrained Application Protocol**

As described above, some of the IoT devices will have difficulties to use protocols and mechanisms that are common in the big Internet. Low-power protocols such as IEEE 802.15.4 save system resources and energy, but employ small frame sizes for data transmission. Higher level protocols must not produce much overhead to avoid fragmentation.

HTTP is an implementation of REST concepts. Together with the Uniform Resource Identifier (URI) it defines the Web's generic interface (Fielding 2000, p. 107).

HTTP was designed without constrained environments in mind. It requires a relatively large amount of storage space and uses up a significant amount of network resources (Bormann et al. 2012, p. 64). It therefore is not a good fit for constrained environments.

The IETF Constrained RESTful Environments (CoRE) Working Group<sup>3</sup> designed the Constrained Application Protocol (CoAP) to implement REST specifically for constrained environments. CoAP is now established for Web of Things applications. It is the application layer protocol in specifications from, e.g., OMA Specworks<sup>4</sup> (Open Mobile Alliance 2019, p. 3) and the Open Connectivity Foundation (Open Connectivity Foundation 2019, p. 6), and is part of the IoT operating system RIOT (Hahm et al. 2014, p. 330). Members of the OCF are companies such as Cisco, Samsung, Microsoft, Intel and LG. The Lightweight Machine-to-Machine (LwM2M) specification defined by OMA Specworks is deployed in products from vendors such as ARM, Qualcomm, Ericsson and Huawei.

CoAP translates to HTTP (see RFC 8075) and thus allows constrained devices to integrate with the big Internet. The protocol can be used by constrained and less-constrained devices. To communicate with devices that do not speak CoAP, intermediaries can be used: they can speak CoAP on one side and HTTP on the other (Bormann et al. 2012, p. 64). CoAP uses small headers: a typical request has a header size of only 10 to 20 bytes (Bormann et al. 2012, p. 64). According to Colitti et al. the number of bytes used by HTTP transactions is about 10 times larger than for CoAP transactions (Colitti et al. 2011). CoAP messages should fit into an IP datagram which defines the upper bound to the message size (RFC 7252, p. 25). CoAP recommends 1152 bytes as upper bound to the message size and 1024 bytes as the maximum payload size. For larger messages, block-wise transfer (RFC 7959) can be used; multiple blocks of information from a resource representation can thereby be transported in multiple request-response pairs. Servers can handle each transfer separately, in compliance with the stateless constraint (RFC 7959, p. 1). CoAP therefore has much less overhead, reduces fragmentation and network traffic.

---

<sup>3</sup><https://tools.ietf.org/wg/core>

<sup>4</sup>In 2018, the Open Mobile Alliance (OMA) and the Ipso Alliance joined to form OMA Specworks.

One of the differences to HTTP is that CoAP defines the User Datagram Protocol (UDP, RFC 0768) as Transport Layer protocol which is better suited for constrained environments (Colitti et al. 2011, p. 2) than the Transmission Control Protocol (TCP, RFC 0793)<sup>5</sup>. Since UDP offers no delivery guarantee or duplicate protection (RFC 0768, p. 1), CoAP defines a simple message layer that can provide additional reliability, if necessary (RFC 7252, p. 10). Messages that are to be transported reliably must be marked as confirmable (CON). The receiver must react to this message; it must either be acknowledged or rejected. To achieve more reliability, the sender retransmits the message if it does not receive an acknowledgement. Messages that do not require an acknowledgement are marked as non-confirmable (NON). In addition, CoAP messages have a message ID to enable endpoints to identify duplicates and to match acknowledgements and reset messages to confirmable and non-confirmable messages. Figure 3.4 shows the CoAP layers.

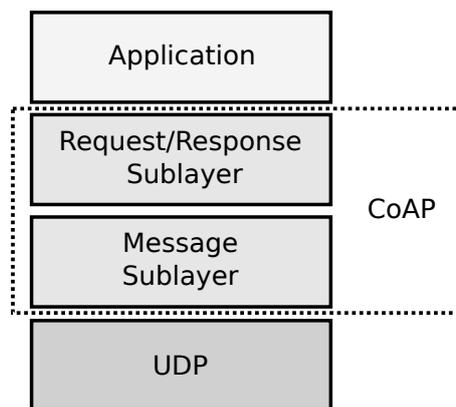


Figure 3.4.: CoAP Layers (based on RFC 7252, p. 10)

Since CoAP implements the REST architectural style, it uses a simple client-server architecture and focuses on defining a generic interface. CoAP uses a similar request/response model as HTTP. Content is represented by resources which are identifiable by URIs. CoAP uses similar URIs as HTTP (RFC 7252, p. 59). An absolute URI (RFC 3986) comprises a scheme, an authority part, and a path and may additionally contain a query. The scheme specifies the syntax and semantic

<sup>5</sup>Since existing enterprise infrastructures might have problems with UDP, e.g., packets might be blocked by firewalls, the CoRE working group also designed a solution for CoAP over TCP (RFC 8323).

of the URI. The authority part identifies the server while the path component defines the resource on the server. A relative reference must be used in combination with a base URI (RFC 3986, p. 28).

CoAP messages are self-descriptive; Requests contain the identifier of the resource, the payload and its media type, the request method and optional metadata about the request (RFC 7252, p. 31). Media types describe the content in the payload. While HTTP defines content types that describe only the media types, CoAP specifies *content formats* that represent the media types together with their representation on the wire. CoAP allows for content negotiation, i.e., the client can specify which content-format it prefers. The content-format comprises media types such as `text/plain`, `application/link-format` and `application/json`. The list of CoAP Content Formats<sup>6</sup> is managed by the IANA; additional media types can be added if necessary. Request methods define which action is to be performed on the resource. Like HTTP, CoAP uses the basic request methods `GET`, `PUT`, `POST` and `DELETE`. Responses are matched to requests with the assistance of tokens. The tokens are specified by the client in the request and reflected by the server in the response. The type of the response is identified by the response code. CoAP response codes base on HTTP response codes, but are represented in a single byte (Bormann et al. 2012, p. 64). E.g., if a resource in a `GET` request does not exist, the CoAP response code is 4.04 instead of 404 as in HTTP. Responses can be transmitted in the acknowledgement message as a *piggybacked response*. If this is not possible for some reason, e.g., because retrieving the resource representation takes too much time, the response is transmitted in a *separate response*. Responses to non-confirmable messages are always transmitted separately. Both requests and responses might contain options. Each option in a message comprises the option number that indicates the option type, the length of the option value and the value itself. Options are only added when needed, which helps to make CoAP messages smaller.

CoAP allows for the caching of responses to reduce the response time and network bandwidth consumption (RFC 7252, p. 42). It uses the Max-Age option to indicate how long a response may be cached before it is no longer fresh.

---

<sup>6</sup><https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>

### 3.5.1. Observe

In the REST model the client sends requests concerning a resource to a server. This approach helps a client that is interested in viewing or manipulating the current resource state, but is less useful if a client is interested to always have the most recent representation over a period of time: since the client does not know when an update is available, it is required to poll repeatedly which may generate significant overhead.

The observer design pattern (Gamma et al. 1994), also called publish-subscribe, describes how a *subject* can have several dependent *observers* that are notified if the subject changes its state. The observe protocol (RFC 7641) implements this design pattern for CoAP while keeping the design properties of REST (RFC 7641, p. 4). In the CoAP context, subjects are resources hosted by CoAP servers. To become an observer, clients must register their interest in a resource. Clients will be notified about updates to the resource as long as the server can determine that the client is still interested in it. Clients can actively cancel their registration. Also, they are automatically deregistered if a timeout occurs after several unsuccessful transmission attempts (RFC 7641, p. 6).

### 3.5.2. Discovery

Devices in machine-to-machine communications need to discover the services provided by other devices to be able to communicate. In CoAP, this means discovering resources on servers. Clients can either learn the URIs that identify resources in the namespace of a server, or they can use multicast CoAP to find CoAP servers (RFC 7252, p. 64).

If clients already know the IP address of the server, they can discover its resources by sending a GET request with the URI `/.well-known/core` to the server (RFC 6690, p. 4). The response is encoded in the CoRE Link Format (RFC 6690) which is used by CoAP servers to describe resources, their attributes, and relationships between links (RFC 6690, p. 1). Clients can also search for resources with certain attributes using query filtering on `.well-known/core` (RFC 6690, p. 12). They

may, e.g., use the resource type attribute to discover only resources of the outdoor temperature type. Additionally, clients may use multicast service discovery to find a certain service (RFC 7252, p. 65).

Direct discovery of resources may not always be easy to accomplish in constrained environments, e.g., because devices sleep most of the time (see also section 3.1.4). Instead, resource directories, i.e., special servers that host resource descriptions, can be deployed (draft-ietf-core-resource-directory-25). Servers register their resources with these resource directories which can then provide the resource descriptions to others.

### 3.5.3. Security

CoAP specifies a Datagram Transport Layer Security (DTLS, RFC 6347) binding with four security modes (RFC 7250, p. 68). In the NoSec mode, DTLS is disabled and an alternative security mechanism must be deployed if necessary. CoAP nodes in the PreSharedKey mode use symmetric pre-shared keys (RFC 4279) to communicate. The nodes must have obtained the cryptographic keying material and must know with which nodes these keys are to be used.

Both the Certificate mode and the RawPublicKey mode use asymmetric keys. In the Certificate mode, nodes have X.509 certificates. To validate certificates of others, nodes require a list of root trust anchors. The RawPublicKey mode provides an alternative to the certificate mode. It uses asymmetric keys without certificates called raw public keys (RFC 7250). These keys must be validated by an out-of-band mechanism<sup>7</sup>. CoAP with DTLS provides for authentication but lacks a mechanism for authorization.

## 3.6. Requirements Resulting from Hardware Limitations

Authorization solutions for the IoT must consider the limitations and characteristics of the participating devices. Not all smart objects will be constrained in

---

<sup>7</sup>We analyze the characteristics of symmetric and asymmetric algorithms more closely in section 10.1.

the same way; devices with various different hardware limitations may need to communicate with each other. Protocols for the IoT need to consider the various dimensions of possible constraints.

As described in section 3.1, class 1 devices may have difficulties to use common Internet protocols but are able to use protocols that were specifically designed for them. This dissertation postulates that constrained devices that are class 1 need to be enabled to enforce their owners' security policies (**Class 1 Req**). Class 0 devices will likely not be able to securely speak IP without assistance. Special means are necessary to support them. Protecting class 0 devices is not in the focus of this dissertation.

Since the IoT is expected to have a much larger number of devices, authorization solutions must be scalable to be useful (**Scalability Req**). Also, they must consider that smart objects often only have very limited energy (**Energy Req**). Since sending and receiving is particularly expensive, traffic should be reduced and unnecessary overhead should be avoided.

Constrained devices may only have a limited ability to measure time (see section 3.1.4). A solution that is applicable to constrained devices needs to address cases where no real-time clocks are available and time synchronization may be very difficult (**Clocks Req**).

The environments where the devices are deployed often have special characteristics; e.g., the devices will often have no user interfaces and displays and need to communicate autonomously. We will analyze the application scenarios and resulting requirements in chapter 4.

As described in section 3.4, the success of the Web is attributed mainly to its software architecture. The IoT can be expected to also benefit from a RESTful architecture style. The authorization solution therefore should use uniform interfaces and REST design patterns (**REST Req**). CoAP was specifically designed to implement REST for constrained devices and is suitable to fulfill the **REST Req**. It also facilitates achieving the **Class 1 Req** and the **Energy Req**. Blockwise transfer is required for the transmission of larger messages. Observe can further reduce network traffic since it relieves the client from continuous polling. Authorization

solutions for the IoT that use REST therefore should use CoAP (**CoAP Req**) as the transport protocol and support CoAP blockwise transfer. Where necessary, the solution should use observe (**Observe Req**). Service Discovery reduces the configuration effort for overseeing principals.

The following list summarizes the requirements that derive from the hardware limitations that must be considered by authorization solutions:

**Class 1 Req:** constrained devices that are at least class 1 should be supported by the authorization solution.

**Scalability Req:** the authorization solution must be scalable: the authorization solution should not limit the number of devices that an endpoint can communicate with, and should not inflict unnecessary overhead.

**Energy Req:** energy must be saved when possible. In particular, network traffic must be reduced and unnecessary overhead must be avoided.

**Clocks Req:** the authorization solution should support devices that do not have a real-time clock and that have difficulties to perform time synchronization.

**REST Req:** the authorization solution should implement a RESTful architecture style and REST design patterns.

**CoAP Req:** CoAP should be used as a transfer protocol for authorization messages.

**Observe Req:** the authorization solution should support CoAP observe where necessary.

### 3.7. Conclusion

This chapter introduced special characteristics of constrained devices that are caused by their hardware limitations. We showed that constrained devices benefit from protocols that were especially designed for their needs. Low-power protocols such as IEEE 802.15.4 are designed for low-cost devices and have only little transmission capacity. IP as the unifying factor enables the IoT to not become a silo solution but to be integrated into the big Internet.

IoT devices can benefit from the communication patterns that made the World Wide Web popular by implementing the REST architectural style. The Constrained

Application Protocol implements REST for constrained environments, and provides small packet sizes and little overhead and can be translated to HTTP. The observe protocol further reduces the strain on constrained networks by enabling clients to register for resources they are interested in and getting updates without the need for continuous polling.

Security protocols must consider the limitations and characteristics of constrained devices and their environments. They must be flexible to support the wide variety of hardware. We provided a list of requirements deriving from the hardware limitations that an authorization solution must consider. CoAP is a good fit for constrained environments since it considers their special requirements. Security protocols that use CoAP must be careful to sustain the CoAP design and the REST architectural style.

*It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.*

– Sir Arthur Conan Doyle (Sherlock Holmes)

## 4. Authorization in Constrained Environments

Endpoints that participate in the Internet are exposed to threats as we describe in section 2.3; security solutions for the IoT must be prepared to deal with the usual security problems. Authorization assists with achieving integrity and confidentiality of data (see also section 2.1) and is therefore also necessary in the IoT. However, some smart objects may have difficulties to use authorization protocols that are common for the big Internet. As described in chapter 3, devices that participate in the IoT come in various sizes and shapes and will often be limited in terms of processing power, storage space, energy and networking capabilities. A lot of effort needs to be made to reduce code size and packet sizes and to make code and protocols more energy-efficient.

However, making the code smaller is only part of the changes that are required to address security problems in the IoT. Smart objects can be deployed in new applications that have not been possible before. This influences how the devices communicate with each other. We will analyze typical application scenarios and discover how the special characteristics of IoT devices and the environments where they are used influence the authorization process. In our analysis we will identify the special challenges for authorization solutions in the IoT.

The chapter is organized as follows: Use cases that are representative for authorization in the IoT are introduced and analyzed in section 4.1. Section 4.2 discusses special aspects concerning the scenarios and environments that constrained devices are used in. The requirements for an authenticated authorization solution are derived in section 4.3. Our findings are summarized in section 4.4.

## 4.1. Use Cases for Authorization in Constrained Environments

The IETF Authentication and Authorization in Constrained Environments (ACE) working group<sup>1</sup> aims at standardizing authorization solutions for constrained environments. In the context of this working group, we worked with leading companies and researchers in the IoT area to collect and revise use cases that are representative for authorization scenarios in the Internet of Things. The result of this work is **RFC 7744**. It describes the application environment and lists the authorization problems that arise for OVPs. The goal of this work is to provide a guideline for the design of authentication and authorization solutions in constrained RESTful environments. From the application areas that are described in the document, we pick four that are suitable to describe typical characteristics of IoT environments and to derive the requirements that must be met by an authorization solution: Container Monitoring, Sports and Entertainment, Building Automation and Smart Metering.

The **main authorization directive** defined in section 2.2 states that the decisions of the overseeing principals as the main authorities for their data and devices must be followed. By analyzing the use cases, the OVPs' requirements for an authorization solution are derived. The first step is to identify the OVPs in the use case. We then must determine the OVPs' security objectives. They are required for defining the authorization rules. To be suitable for a use case, a security solution must be able to assist in achieving its security objectives. To determine the security objectives, the data that needs to be protected and the kind of protection it requires must be identified. The goal of a *protection requirements analysis* is to determine the valuable data in a system and to classify the impact of security breaches. The result of the analysis is the required level of security for each security objective. We provided an exemplary analysis for constrained environments in earlier work (Gerdes and Bergmann 2013). However, protection requirements depend on the characteristics of the environment (Sandhu and Samarati 1994, p.41). A detailed protection requirements analysis of all use cases goes beyond the scope of this dissertation. We therefore only give a rough overview of the types of application data and required security objectives; we mainly aim at de-

---

<sup>1</sup><https://tools.ietf.org/wg/ace>

iving more general characteristics such as the required level of granularity that the authorization solution must provide (see also section 2.4), or similarities and distinctions between the various overseeing principals' security objectives in a certain use case.

The devices themselves and the environment they are deployed in have special characteristics that are independent of the security objectives. E.g., devices are often used in scenarios where their OVPs are not able to intervene and they need to act autonomously. Authorization solutions must consider those characteristics and meet the resulting requirements. We therefore also aim at identifying such special characteristics of the environment and the scenario.

Certain requirements can be found in multiple or even all use cases. To improve the readability, we only mention them once. A list of requirements for every use case can be found in RFC 7744.

#### **4.1.1. Container Monitoring**

The container monitoring use case describes how constrained devices of various overseeing principals are used to monitor and handle perishable goods during transport and storage. Sensors provide information about the goods' environment such as temperature, humidity and gas content. The goods require constant temperature and proper ventilation to avoid spoiling.

The use case in this application area is about a fruit vendor that sells bananas to a German supermarket chain. The fruits are transported in containers via ship from Costa Rica to Germany by a transport company, and stored in a ripening facility until they can be sold. The banana boxes are equipped with sensors that constantly monitor the state of the bananas during shipment and storage. To provide an optimal environment, the sensors also communicate with the respective climate-control system. Since direct communication between nodes is difficult due to the high water content of the bananas, messages are forwarded over multiple hops. During shipment, the sensors might not always be able to reach servers in the Internet.

The transport company might transport goods of multiple customers. The transloading personnel must be able to locate the goods of a specific customer but is only allowed to access logistic information, and only for the time of the transloading.

The sensors in the banana boxes belong to the fruit vendor, who therefore is their overseeing principal. The OVP of the climate-control system in the containers is the respective container owner. The third OVPs in this scenario is the transloading company that oversees the devices of the transloading personnel.

The banana sensors must communicate with various endpoints of the other overseeing principals: the container owner's climate control system requires the sensor information to adjust the temperature and ventilation. The transloading personnel must be able to access logistic data. The fruit vendor must know the state of the goods. The OVPs require the availability of the respective data.

The differences in the data that other endpoints require from the banana sensors influence the required level of authorization granularity (see also section 2.4). The principle of *Least Privilege* states that each entity must only have the privileges that are necessary for its work (Saltzer and Schroeder 1975, p. 1282). Fruit vendors therefore need fine-grained authorization: they must be able to define distinct authorization rules for their own devices, the climate control system and the devices of the transloading personnel (**Fine-Grained Authorization Req**).

We will now determine if the overseeing principals have similar or distinct security objectives. Goods may be specifically targeted if their owner is known, e.g., by competitors. To avoid this type of threat, fruit vendors require the confidentiality of logistic data.

Since the climate-control system adapts the temperature and ventilation according to the sensor values, the container owner and the fruit vendor require the integrity and authenticity of this data. Attackers might try to send forged sensor data to manipulate the climate-control system. Without protection, the fruit vendor might lose its goods and the container owner might lose the trust of its customers.

The personnel of the transloading company must be able to determine the owners of the goods. Therefore, it must be able to access administrative data on the

sensors. If attackers can manipulate the sensor data, the transloading company may assign the goods to the wrong customer. The transloading company and the fruit vendor therefore require the integrity and authenticity of the administrative data.

As we can see, the overseeing principals have similar security objectives in many cases. But they may have different ideas which data might be accessed. E.g., the fruit vendor has an interest to keep the state of the goods secret and therefore requires the confidentiality of this data, while this is not especially important to the transloading company. An overseeing principals may not know all security objectives of all other overseeing principals. Also, an OVP may not always have sufficient incentive to protect other OVPs security objectives, i.e., they do not suffer (enough) if the security fails (see also Anderson and Moore 2006). It is therefore important to determine and enforce the security policies of all participating overseeing principals (**Distinct Interests Req**).

Overseeing principals will often not be able to manually intervene in the authorization process. The fruit vendor is not present during shipment, and manually controlling hundreds of sensors in a banana container is not a feasible solution. The climate control system might not have a user interface and therefore must be controlled remotely. Container owners might also not be present at the time of the communication, or it might be inconvenient for them to manually intervene in the authorization process. In cases where OVPs are not present and not able to intervene, their authorization rules must still be enforced by their devices (**Absent Principal Req**).

The devices may not always be able to reach endpoints in the Internet and still need to securely communicate locally (**Offline Req**).

Some communication partners only require temporary authorization; e.g., the sensors need to communicate with the devices of the transloading personnel only during transloading. The climate-control system in the container must no longer communicate with the sensors in the banana boxes when the bananas are sold. Permissions must only be granted while they are needed. (**Temporary Permissions Req**).

Messages may need to be transported over multiple hops. The relaying endpoints are not necessarily authorized to read or manipulate the content of messages. In this case, the overseeing principals' security objectives must still be achieved (**Multiple Hops Req**).

The devices that participate in the processing of the bananas might be compromised, e.g., when they are stolen. In this case, their permissions must be revoked and the authorization rules of their communication partners must be updated as soon as possible (**Revocation Req**).

#### 4.1.2. Sports and Entertainment

Constrained devices enable a new area of applications for leisure-time activities. Because of the small size and weight of the devices, they can be integrated into fitness equipment, games and even clothes, thereby enhancing the functionality of these products. E.g., sports equipment can measure speed and movement and thus assists athletes in enhancing their performance.

The devices will often carry very personal, in some cases even sensitive medical data. The continuous monitoring of the users' personal area allows the creation of behavioral profiles. If unprotected, attackers can gain deep insight into the users' lives, even more so if the data of multiple devices is aggregated.

In the sports and entertainment use case, two friends, Jody and Lynn, want to interconnect their training equipment. Lynn has a fitness watch that can measure her pulse, shows speed and distance and keeps track of her training program. Jody has smart running shoes with pressure sensors in the sole which help her improve her posture and running style and count her steps. Lynn lends her watch to Jody for the afternoon, but does not want Jody to access and configure her personal data. Jody wants to couple the watch temporarily to her online fitness account while she is using it. After an hour, she gives the watch back to Lynn and the girls terminate the connection between their devices.

In the sports and entertainment area, usability is especially important. Overseeing principals do not want to spend much time on the configuration of their devices and often only have a limited technical background. Users will often want

their devices to spontaneously interact with others. It must therefore be possible for the OVP to dynamically update authorization rules when needed (**Dynamic Updates Req**). Also, the configuration of authorization rules must be simple and not require much effort (**Simple Configuration Req**). Overseeing principals benefit from preconfigured authorization rules that grant certain access permissions to endpoints with certain attributes (**Preconfiguration Req**). The configuration effort at the time of access can thereby be reduced. The overseeing principals want their devices to interact but don't want to hand over the authority for their devices (**Separate Authority Req**).

### 4.1.3. Building Automation

Constrained devices can also be deployed in buildings for commercial use such as office buildings or shopping malls. Common applications are, e.g., illumination; heating, ventilation and air conditioning (HVAC); or fire alarms. Often a facility-management company is contracted for managing the infrastructure.

Buildings are often inhabited by multiple companies. Usually, each of them has its own area for exclusive use, while other parts of the building, such as the entrance area, are often shared. The companies want to control parts of the infrastructure such as illumination or HVAC exclusively for their rooms. Other parts such as the fire alarm system must be operated together.

In this scenario, we will focus on describing the lifecycle of devices (see also section 3.2) and the resulting requirements. The scenario starts with the commissioning phase. The manufacturing phase is therefore omitted.

The commissioning phase of building automation devices often starts during the construction of the building when the system's components are installed. After the installation is complete, the devices are configured and added to the administrative domain of the building by the commissioning team. The network infrastructure is rolled out later to avoid damage to the networking and computing equipment. Therefore, the devices may be in various network islands with no connectivity between them. The commissioning team must then perform the

configuration locally. Some devices, e.g., certain light points, are grouped so that they can be controlled together. The smart objects may need to interact with devices from other manufacturers.

When the installation of the building automation system is completed, the operation phase is entered. The companies that participated in the installation no longer need to communicate with the devices. The facility manager is now responsible for the system and must ensure that the needs of the building's residents are met. If devices are controlled together, they may benefit from using a group communication mechanism.

The lighting in the offices is controlled by presence sensors. Employees can also manually change the brightness and color of the lights in their offices with switches or handheld controllers. When the employees leave their offices at the end of the day and their presence is no longer detected, lighting is dimmed or switched off.

The maintenance phase starts when problems with the building automation system arise, e.g., configuration problems, out-of-date software or broken hardware. For profound changes the facility manager hires a commissioning company. The maintenance staff must be able to update the software on the devices, change their configuration, and remove and add endpoints in the system.

In the decommissioning phase, devices are removed from the system because they are no longer required or become unsuitable for their service since they are broken or need to be replaced by newer or better devices. A device may then reach the end of its lifecycle and be discarded. In some cases, devices are sold or given away so that they can be deployed elsewhere by another overseeing principal. They are then decommissioned and recommissioned in the handover phase.

By analyzing this scenario, we can derive the requirements. In some cases, access is only granted in defined time periods. Also, a device's permissions may depend on its location (**Context-Based Authorization Req**).

If a group of devices has similar functionality and is controlled together, it may be useful to deploy a group communication mechanism. Devices must be enabled to

communicate securely, even when they use group communication (**Group Communication Req**).

The facility manager must be able to delegate their own permissions to the commissioning company to allow them to perform maintenance (**Permission Delegation Req**). Software or firmware updates must only be accepted by the devices if the authorization of their provider is validated. The commissioning company only requires access to the data during maintenance (**Temporary Permissions Req**).

In the decommissioning phase, nodes that are no longer needed must be securely removed from the system (**Decommissioning Req**). Sensitive data, e.g., system credentials, must be erased from the devices. The authorization rules for the devices and their communication partners must be updated accordingly (**Revocation Req**).

The handover phase comprises a decommissioning and a commissioning. The old overseeing principal ceases to be the authority for the device: the complete control of the node is given over to the new overseeing principal (**Handover Req**). Authorization rules for the old network must be removed; new authorization rules must be provisioned to the device.

The tenants that occupy parts of the building want the nodes in their subsystem to communicate with those of others, but still want to be in charge of their own devices, i.e., define authorization rules for them (**Separate Authority Req**).

#### **4.1.4. Smart Metering**

Smart metering systems are designed to autonomously measure consumer consumption of utility services such as electricity, water or gas. The utility company that provides the service needs the measured data for accounting. The smart metering equipment is installed on the premises of the consumers and is often battery-powered. In some cases, the metering equipment belongs to service operators that maintain the meters.

Systems for water and energy distribution belong to the critical infrastructures. Damages to a critical infrastructure, i.e., their destruction or disruption, have a significant impact on the well-being of people (European Council Directive 2008/114/EC, Article 2a). Attackers might use smart meters to attack these infrastructures. E.g., if smart meters that can be used to remotely switch off the power are deployed in millions of households, attackers might use them for a large-scale attack on electrical power supply (Anderson and Fuloria 2010, p. 98).

One interesting characteristic of this use case is that the involved stakeholders have very distinct interests. The utility company needs the data for accounting and thus requires the authenticity and integrity of the accounting data. Attackers might try to tamper with the meters to steal from the utility provider (Krebs 2012; McDaniel and McLaughlin 2009). Consumers do not want to be overbilled but they would profit from manipulating the meter to reduce their bill.

Only authorized entities must be allowed to send control data to the meters; the integrity and authenticity of this data is therefore also required.

The consumption data reveals details about the consumers' daily lives; e.g., Non-intrusive Load Monitoring (NILM) techniques can be used to derive detailed information about operated devices from measuring electrical load solely at the utility service entry (Leeb et al. 1995; Laughman et al. 2003). This allows consumers to save energy, but also discloses details about the consumers' habits: the times when they are at home, when they get up etc. Consumers therefore have an interest in the confidentiality of this information.

The smart metering equipment is installed in an environment where it may be physically accessed by attackers. Without protection, the data on a captured device might be used to attack the metering infrastructure (**Capture Req**).

## 4.2. Discussion

The use cases presented in section 4.1 provide some information about authorization in constrained environments but do not discuss every detail. Some aspects depend on the specific implementation of the use case.

Devices that participate in the IoT come in various sizes and shapes and may have various limitations. Not all of them will necessarily be constrained devices. Section 4.2.1 discusses the necessity for constrained to constrained and constrained to less-constrained communication. Section 2.4 introduced levels of granularity for the authorization. Fine-grained authorization may not always be required in IoT applications. Section 4.2.2 analyzes the required level of granularity for applications. One special characteristic of constrained applications is the need to communicate autonomously. How this influences the authorization is analyzed in section 4.2.3. Constrained devices are particularly prone to Denial of Service (DoS) attacks as we will discuss in section 4.2.4. Applications may be designed to require access to servers in the Internet, particularly cloud servers. Section 4.2.5 describes the consequences of this approach. One approach to secure smart objects is the use of firewalls. Unfortunately, this method has serious problems that we will discuss in section 4.2.6. The use cases indicate that there are often several stakeholders involved in an IoT application. The need to consider the decisions of all involved overseeing principals is discussed in section 4.2.7.

#### **4.2.1. Constrained and Less-Constrained Communication**

The use cases do not specify in detail which devices are constrained and which are not. E.g., the battery-operated sensors in the banana boxes will likely be constrained devices, the cooling system might be constrained but might also be a less-constrained device. The fitness watch and the running shoes in the sports and entertainment use case will likely both be constrained. Light bulbs and light switches in the building automation use case might also be constrained devices.

Constrained devices will often need to communicate with less-constrained devices. Some applications require that one communication partner is more powerful. Users will often use a general-purpose device (e.g., their smartphone) to communicate with their constrained devices. Therefore, authorization protocols must be able to support constrained to less-constrained communication (**C2L Req**).

Less-constrained devices will in most cases be more expensive and consume more energy. It is therefore beneficial for overseeing principals to use constrained devices where possible. Less-constrained devices will usually be able to perform

tasks that a constrained device can perform but not vice versa. The authorization solution must be designed for constrained to constrained communication to fully support interconnected smart objects in a true Internet of Things (**C2C Req**).

#### **4.2.2. Granularity**

If a server has only a single Web resource or the overseeing principal has only a single security objective, binary authorization (see also section 2.4) is sufficient. In these cases, it must be possible to avoid unnecessary overhead caused by the authorization solution.

For multiple resources or a single resource that can be accessed in different ways (e.g., be read and written) the authorization rules for distinct overseeing principals may differ. In this case, fine-grained authorization is required.

In addition to the application data, devices also require configuration data that is necessary for the proper operation of the node and for communication within the network, such as gateway addresses and human readable node identifiers (see also Gerdes and Bergmann 2013, p. 234). Changes to the configuration data often require special permissions that might, e.g., only be granted to overseeing principals.

Also, firmware and software on the device need to be updated from time to time to fix software defects and address newly discovered vulnerabilities. This type of data has special protection requirements; attackers might try to upload their own programs and thereby, e.g., add backdoors to the system. Firmware and software updates therefore must only be performed with special authorization.

Cryptographic mechanisms that are required for authorization and secure communication in most cases require a secret that is stored on the device. These secrets are very sensitive data and must only be changed with special permissions.

In the IoT it might not always be possible to locally change data on the device, e.g., by using a user interface or a USB cable. Smart objects might have no user interfaces or might be installed in places where they are difficult to reach. Also, the

large number of devices might make it difficult to access each of them manually. If all data must be accessed remotely, there is a strong reason for a fine-grained authorization solution.

### 4.2.3. Autonomous Communication

In Web scenarios, HTTP servers in most cases act autonomously: they wait for and act upon incoming requests without human intervention. To perform authentication and authorization, they must know the authorization rules and be able to determine if these apply to the entity they are communicating with. The web servers must either be pre-provisioned with authorization rules or they must obtain the authorization rules remotely from their overseeing principals. With these rules, the servers must then determine whether a client is allowed to access a certain resource as requested. Authorization solutions for the IoT must be able to support autonomous servers (**Autonomous Server Req**).

HTTP clients are typically controlled directly by their overseeing principals. E.g., users direct their browser to a Web page they intend to access by typing its name into an address field or clicking on a hyperlink. In IoT applications, the client often needs to act autonomously. It may use service discovery mechanisms to find servers to communicate with, and interact with them without human interaction. In these cases clients must be enabled to act in their overseeing principals' interest. Their principals must be able to define the resources and servers that clients are allowed to access. Such information may be pre-provisioned, e.g., hard-coded in the client software, but that makes the client less flexible and difficult to maintain. A more flexible solution enables the client to obtain the authorization information remotely. Clients must then determine if an authorization rule applies to a certain server. Authorization solutions for the IoT must be able to support autonomous clients (**Autonomous Client Req**).

### 4.2.4. Denial of Service

Denial of Service (DoS) attacks aim at preventing the authorized access to a service or system resource or delaying system operation and functions (cf. RFC 4949,

p. 101). An attacker might perform a DoS attack by depleting the hardware resources of the attacked device. Because of their limited resources, constrained devices are more prone to this type of attacks. DoS attacks are difficult to prevent completely. Nevertheless, authorization solutions must be designed to mitigate the risk of DoS attacks on IoT devices that use the solution, i.e., the authorization solution must be protected against DoS attacks (**DoS Req**).

Since servers have the passive role in the communication and cannot control who tries to contact them, they are more vulnerable to these attacks. Protocols must therefore protect servers in particular. Servers must use as little system resources as possible, especially when they communicate with unauthorized clients. Storing data about clients between requests requires system resources. The REST stateless-server constraint (see section 3.4) therefore mitigates the risk of DoS attacks on servers (although it is still possible to, e.g., send large amounts of data in a single request to deplete the server's resources). To communicate securely, endpoints require certain information about their peer, which makes it difficult for security solutions to comply with the stateless-server constraint.

Clients are the initiators of communications and thus have more control over the system resources they use for the communication. They are therefore more difficult to attack. Nevertheless, protocol designers must be careful with spending the client's (maybe very) limited system resources. Solution designers must consider that in IoT applications clients may act unsupervised (see section 4.2.3); they are likely not able to reflect on their behavior and adjust it if an unexpected event occurs.

IoT devices cannot only be the target but also an instrument for DoS attacks. If attackers gain control over IoT devices, they can abuse them to build so-called botnets for DDoS attacks. These attacks aim at incapacitating servers by generating vast amounts of traffic. In the end of 2016, several large-scale DDoS attacks were performed by IoT botnets, the most prominent on the company Dyn that controls large parts of the Internet's Domain Name System (DNS) infrastructure (Hilton 2016). The attack came primarily from the Mirai botnet that consists mainly of digital cameras, DVR players, and routers (Antonakakis et al. 2017). One method to conduct a DDoS attack is the amplification attack where a certain request triggers a comparatively large response (see, e.g., Bing 2019). Authorization solutions

can mitigate these threats because they allow servers to provide long responses only to authorized requests.

#### **4.2.5. Offline / Cloud**

Many IoT applications require access to servers in the cloud. This approach has the advantage that application logic and application data can be stored in the cloud which may save storage space and system resources on the constrained device. Also, storing data in the cloud may mitigate firewall traversal problems. Unfortunately, with this approach the IoT device can no longer operate if the connection to the cloud fails.

In some applications, e.g., during the installation of devices in the building automation use case, the devices are not always able to reach servers in the Internet. Also, the connection may be disrupted due to connectivity problems. And the cloud service itself may be disturbed, which can even concern services of providers with significant resources; e.g., in February 2017, the Amazon S3 Service was affected by a malfunction that was caused by an error during maintenance (Amazon 2017).

#### **4.2.6. Perimeter Security**

One approach to secure the data on an endpoint is to direct all traffic for this endpoint through an intermediary that determines if this communication is allowed or not. For this approach, the intermediary must analyze all data for the endpoint (Lampson 2007, p. 4). The protection is broken if an attacker manages to communicate directly with the endpoint that hosts the resource. E.g., if a device is secured by allowing them only to communicate with devices within radio range, attackers can bypass the protection if they manage to get close to the device.

A smart object that does not enforce the authorization rules itself cannot participate in protecting the overseeing principals' security objectives. For devices that handle sensitive data this approach is not recommendable. It is therefore important to develop authorization solutions that support the constrained devices themselves.

### 4.2.7. Multiple Stakeholders

The use cases show that in IoT applications often multiple stakeholders are involved who have different (sometimes even conflicting) interests and security objectives. If a company does not suffer directly for a security breach, it has less incentive to provide sufficient protection (see also Anderson 2001, p. 1). Best results can be expected if each overseeing principal can directly protect the security of her own data and devices. OVPs may not be willing to let other OVPs decide about their assets.

The **main authorization directive** can only be complied with if overseeing principals have the possibility to make decisions for their data and devices and these decisions cannot be tampered with against the overseeing principals' wishes (**Separate Authority Req**).

### 4.2.8. Mutual Authorization

If data needs to be securely transmitted between two endpoints, both endpoints must participate in the protection. To prevent confidentiality breaches, the sender must always validate that the receiver is allowed to receive the data. Integrity breaches can only be prevented if the receiver ensures that the sender is allowed to provide the data. Otherwise the receiver might accept false data from an attacker which may be processed by the receiver and be provided to other communication partners.

All communicating endpoints must be aware of the protection requirements and act accordingly. Relying on the peer to perform the authorization (e.g., only the server validates that the client is authorized but not vice versa) is not sufficient. The authorization must be *mutual*, which means that each endpoint individually validates the authorization of its communication partners. Authorization solutions that provide mutual authorization supply each endpoints with the necessary information to validate its peer's authorization.

Where endpoints are not directly controlled by their overseeing principals, they must validate their peer's authorization by themselves. In IoT scenarios, both

server and client may be acting autonomously (see section 4.2.3). Authorization solutions should therefore support mutual authorization (**Mutual Authorization Req**).

### 4.3. Requirement Summary

This section summarizes the requirements that derive from the use cases. For a better overview, we divide them into categories. Environment requirements derive from the characteristics of the environments where smart objects are deployed (section 4.3.1). Requirements concerning the content and configuration of authorization rules are summarized in section 4.3.2. Requirements for updates of authorization rules are addressed in section 4.3.3. For an effective security solution, the whole lifecycle of smart objects must be protected (see section 4.3.4).

#### 4.3.1. Environment Requirements

The requirements that result from the characteristics of the environments where smart objects are deployed are listed below.

**Absent Principal Req:** the overseeing principals may not be able to intervene at the time of the communication. Nevertheless, their security objectives must be achieved.

**Offline Req:** the devices may not always be able to reach endpoints in the Internet, but still need to securely communicate locally.

**Multiple Hops Req:** the communication may need to be securely transported over multiple hops.

**C2L Req:** the authorization solution must enable constrained devices to securely communicate with less-constrained devices.

**C2C Req:** the authorization solution must enable constrained devices to securely communicate with other constrained devices.

**Group Communication Req:** the devices must be enabled to communicate securely if a group communication solution is used.

**Capture Req:** if a device is captured, the risk of attacks on other parts of the infrastructure with the help of the data on the device should be limited.

**DoS Req:** the authorization solution must aim at mitigating the risk of DoS attacks on constrained devices and save system resources where possible.

### 4.3.2. Authorization Rule Requirements

This section summarizes the requirements for the configuration and content of authorization rules.

**Distinct Interests Req:** the security policies of all participating overseeing principals must be determined and enforced by the authorization solution.

**Separate Authority Req:** the authorization solution must enable overseeing principals to keep the exclusive control over their devices, i.e., overseeing principals must be enabled to specify authorization rules for their own devices.

**Fine-Grained Authorization Req:** the authorization solution must enable fine-grained authorization.

**Simple Configuration Req:** The configuration of authorization rules must be simple and require little effort.

**Preconfiguration Req:** It must be possible to preconfigure authorization rules.

**Context-Based Authorization Req:** The overseeing principals must be able to define time- or location-based permissions for cases where, e.g., a permission is only valid in defined time-periods during the day.

**Permission Delegation Req:** in some cases the holder of a permission must be able to delegate this permission to others.

**Autonomous Server Req:** servers must be able to determine if their overseeing principal decided that a certain client is authorized to send and receive certain information.

**Autonomous Client Req:** clients must be able to determine if their overseeing principal decided that a certain server is authorized to send and receive certain information.

**Mutual Authorization Req:** the authorization solution should support mutual authorization.

### 4.3.3. Authorization Rule Update Requirements

Requirements concerning the updates of authorization rules are listed below.

**Dynamic Updates Req:** overseeing principals must be able to dynamically update authorization rules when needed.

**Temporary Permissions Req:** overseeing principals must be able to grant temporary authorizations so that endpoints are enabled to get or receive certain data only for a limited time period. While context-based authorization rules are typically defined for repeating events and thus persist in the system, temporary permissions are often granted for a single occurrence and can be removed afterwards.

**Revocation Req:** if devices are no longer to be trusted, the authorization rules of affected systems and endpoints, e.g., of potential communication partners, must be changed as soon as possible.

### 4.3.4. Lifecycle Requirements

From the lifecycle of smart objects derive a number of requirements that are listed below.

**Commissioning Req:** devices must be securely added to the overseeing principal's administrative domain, thereby becoming devices that are overseen exclusively by this overseeing principal.

**Decommissioning Req:** devices that are no longer part of the network must be securely decommissioned.

**Handover Req:** decommissioning and recommissioning of devices must be performed securely.

## 4.4. Conclusion

In this chapter, we analyzed the characteristics of smart objects and of the environments and applications where they are used to derive the requirements

for an authorization solution for the Internet of Things. Smart objects come in various sizes and shapes and often lack the hardware capabilities that general-purpose devices such as PCs and smartphones have. Although it is possible to use stronger hardware, these devices would be more expensive and consume more energy. The widespread deployment of billions of things is made possible by low-cost devices.

The use cases show that the application scenarios for smart objects differ from Web scenarios. Making the code and protocols for smart objects smaller and simpler is not sufficient to solve every problem. Smart objects will often need to communicate autonomously and need to protect data even when their overseeing principals are not able to intervene during the process. IoT applications often involve multiple organizations with different trust boundaries and distinct interests. Therefore, authorization solutions must be prepared to safeguard the interests of multiple overseeing principals with distinct security objectives.

We will use the requirements that we identified in this part of our work to evaluate our architecture model in chapter 9 and our authorization solution in chapter 13.



## **Part II.**

# **Authenticated Authorization Model**

*First Sight is when you can see what's really there, not what your heid [head] tells you ought to be there.*

– Terry Pratchett (The Wee Free Men)

## 5. Authenticated Authorization Model

An established approach to determine the security of a solution is to conduct a mathematical proof. Formalized models are often considered to be the most advanced (Stacewicz and Wlodarczyk 2010, p. 158). While provable security is useful to analyze the security of a solution, it is not fail-safe. Koblitz and Menezes describe four aspects that result in successful proofs of systems that nevertheless are not secure (Koblitz and Menezes 2019, p. 519):

1. implicit assumptions in the protocol description,
2. wrong or incomplete assumptions about the attacker,
3. idealized assumptions about the systems,
4. the proof may have gaps or may not be applicable in practice.

Although proofs are an important tool for increasing the security of solutions, they may nevertheless lead to a false sense of security. To apply a formalized model to a solution, the security properties that must be achieved by the solution first need to be defined. Determining the required security properties is a difficult problem. Without assistance, properties might be wrongly assigned or important properties might be overlooked. To improve this situation, our Authenticated Authorization Evaluation Model (ANAZEM) aims at specifying the fundamental requirements of an effective authenticated authorization process. ANAZEM is a theoretical model that uses verbal descriptions. It is designed to be general but to not unnecessarily restrict how the solution is built. Thus, it can be used for the evaluation of every secure authorization process. Designing a formal model that is as general and still covers the whole authenticated authorization process would be very difficult. The intention of our model is not to provide a tool for automatically calculating the security of a solution, but to deepen the understanding of

---

the authenticated authorization process: the model reflects the fundamental characteristics, shows the difficulties in protecting the process, and explains why it is not possible to provide perfect security. Understanding the characteristics and problems for the protection of the authenticated authorization process enables solution designers to specify appropriate solutions. The model can serve as a check list to find necessary security properties that can then be used for formal verification. It allows solution designers, developers and security researchers to detect implicit assumptions and requirements and thus provide them with a guideline for developing, testing and proving secure solutions. The vocabulary presented in the model is valuable for identifying and discussing security characteristics.

To design security solutions that require minimal system resources, it is useful to rethink which minimal steps devices must perform to assure a secure and therefore effective authorization process. Thus, constrained devices may limit themselves to performing only tasks that are essential for the protection of their overseeing principals' data. Offloading security tasks to more powerful devices may reduce the burden on the devices and thus is worth investigating. We focus on enabling devices to participate in the protection of their overseeing principals' data instead of relying on perimeter security (see section 4.2.6); the endpoint must be informed about its OVPs' decisions and enforce them where necessary.

There is much confusion how the terms authentication and authorization relate. Li et al. use the term authorization to refer to the process of authentication and access control (Li et al. 2003, p. 133) while Lampson states that access control comprises authentication and authorization and states that the boundary between them is not clear (Lampson 2007, pp. 4–5). The reason for these difficulties is that authentication and authorization in most cases cannot function without each other: authorization requires authentication to distinguish authorized from unauthorized entities. To authenticate an entity, an endpoint requires certain knowledge about this entity, which must securely be obtained prior to or during the communication. Most authentication mechanisms in communication scenarios use authorized (“trusted”) third parties who vouch for certain attributes of an entity, e.g., the name of its owner, and thus assist in the authentication. Nested into the authorization process are therefore authentication and authorization subprocesses. We will share the notion of Ferraiolo et al. who state

that authentication and authorization are closely related and thus often confused, but still denote distinct concepts (Ferraiolo et al. 2003, p. 3). To avoid confusion, we use the term *authenticated authorization* to refer to the required synthesis of mechanisms for authentication and authorization.

The attacker model described in section 2.3 shows the necessity for endpoints that are communicating over the Internet to protect their communication. The cryptographic protocols that are used for authorization must obey the explicitness principle (Anderson and Needham 1995, p. 438) to be robust and thus effective: any necessary naming, typing and freshness information must be made explicit, and protocol designers must define their starting assumptions and goals; implementers must know under which conditions the security solution meets which security objectives. Security mechanisms can only be effective if all relevant information can be validated. Messages must be self-descriptive: “the interpretation of the message should only depend on its content” (Abadi and Needham 1994, p. 123). ***Continuous security can only be achieved if every step of the whole authenticated authorization process is secured, every piece of information that is used within the process is obtained securely from authorized entities, and the authorization of each entity is traced back to the overseeing principal. Also, for claims about an entity the chains that link the claim to that entity must be continuously unbroken.***

This chapter refines the authenticated authorization model we introduced in earlier work (Gerdes et al. 2015c,d). It provides a representation of the authorization process that reflects the requirements and problems for performing effective authorization in communication scenarios. **We will identify the fundamental authorization requirements that authorization solutions must meet to provide perfect security.** Unfortunately, actual applications have characteristics that preclude perfect security. **We will show that actual applications cannot be perfectly secure since they are unable to completely satisfy the fundamentals.** Nevertheless, the model is helpful to evaluate the security of a solution. The degree to which a solution satisfies the fundamentals defines its quality. The identified fundamentals and tasks facilitate finding omissions and vulnerabilities in authorization protocols. We showed how the model is used by applying it to the DTLS profile (Gerdes et al. 2018a, see also section 11.2).

---

A protocol that does not even sufficiently satisfy the model's fundamentals is not secure. Security protocols not necessarily need to perform every task, but may instead rely on other security solutions. In these cases, the model enables the protocol designers to explicitly specify which additional measures are necessary to achieve a secure solution. ANAZEM thus assists with identifying requirements on underlying security protocols and specifying the tasks that must be conducted prior to the protocol execution.

A promising approach to reduce the burden on constrained devices is to delegate difficult security tasks to more powerful devices. We define the respective delegation fundamentals. We will show that a device must be able to perform certain authorization and delegation tasks itself to participate in the protection of its user's data.

The decision making process itself is not part of ANAZEM. Because of the **main authorization directive**, the overseeing principals' decisions must be enforced, regardless if they are right or wrong. Our model does not address the correctness of the overseeing principals' decisions.

Authorization cannot protect against communication partners that are authorized but not trustworthy. In conformance with the Internet Threat Model (see section 2.3), authorization protocols need to assume endpoints to be secure as long as their authorization is valid. ANAZEM aims at protecting the authorization decisions of the overseeing principals. The actual trustworthiness of an entity is therefore irrelevant for our model. Changes in the overseeing principals' authorization decisions are an important part of the authorization process and therefore included in ANAZEM.

Endpoints rely on their own software and configuration files to represent their overseeing principals' interests. With incorrect software or configurations, endpoints may not be able to correctly enforce their overseeing principals' authorization rules. But ensuring the correctness of the software is not in focus of this work.

ANAZEM was designed for authenticated authorization and describes the fundamentals and tasks that are required to achieve it. Certain security objectives

must be protected for an effective authenticated authorization process such as the integrity, authenticity and confidentiality of certain data (see also section 2.1). These security objectives are covered by the model. In this chapter we will introduce an additional security objective, the data destination verifiability, that is also considered by ANAZEM (see section 5.2.1). The model does not address other security objectives such as third-party verifiability or privacy, since these do not directly influence the authenticated authorization process. If they are required in an application scenario, additional measures may be needed for their protection.

An overview of related work in this area is given in section 5.1. Section 5.2 introduces the authorization fundamentals that must be considered for effective authorization. These fundamentals are then used to derive the necessary tasks for performing authorization in a communication scenario. Section 5.3 shows how authentication must be used in the authorization process. Since devices may not be able to perform every task by themselves, they must be able to securely delegate tasks to others. The delegation fundamentals and resulting delegation tasks are introduced in section 5.4. Section 5.5 outlines how application data must be secured. All tasks must be performed for a secure authorization process, as we will show in section 5.6. Section 5.7 analyzes dependencies between tasks and derives the minimum set of tasks that an endpoint must be able to perform. Not all fundamentals can be completely satisfied by an actual solution. In section 5.8, we analyze to which degree a solution must satisfy the fundamentals to be sufficiently secure. An important design criterion for security models is their completeness, which we will discuss in section 5.9. In section 5.10, we will apply ANAZEM to a wide range of protocols with known vulnerabilities, and show that each of these vulnerabilities occurred because certain tasks were omitted and certain fundamentals were therefore not satisfied. Section 5.11 summarizes our findings.

## 5.1. Related Work

The effectiveness of formal verification methods depends on the accuracy of the underlying security concepts and requirements. Pai et al. distinguish two broad

categories for verifying security protocols: model checking and logical inference (Pai et al. 2011, p. 655).

Model checkers can detect design flaws in security protocols; e.g., Lowe used the Failure Divergences Refinement Checker (FDR) for generating the model to detect a vulnerability in the Needham-Schroeder Protocol (Lowe 1996). For model checking approaches, an abstract representation of the system, the model, is generated that can then be automatically validated against a specification (Clarke et al. 1999, p. 4). The representation must correctly reflect the protocol to be useful; designers must be careful not to model what the protocol is supposed to do, but what it actually does. Encoding protocols in a formal language is a difficult and time-consuming task. To determine if a protocol has a certain security property, e.g., the confidentiality of a certain piece of data, model checkers determine if the protocol's reachable states have this property (Basin et al. 2018, p. 741). A protocol may have an infinite number of reachable states, which makes checking every possible state infeasible. Methods such as abstraction can reduce the complexity but may lead to false positives in the verification (Stern and Dill 1995).

Model checkers require that the desired security properties of the protocol are known, but do not specify how this knowledge is obtained. Therefore, properties that are essential for the security of the protocol may be omitted and are then not validated by the model checker. ANAZEM aims at specifying the characteristics of authorization and task delegation protocols. It thereby helps to identify the protocols' required security properties that can then be used as input for protocol verification. Thus, gaps and vulnerabilities in a protocol can be detected. Unlike model checkers, ANAZEM also enables us to determine the minimal number of tasks that an endpoint must be able to perform to participate in the protection of its overseeing principals' data.

Logical inference or logics of knowledge and beliefs use statements that represent knowledge in the system, and rules to derive beliefs from this knowledge to validate security protocols (Meadows 1995, p. 136). One example for such a logic is the BAN logic (Burrows et al. 1990). It aims at describing the beliefs of the parties that are participating in an authentication process, and determining if an endpoint can rightfully believe them. It contains statements that reflect assumptions about the communication, and defines rules that describe how beliefs can

be derived from these statements. The BAN logic is designed for authentication and not for authorization, but it acknowledges the need for a trusted authority for authentication. How an entity becomes an authority for an endpoint is not specified. The initial establishment of security associations between endpoints, and between endpoints and their overseeing principals, is not addressed. Also, the BAN logic does not consider the confidentiality of messages. Therefore, only part of the authentication process can be validated.

ANAZEM differs from logical inference methods since it does not define statements and derivation rules. Our focus is to ensure that the endpoint has all required knowledge to securely perform a certain task. One important goal of our model is that it allows us to determine which architecture and which protocol are best suited to support constrained environments. ANAZEM is therefore designed to be general, i.e., it does not unnecessarily restrict the design of authorization solutions. A similarly general inference logic is difficult to design. Inference logics are used to detect problems in protocols, but our fundamentals and tasks make it easier to detect the cause of the problem, and to determine how it can be fixed. Like model checking approaches, inference logics cannot be used to determine the minimum set of tasks that endpoints must be able to perform to participate in the protection of their overseeing principals' data.

Authorization has several important aspects that may need to be addressed by authorization solutions. Abadi et al. discuss the secure delegation of permissions; in particular, they analyze how an endpoint can believe that an entity acts in behalf of another entity (Abadi et al. 1993, p. 708). They define a modal logic to express relationships between requesting entities, e.g., that an entity A speaks for an entity B and has the credentials to prove it.

Li et al. define a logic-based delegation language (Li et al. 2003). They emphasize the importance of authenticated attributes such as the age or affiliation of an entity as the base of access control and of decentralized attribute authorities. Their language is able to express details such as limiting the delegation-depth when delegating permissions, and dynamic "k-out-of-n thresholds", where the approval of several users is required for performing a transaction (Li et al. 2003). They also define a trust root for authorization that is the "authorizer of an authorization decision" (Li et al. 2003, p. 138).

A lot of research concerns the formalization of authorization rules. Woo and Lam focus on the representation and evaluation of authorization rules in distributed systems (Woo and Lam 1992a, p. 33). They postulate that authorization mechanisms for distributed systems must consider attributes other than the identity of an entity such as the role or group the entity belongs to, or the entity's location. Also, they emphasize that distributed systems may consist of multiple independent domains with different administrative authorities that must be considered. They define a language for expressing authorization policies and present an approach for dealing with authorization rules from multiple overseeing principals.

Jajodia et al. introduce the problem of conflicting permissions that may arise in distributed systems with multiple overseeing principals. The authors propose a language that enables overseeing principals to define positive and negative authorizations and allows for conflict resolution policies (Jajodia et al. 2001).

It becomes clear that a lot of attention has been paid to the question how authorization decisions must be represented to make them interpretable, including how a language for expressing authorization rules would need to look like. Validation mechanisms mostly focus on authentication and cover only parts of the authorization process. An investigation that encompasses the complete authorization process and the tasks that are required to secure it is missing. We include the findings of the mentioned authors in our model. In addition, our approach considers the whole authorization process including the generation and provisioning of authorization rules, and shows that authentication and authorization must complement each other without gaps to achieve the security objectives. Instead of assuming only vaguely defined trust relationships between certain entities, our approach explicitly states which relationships between endpoints, peers, overseeing principals, third parties and authorization rules are required and how they must be validated.

## 5.2. Authorization

The **main authorization directive** (see section 2.2) states that the overseeing principals' decisions about their data and devices must be followed. Endpoints that

handle sensitive data must be informed about the overseeing principals' security policies and must participate in their enforcement (see also section 4.2.6).

A difficult problem in the authorization process is to securely obtain all information that is required for the authenticated authorization. Every piece of information that influences the process must be validated. A single inaccuracy can cause a vulnerability that renders the whole authorization process ineffective. We will identify the pieces of information that are required for authorization and will introduce the necessary rules for validating them.

In a basic communication scenario a communication endpoint communicates with a peer. The endpoint acts for one or more overseeing principals and needs to enforce their security objectives. The OVPs express their wishes with the help of authorization rules. Main components of an authorization process therefore are the peer, the endpoint's overseeing principals, and the endpoint itself.

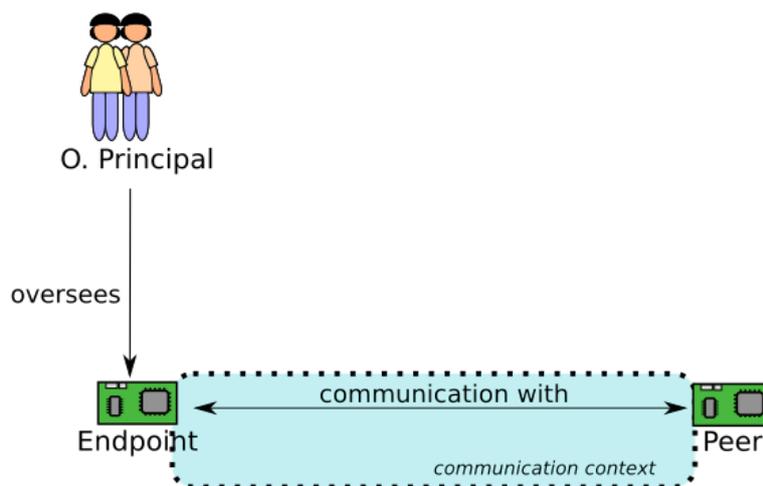


Figure 5.1.: Components of a Communication Scenario

### 5.2.1. Data Destination Verifiability

In section 2.1 we gave an overview of the main security objectives in the literature. With these goals, we can describe the security properties for validating the peer

and the OVP: data sent to the peer or OVP may require confidentiality, while data from the peer or OVP may need integrity and authenticity.

A security objective that is missing in the literature is the validation of the designated data destination: **an endpoint must be able to validate if the designated sender intends it to be the receiver of a piece of information.** We call this security objective *Data Destination Verifiability*. To achieve data destination verifiability, the integrity and authenticity of the data destination must be ensured: the destination of a piece of information must be bound to the information and be approved by the sender.

Data destination verifiability can easily be confused with confidentiality. But an endpoint that is authorized to read a certain piece of information is not necessarily the designated destination endpoint for this data. Imagine a message that is sent to a rocket launcher to initiate the starting procedure for a rocket. The message might be seen by other rocket launchers (no confidentiality required) but those must be able to determine whether the command was meant for them or not.

In a secure solution, the destination of a message is part of the data that is authenticated. But data destination verifiability must not be confused with authenticity. A message for which the origin can be validated and which is unmodified and fresh has integrity and authenticity, but may still not provide data destination verifiability. For example, a firmware update manifest may be signed by an authorized entity. If it is accepted by a device for which it is not intended, it may render this device inoperable.

The wide mouthed frog protocol (Burrows et al. 1989, pp. 25–26) is an example for a protocol where data destination verifiability is missing. The protocol is used to securely transfer a key that is generated by A to B using the trusted third party S. S shares  $K_{AS}$  with A and  $K_{BS}$  with B.

1.  $A \rightarrow S : A, \{ T_A, B, K_{AB} \} K_{AS}$
2.  $S \rightarrow B : \{ T_s, A, K_{AB} \} K_{BS}$

The second message can easily be manipulated to look like the first and sent to S. S would then generate a message with a new timestamp, and send it to A.

Attackers could thereby keep the session key alive until they can manage to steal it (Anderson and Needham 1995, p. 430). This problem could be prevented if data destination verifiability was considered.

That data destination verifiability is not recognized as a distinct security objective leads to a limited awareness and thereby to vulnerabilities. In a man-in-the-middle attack, also called middle-person-attack<sup>1</sup> (Anderson 2008, p. 74), attackers manage to impersonate the communication partner for the communicating parties; e.g., if A communicates with B, the attacker poses as B to A and as A to B. These attacks can be prevented if the data destination verifiability and the integrity of messages are enforced, while integrity-protection alone is not sufficient.

Data destination verifiability is very important for authorization because authorization rules often differ for distinct endpoints. E.g., the permissions for a door lock usually differ from those of a light switch. Also, different door locks may need to enforce distinct authorization rules. E.g., permissions for a bank's vault door differ from those of the entrance door.

### 5.2.2. Authorization Fundamentals

The authorization fundamentals derive from the **main authorization directive**. We define six distinct fundamentals to explicitly list the most important aspects of securing the authorization process. The fundamentals serve a common goal: a solution must comply with all fundamentals to be effective. Highlighting the various facets of secure authorization enables us to identify gaps. Also, distinguishing between the various fundamentals provides us with terminology to communicate and discuss potential gaps.

We will analyze the relationships between authorization rules, the communication endpoints and their overseeing principals. The pieces of information that establish and represent these relationships must be explicitly validated. The OVPs' authorization rules must always be enforced. This can only be achieved if the

---

<sup>1</sup>While *middle-person attack* is the more gender-inclusive term, the expression *man-in-the-middle attack* still is more common. In this work we use both terms interchangeably.

authorization information is correct (reflects the OVPs' decisions), and complete: the endpoint must have all current information that is required for the authenticated authorization.

### 5.2.2.1. Complete Mediation

Graham and Denning condense the aspect of enforcing the overseeing principals' wishes at all times into the concept of a reference monitor "through which all access to objects of that type must pass to be validated" (Graham and Denning 1972, p. 419). Saltzer and Schroeder's design principle *complete mediation* similarly demands that "every access must be checked for authority" (Saltzer and Schroeder 1975, p. 1282). Adapting this observation to communication scenarios, **we define the first authorization fundamental as: before every disclosure of a piece of data to an entity, and before accepting a piece of data from an entity, the authorization of the entity to obtain and provide this data must be validated (reference monitor fundamental)**. For cases where all entities are allowed to have unconditional access (see section 2.4) these validations are trivial and unrestricted authorization is used.

### 5.2.2.2. Correctness

To ensure that the authorization is enforced correctly, every information that is part of the OVPs' authorization decision must be made explicit. Otherwise, attackers might use implicit assumptions to their advantage. As described above, communicating endpoints are overseen by principals whose wishes are represented by authorization rules. A *permission* derives from instantiating an authorization rule. The relationships between the permission and the components in the scenario must be validated: Were the authorization rules approved by the endpoint's OVPs and thus reflect their decision (**principal fundamental**)? Do they really apply to the endpoint's peer (**correlation fundamental**)? Are they actually meant to be enforced by this endpoint (**destination fundamental**) in this context (**context fundamental**)? An overview of the relations between permissions and components is given in figure 5.2.

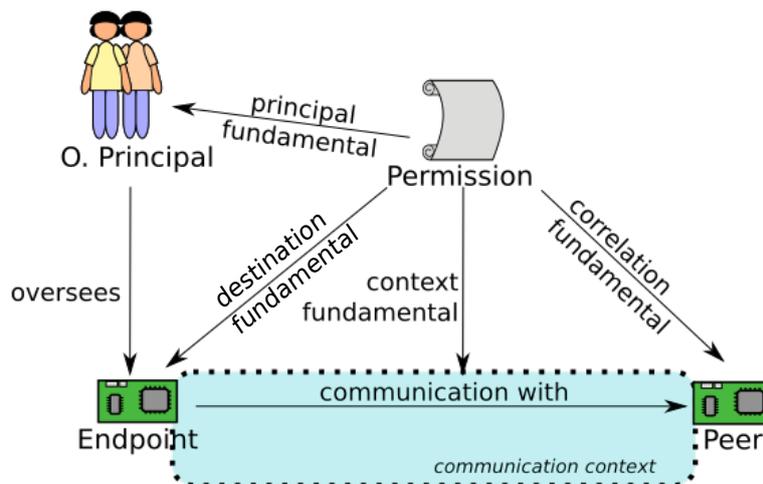


Figure 5.2.: Authorization Relations

Entities that are able to define authorization rules control who can access data and how. The overseeing principals' security objectives can only be achieved if the authorization rules are protected: authorization requires the integrity and authenticity of authorization rules to be effective. These goals must therefore always be a part of the OVPs' security objectives when authorization-related data is involved. The **main authorization directive** and the **reference monitor fundamental** apply: since OVPs are the main authorities for their data and devices they must approve of every authorization rule. **This results in the second authorization fundamental: each authorization rule must be approved by the overseeing principals (principal fundamental).** If a solution does not comply with this fundamental, it might not protect the OVPs' security objectives but act in someone else's interest. The correct interpretation of authorization rules is necessary to satisfy the **principal fundamental**.

To make sure that the correct authorization rules are enforced, it must be determined if an authorization rule really relates to the entity for which the authorization decision is to be made (cf. RFC 3281, p. 3). E.g., if an entity attempts to access a file, it must be validated that the authorization rule that grants access actually applies to this entity. Otherwise an attacker could impersonate the entity. Li et al. call this the proof-of-compliance problem: "Does a set of credentials prove that

a request complies with a policy” (Li et al. 2003, p. 128) **We define the third authorization fundamental as: the authorization rules must continuously relate to the entity for which the authorization rule enforcement is made (correlation fundamental).**

Overseeing principals usually grant authorization to an entity because it has certain attributes in the physical world, e.g., it belongs to a certain human being or company. The link between these attributes and the entity must be continuous. This fundamental includes that endpoints can determine whether a received message was actually intended by an authorized entity to be processed now and is not simply a copy of a previously sent message.

For authorization decisions that only depend on entity-independent constraints such as the time of access, satisfying the **correlation fundamental** is trivial since the same authorization applies to all entities. However, in this case we have to validate the time (see **context fundamental** below).

Also important for the correctness of the authorization is the data destination verifiability: before enforcing an authorization rule, the enforcing entity must be sure that it is meant to enforce the rule. Otherwise, an attacker could mislead an endpoint with authorization rules that were created for a different endpoint. This is especially problematic if various endpoints handle objects with the same name. Without data destination verifiability, entities can only assume that they are the destination of an authorization rule. **The fourth authorization fundamental is therefore defined as: the authorization rule must actually be meant to be enforced by the endpoint that enforces it (destination fundamental).** The wide mouthed frog protocol described in section 5.2.1 does not comply with this fundamental and is therefore vulnerable to attacks.

Additionally, the context of the authorization must be considered: the enforcing entity must determine if an authorization rule applies in the current situation. For classic access control, authorization rules protect objects such as pieces of information. If an authorization rule was designed to do so, the endpoint must be able to identify which object the rule relates to. Otherwise attackers might trick the endpoint to grant them access to a different object on the same location. Additionally, the context comprises constraints where authorization is only granted

when certain conditions are true. E.g., during nightshifts, physicians in a hospital are often responsible not only for their own patients but for several wards, and thus may be allowed to access patient data of other wards during the night. Finally, the enforcing entity might need to determine if a message is part of a certain conversation, e.g., if it is the answer to a certain request. If the context of the authorization rule is not checked, an attacker might trick the endpoint to enforce an authorization that was meant for a different context. **The fifth authorization fundamental is therefore defined as: authorization rules are only applied if their designated context fits to the current context (context fundamental).**

### 5.2.2.3. Completeness

Vollbrecht et al. state that authorization can be considered as the “result of evaluating policies of each organization that has an interest in the authorization decision” (RFC 2904, p. 16). To ensure the completeness, all currently valid decisions of all overseeing principals must be considered. This encompasses that authorization rules are up to date: if a rule is no longer valid and was replaced with a different rule, this change must be considered. Also, multiple rules may need to be considered for the authorization decision: a single OVP might define several rules, or, if multiple OVPs are involved, each of them might define own authorization rules. Some of the rules may even be conflicting, when the OVPs have different opinions about the authorization (cf. Jajodia et al. 2001, p. 216), and need to be evaluated before they can be applied. **Accordingly, we define the sixth authorization fundamental as: when the authorization of an entity is enforced, the authorization must reflect the decisions of all overseeing principals at this moment (rule completeness fundamental).**

### 5.2.3. Authorization-Related Tasks

The fundamentals described above must be considered for effective authorization. But how can they be satisfied by the communication partners? In this section we analyze the communication scenario (figure 5.1) to derive a list of essential tasks that must be performed to satisfy the authorization fundamentals. All

of these tasks must be performed for a secure (and thus effective) authorization. However, a solution might not solve all of them itself and not all of these tasks necessarily need to be solved technically. Solution designers, developers and security researchers can use the tasks as a checklist to identify gaps and vulnerabilities in a solution. Also, solution designers can explicitly state which tasks are addressed by their solution and which tasks additionally need to be performed, e.g., by an underlying security solution.

To distinguish the communicating endpoints we call one of them *endpoint* and the other the *peer*. To avoid confusion, we will concentrate on one side of the communication: the endpoint's side. It acts as the Policy Enforcement Point (PEP) (see RFC 2904, p. 30) for its overseeing principals. The tasks that are discussed in this section are required to achieve the principals' security objectives. We intentionally use indistinct terms to emphasize that either of the communication partners must perform the tasks for a secure communication: in an actual communication, an endpoint is always also the peer to the other endpoint. The same kind of tasks need to be performed on the peer's side as well to achieve the security objectives of the peer's OVPs. We use labels (A1 to A9) to distinguish between the required tasks. A list of these tasks can be found below.

Figure 5.3 gives an overview of a basic authorization scenario: The OVP as the main authority oversees the endpoint (**main authorization directive**) that communicates with a peer. Authorization rules manifest in permissions. All permissions of all OVPs must be applied to every part of the communication (**rule completeness fundamental, reference monitor fundamental**). The authorization rules must be approved by the overseeing principal (**principal fundamental**), relate to the peer (**correlation fundamental**), and must be meant for the endpoint (**destination fundamental**) in this context (**context fundamental**).

To comply with the authorization fundamentals, every step in the authenticated authorization process must be secured. Vollbrecht et al. distinguish three phases for authorization: retrieval, evaluation and enforcement of policies (RFC 2904, p. 16). To include tasks that are necessary for generating the policies, we add the generation phase. Also, we split the evaluation phase to distinguish between validation tasks and evaluation tasks. Altogether, we distinguish five phases of the

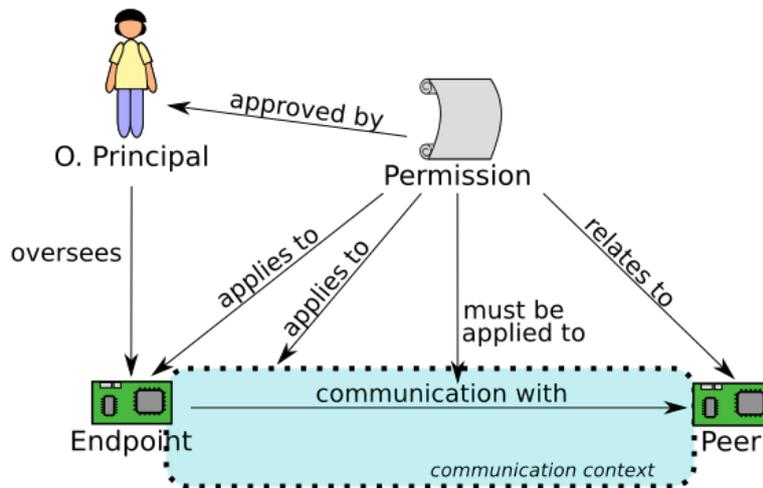


Figure 5.3.: Simplified Authorization Scenario

authorization process: the generation of security information, the provisioning of this information to the endpoint that requires it, the validation of the obtained information, the evaluation of all obtained information, and finally the enforcement, where the information is applied to the communication. Figure 5.4 shows the resulting phases.



Figure 5.4.: Phases in the Authorization Process

While the authorization process usually starts with the generation phase and ends with the enforcement phase, the phases cannot always be clearly distinguished and they are not necessarily sequential; e.g., a task from the validation phase might be performed together with a task from the enforcement phase. To perform a task, additional tasks may need to be performed. Phases may therefore also be nested. The following subsections describe the tasks that must be performed in each phase.

### 5.2.3.1. Generation

The authorization process starts with the generation phase: authorization rules must first be created (task **A1**, see also p. 95). They must reflect the overseeing principals' decision and be approved by them (**principal fundamental**). The approval must include the relation to the peer to which the rule applies (**correlation fundamental**), the endpoint that is meant to enforce the rule (**destination fundamental**), and the context in which the rule applies (**context fundamental**). The OVPs must somehow endorse their approval. Not in every case, authorization rules need to be stated explicitly. E.g., the authorization to be the overseeing principal (see also section 5.2.3.3) is often represented by a cryptographic key that is stored in a special location on the endpoint.

The intended peer, endpoint and context that the OVP configures do not necessarily need to be stated explicitly in the permission, but they must be bound to the permission: a secure relation between them and the permission must exist that can be validated. A message containing a permission is *self-descriptive* if it directly comprises all of the mentioned necessary information.

In the context of the permission we use the more general term *holder* for the intended holder of the permission (e.g., the peer), and the term *issuer* for the origin of the permission (e.g., the overseeing principal). If additional measures are required to satisfy the fundamentals, e.g., a validity period is added for satisfying the **rule completeness fundamental**, the corresponding data must also relate to the permission. The issuer must endorse the permission including all related information.

Figure 5.5 shows the relations that must be made explicit by the OVP. Each labeled arrow refers to the corresponding task. Generation tasks are identified by brown labels.

### 5.2.3.2. Provisioning

Endpoints must obtain the permissions (task **A2**) to be able to apply them to the communication. To comply with the **rule completeness fundamental**, all authorization rules relevant for this communication must be obtained. This includes

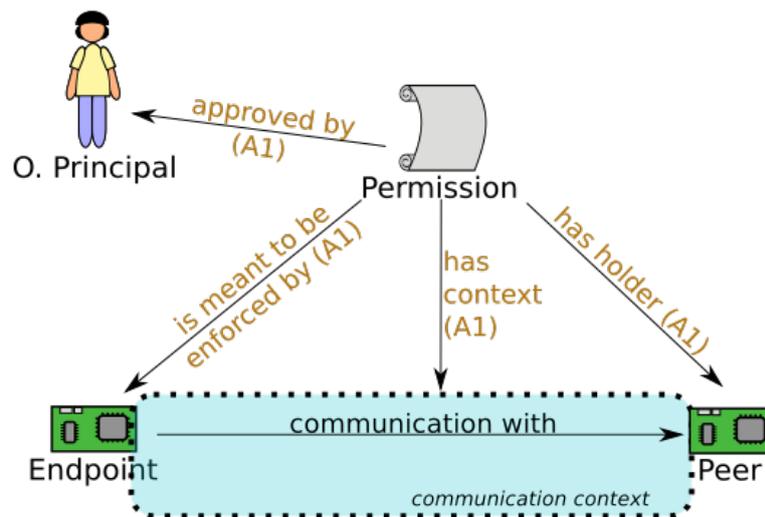


Figure 5.5.: Explicit Permission Relations

validating that the permissions are up to date, i.e., no newer permissions are available. Also, the permissions must reflect the overseeing principals' decision in this moment.

In practice, it is not possible to have perfect rule completeness. Endpoints have difficulties determining if they have received the most recent authorization rules or if an attacker managed to intercept them. Also, the transmission of authorization messages takes time, and in the meantime, authorization rules might change. Also, rules might have changed during the time when the authorization rules are processed, which may allow attackers to exploit that the time of check differs from the time of use (TOCTOU; see, e.g., Bishop and Dilger 1996). Even if the OVPs are present at the time of the authorization and can intervene directly, it is inconvenient for them to constantly affirm their authorization rules. After an initial confirmation, their agreement must therefore often be assumed for a certain time period, e.g., for the remainder of a session and as long as it is not revoked. The requirements of the application scenario dictate how a solution must satisfy the **rule completeness fundamental** and which mechanism or combination of

mechanisms is best suited for it. Also, the peril<sup>2</sup> of attacks must be considered: how much incentive has an attacker to compromise the authorization process, and how much time will it take her to, e.g., steal a session key? The impact and peril of an attack must be balanced against the problems that are caused by the mechanisms for satisfying the **rule completeness fundamental**. Although an authorization solution cannot completely satisfy all fundamentals, it needs to satisfy them to a certain degree. Otherwise, the solution is not secure. We will discuss how solutions can sufficiently satisfy the fundamentals in section 5.8.

After performing task **A2**, the endpoint has the current authorization rules and knows the related holder, endpoint, context and issuer to which they are meant to apply. Even if all entities have the same permissions because binary authorization is used, this task is still required: the endpoint must know that binary authorization is used.

The scenario with task **A2** is depicted in figure 5.6. The provisioning task is represented by a green label.

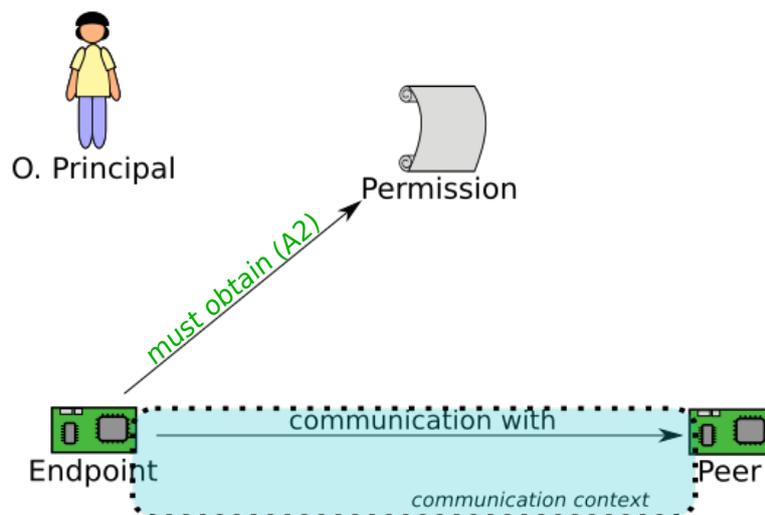


Figure 5.6.: Obtaining Permissions

<sup>2</sup>We would like to write probability here, but attacks do not occur at random; they depend on the attacker's incentive and means. Reducing the attacker's incentive mitigates the peril of attacks.

### 5.2.3.3. Validation

The endpoint must validate all received data that is relevant for the authorization and must ensure that it applies to the current communication. This phase is called the validation phase and comprises tasks that help to achieve the **principal fundamental**, the **correlation fundamental**, the **destination fundamental** and **context fundamental**. For some of these tasks authentication is required: Authentication and authorization must complement each other to achieve the security objectives (see also section 5.3).

To validate the permission in compliance with the authorization fundamentals, the endpoint must determine whether the permission is applicable in the current communication: the issuer, holder and context of the permission must equal the endpoint's current overseeing principal, peer and communication context, respectively. Also, the destination of the permission must be the endpoint itself. The tasks **A3** to **A5** are closely related to the provisioning and will in most cases be performed when the authorization rules are obtained. The tasks **A6** and **A7** must be answered for the current communication. Figure 5.7 shows the endpoint's view of the relations that must be validated.

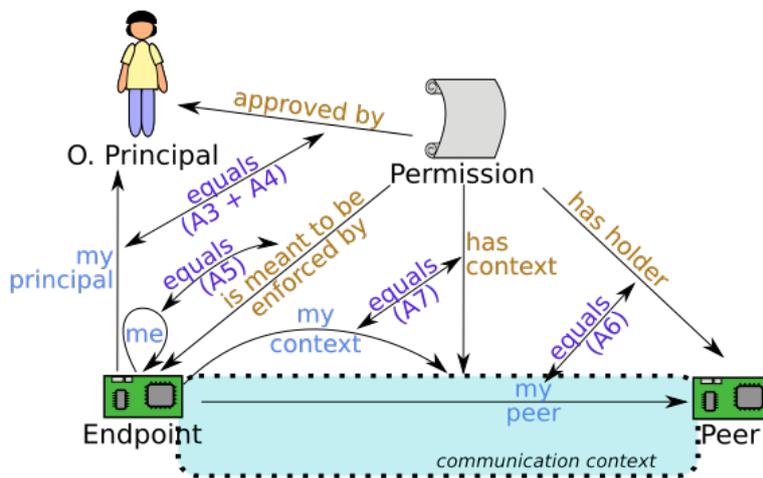


Figure 5.7.: The Endpoint's View on Validating Relations

The endpoint must validate that holder, rules, destination and context actually

relate to each other and stem from the same issuer (task **A3**). Otherwise an attacker might be able to change the data of one of these items and thereby bypass the authorization.

The endpoint must validate that it actually is responsible for enforcing the permission (**destination fundamental**) by determining that it is the destination of the permission (task **A5**). To achieve this, the endpoint and its overseeing principal must have a common way to identify the endpoint.

To validate the relation between the permission and the current peer (**correlation fundamental**), the endpoint must authenticate the peer. The authenticated entity must then be compared to the entity that is mentioned in the permission (task **A6**). For received messages, the endpoint must also validate that they actually came from the authorized peer and are not simply a *replay*, i.e., a copy of a previously sent message. We will discuss replay protection and freshness in section 10.3.

For entity-independent (e.g., time-based) authorization, validating the relation to the peer is trivial since the permission is valid for all peers. Nevertheless the endpoint must validate that the permission actually is entity-independent.

To satisfy the **context fundamental**, the context of the permission, including the object that is to be protected, must be validated (task **A7**).

The endpoint must ensure that every authorization rule for the peer (the peer authorization) can be traced back to the overseeing principal, (**reference monitor fundamental, principal fundamental**). The authorization of the permission's issuer to define permissions for this endpoint must be ensured (**A4**). At this point, the authorization process therefore encompasses another authorization process: the issuer authorization. To perform this task, all authorization tasks must be performed for the permission's issuer. The issuer not necessarily is the overseeing principal; principals may delegate some or all of their authorizations to others (see e.g., Li et al. 2003; Abadi et al. 1993). The authorization fundamentals apply: every authorization rule regarding the issuer (the issuer authorization) must be obtained, validated, evaluated and enforced. The issuer authorization must again be traced back to the overseeing principals for task **A4**.

Validating the issuer authorization therefore is a recursive problem. Since overseeing principals are the authorities for their data and devices, they are per definition authorized to issue authorization rules. However, the endpoint requires sufficient proof that the entity actually is the OVP. It must use authenticated authorization to determine that. If the issuer authorization is not performed correctly, i.e., if not all authorization tasks are performed for the OVP, attackers might be able to claim to be the OVP. ANAZEM therefore demands that the authorization of the OVP must be validated by the endpoint. The authorization recursion therefore never ends.

In an actual application, the recursion must end somewhere; authorization mechanisms often have a special mechanism to make the overseeing principal initially known to the endpoint. Such a mechanism must attempt to make sure that the endpoint can recognize its OVPs and validate their authorization. A common solution is to provision information about the OVP to the endpoint by a means that is assumed to only be usable by an overseeing principal. *Out-of-band* mechanisms use channels or methods that differ from the main communication channel or usual method (RFC 4949, p. 212). E.g., a USB connection may be used to directly store a representation of the OVP, e.g., a cryptographic key, at a special location on the device, thereby creating an initial security association. If attackers manage to use such an out-of-band channel, they can thereby gain control over the device. A security solution can only be as secure as the ownership transfer that makes the OVP known to the endpoint. We examine secure authorization management for constrained devices in chapter 14.

#### 5.2.3.4. Evaluation

According to the **rule completeness fundamental**, the validated permissions must be evaluated (task A8) to determine the actual authorizations of a peer; if there are multiple permissions, it must be clear, how they are handled, e.g., if they complement each other, or if a permission is only valid if all principals agree. This step might be more or less complicated depending on the number of overseeing principals and the number of permissions that these OVPs have defined.

The endpoint and its OVP must have a common understanding how authorization rules are evaluated. Even authorization solutions with only a single OVP need to define how multiple valid permissions must be handled, e.g., do they complement each other, or does the older permission become invalid if a newer permission arrives.

#### 5.2.3.5. Enforcement

The goal of the authorization process is to enforce the overseeing principals' decisions in the communication. After determining the OVP's decisions, endpoint must enforce them (task **A9**).

By applying the **reference monitor fundamental** to communication scenarios we can derive that the endpoint must validate the authorization of the peer before sending and after receiving data representations. To perform task **A9**, the endpoint must accordingly apply the permissions to the communication, i.e., every incoming and outgoing message.

How the enforcement is performed depends on the goal of the authorization. If unauthorized entities must be prevented from accessing messages, i.e., for all but unrestricted authorization (see chapter 2), the authorization enforcement requires the protection of messages. The required protection depends on the OVPs' security objectives. Communication channels in the Internet are assumed to be more or less under the control of the attacker (see section 2.3). The endpoint therefore must participate in the protection of messages to enforce the authorization, e.g., by encrypting messages to avoid eavesdropping, or by accepting only authenticated, integrity-protected messages. Task **A9** therefore is closely related to task **A6**.

Figure 5.8 gives an overview of all authorization tasks. Validation tasks are identified by blue labels, evaluation tasks are purple. The enforcement task's label is red.

To comply with the **rule completeness fundamental**, the endpoint must enforce the overseeing principals' decisions as they are at this moment. But the authorization rules must have been obtained, validated and evaluated before they can

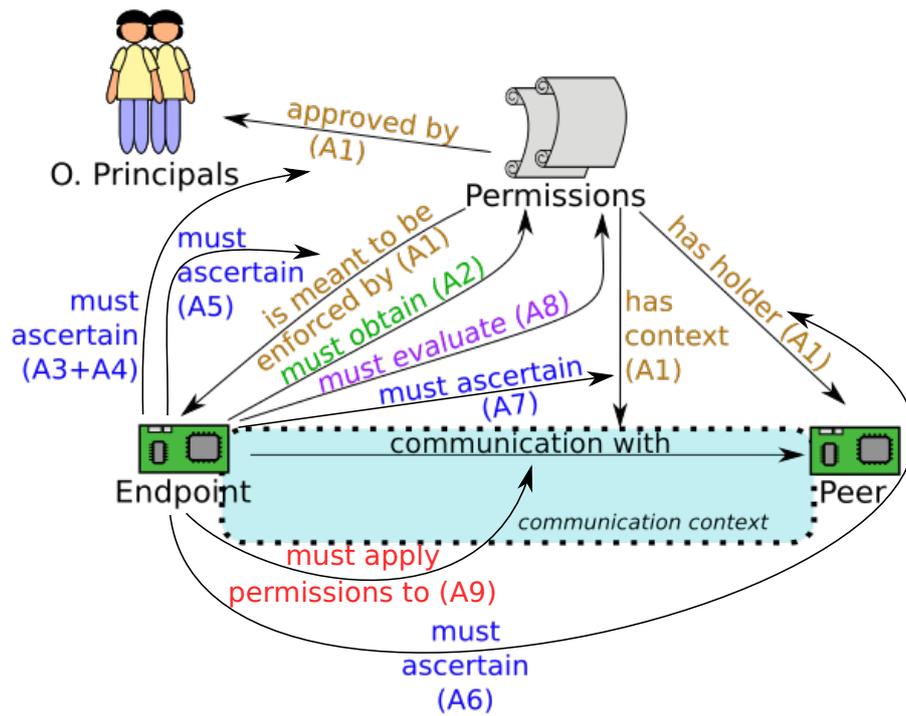


Figure 5.8.: Overview of the authorization tasks

be enforced. In an actual implementation, the OVP's decisions may change while the authorization rules are processed (see also section 5.2.3.2). Completely satisfying the **rule completeness fundamental** is therefore not possible for an actual implementation. The challenge for solution designers and developers is to reach a level of security that suffices for the application (see also section 5.8).

#### 5.2.4. List of Authorization Tasks

The Unified Modeling Language (UML) 2.0 activity diagram in figure 5.9 shows the necessary data flows of the authorization process. The tasks are not necessarily performed in the depicted order. All permissions of all overseeing principals must be configured. For each of these permissions the actions in the expansion node must be performed. Information about the peer must be derived from the communication (see also section 5.3). This might, e.g., be conducted by analyzing a cryptographic signature of the peer or by encrypting a message to the peer with the peer's cryptographic key. The endpoint must validate that the authenticated peer really is the holder of the permission (task **A6**). The information about holder, authorization rule, destination and context must relate to each other and be approved by the issuer (task **A3**). Also, the endpoint must make sure that it actually is the destination of the permission (task **A5**). To do so, it must have information about itself as indicated by the input data "me". Additionally, the current context must be known to validate that it corresponds to the permission context (task **A7**). The authorization of the permission's issuer to endorse the permission must be validated which encompasses another authorization process that is depicted in the action `validate issuer authorization` (task **A4**). The permission must be discarded if the result of one of the tasks A3 to A7 indicates that it is not valid. All valid permissions must be evaluated and the resulting authorization must be enforced. Task **A4** comprises all authorization tasks. The enforcement of the issuer authorization is performed by deciding whether the permission is valid and discarding it if it is not.

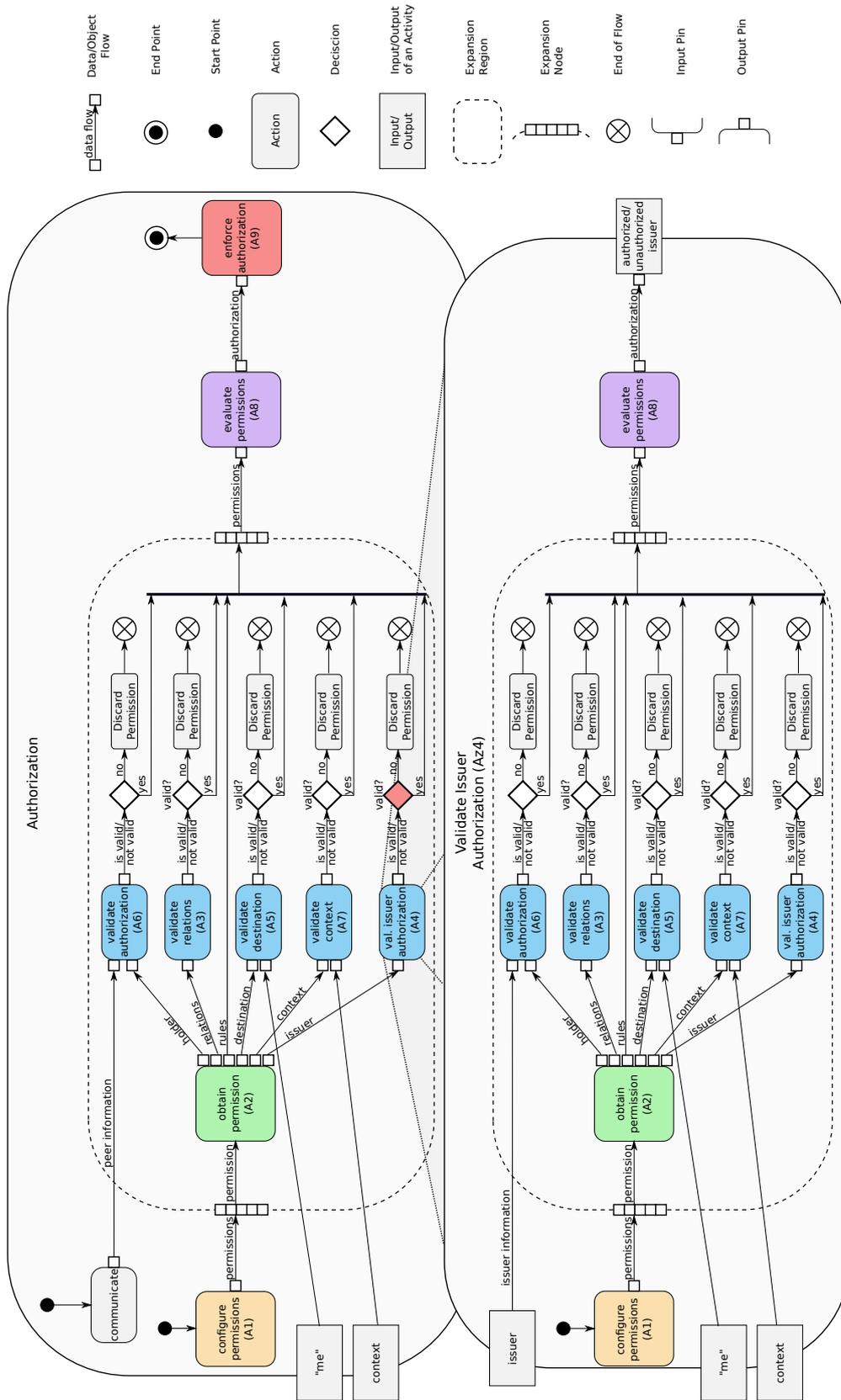


Figure 5.9.: Activity Diagram of the Tasks

Summarizing, the authorization-related tasks listed below must be performed. The tasks are phrased in passive form since the endpoints not necessarily have to perform them by themselves (see section 5.4).

#### Generation

**A1** Configure permissions: overseeing principals must confirm the the authorization rules that realize their authorization policy. This includes that a) the authorization rule is configured, that b) the endpoint that must enforce the authorization rule is specified, that c) the context for the authorization rule is set, and that d) the relation to the intended holder (the entity for that the authorization rule shall apply) is defined. Also, OVPs may need to define additional information to satisfy the authorization fundamentals such as the validity period of a permission or measures for replay protection. These pieces of information must be somehow bound together and endorsed by the overseeing principal.

#### Provisioning

**A2** Obtain/provide permissions: the authorization rules must be made available to the endpoint that enforces the authorization. All permissions that are actually currently valid must be obtained.

#### Validation

**A3** Validate relations: the authorization rule, holder, destination, context, and potential other authorization-related information must relate to each other and stem from the same issuer.

**A4** Validate origin authorization: the authorization of the entity that approved the authorization rule must be validated. This encompasses another authorization process.

**A5** Validate destination: the endpoint must confirm that it is meant to enforce the authorization rule.

**A6** Validate authorization: the authorization of the entity in question (the peer or issuer) must be confirmed by validating that it is the holder of the permission. There must be a continuous link between the entity and the attribute that is the reason for the overseeing principal to grant the authorization.

**A7** Validate context: the endpoint must confirm that the actual context matches the designated context of the permission.

#### Evaluation

**A8** Evaluate permissions: the endpoint must determine the actual authorization of the peer using the permissions.

#### Enforcement

**A9** Enforce authorization: The endpoint that enforces the authorization must apply the authorization rules to the communication, e.g., by securely transmitting a piece of information to the peer.

### 5.3. Authentication

Overseeing principals in most cases grant permissions to a peer because of certain (sets of) attributes it has in the physical world, such as its location, or the name, affiliation, or age of its overseeing principal<sup>3</sup>. Authentication is often associated with validating an identity (ISO/IEC 24760-1; Menezes et al. 1996). However, there are different views how the term *identity* is defined. While various authors agree that an identity consists of a set of *attributes* (Shirey 2007; ISO/IEC 24760-1; Williamson et al. 2009), there are different opinions concerning the uniqueness of the identity. We use the definition from ISO/IEC 24760-1, where an identity is defined as a “set of attributes related to a party”, and no uniqueness of an identity within a domain is assumed unless explicitly stated.

Historically, authorization decisions were based on the unique identity of an entity. While this approach may be necessary for some applications, many others do not require uniqueness. E.g., for a liquor store, the age of its clients is more important than their name to determine whether they are allowed to buy alcohol.

---

<sup>3</sup>As stated above, permissions may also be granted because of entity-independent conditions such as the time of access.

Li et al. offer another example where a book shop grants discount prices to all university students (Li et al. 2003, p. 130).

Throughout this thesis, we therefore use the following definition: **Authentication is the process of validating that an entity actually currently has certain attribute values** (cf. Gerdes et al. 2015c, p. 234). This means that an entity in the system that, e.g., expresses a claim, provides a service, or performed or attempts to perform an action, can be linked to certain attribute values. The authentication of messages usually involves validating that the message stems from an entity with certain attribute values and was not forged or modified (cf. Menezes et al. 1996, p. 25).

Unless entity-independent authorization is used, task **A6** requires attribute validation: before the peer can be authorized, it must be assured that it actually has the required attributes (task **An1**, see also p. 98) that link it to the permission (see figure 5.10).

The authenticated attributes (regardless of whether they are unique or not) must be meaningful for the authorization (**correlation fundamental**), i.e., the information must be useful to determine whether an entity is authorized or not. Also, the principal that oversees the authorization and the entity that validates the attribute values must have a common understanding of the attributes.

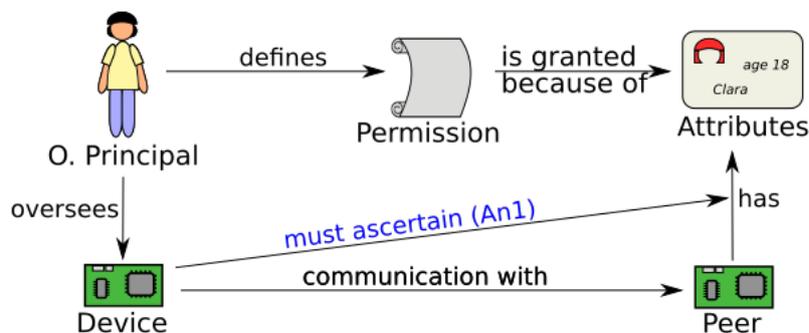


Figure 5.10.: Relations between Peer, Permission and Attributes

Validating the origin authorization (task **A4**) encompasses another authorization process which again may require attribute validation.

Task **A5** requires a different kind of attribute validation: the endpoint must validate that it has the attributes that mark it as the entity (or one of them) that is

supposed to enforce the authorization. This requires that the endpoint and its OVP have a common understanding of these attributes.

The authentication process and the authorization process must both be protected for an effective authorization. Since the authentication provides the information that is necessary to determine whether an entity is authorized or not, compromising the authentication process renders the authorization ineffective.

For conducting authentication, the following task must be performed:

**An1** Attribute validation: it must be determined if an entity actually currently has certain attributes. This may require organizational measures and checks in the physical world.

**An endpoint that performed authenticated authorization, i.e., the authorization tasks and, if necessary, task An1, for a peer, has built a security association with this peer: the endpoint knows what the peer is authorized to do and knows how the peer is identified.**

## 5.4. Task Delegation

The tasks described above are required for effective authenticated authorization. However, an entity does not have to perform every task itself, but can delegate tasks to other entities. For some tasks, delegation may even be required in actual implementations. Task **An1** will usually be delegated since there are very few attributes that an endpoint can validate at the time of the communication. Even assuring the proximity of devices is difficult to validate; the approach of using Near Field Communication (NFC) been shown to be susceptible to attacks (cf. e.g., Francis et al. 2010, p. 36). Ensuring other attributes such as being the owner of a certain endpoint must be prepared beforehand and requires organizational measures and human interaction.

Endpoints cannot delegate every task. E.g., task **A9** must be performed by the endpoint itself. An endpoint that cannot perform task **A9** is not able to participate

in the protection of the overseeing principals' security objectives. We will analyze which tasks an endpoint can delegate in section 5.7.

If an endpoint cannot perform a task such as the attribute validation on its own, it must rely on a third party, the *claim issuer*, to do it instead. The claim issuer performs a task for the endpoint, e.g., validating that the peer actually has the claimed attributes, and delivers the task's result to the endpoint in the form of a claim. Claims always have an origin, i.e. the claim issuer that states the claim. They also always have a *statement*, e.g., the claimed attributes. We call claims that assert attributes of an entity *attribute claims* and the entity that the attributes are accredited to the *holder* of the claim. The entity that delegates a task is denoted by the term *delegator*. The overseeing principal of an endpoint is a special type of claim issuer.

#### 5.4.1. Delegation Fundamentals

Delegators can rely on claim issuers for authorization tasks but must make sure that the authorization fundamentals are fulfilled. As stated above, a compromised authentication renders the authorization ineffective. A malicious attribute claim issuer can falsely pretend that an entity has certain attributes and may thereby accomplish that the entity is authorized illegitimately. Issuers of attribute claims thus need to be trusted with the authorization decision (cf. RFC 2693, p. 13). The same is true for every piece of information that influences the authorization process. To avoid compromising the authorization process, the delegator must validate the authorization of the claim issuer: **We therefore state that for an effective authorization, every entity that provides information that influences the authorization process is a claim issuer; every claim issuer must be authorized to perform authorization tasks for the delegator and to state claims that influence the authorization process (claim issuer authorization fundamental).**

The **main authorization directive** and the authorization fundamentals therefore apply. Overseeing principals might want to authorize claim issuers for some scenarios but not for others. E.g., while a claim issuer might be acceptable for claims

concerning news feeds or weather forecasts, OVPs might not accept claims of this claim issuer for banking applications.

The claim issuer must act in in the interest of the delegator: it must securely perform the task that is delegated to it, and must protect the data that is entrusted to it. This includes that the authorization of the delegator to provide input to the claim and to receive the claim is validated. **The second delegation fundamental is defined as: The claim issuer must participate in protecting the security objectives of the delegator's overseeing principal and satisfy the authorization fundamentals for the tasks that are delegated to it (claim issuer participation fundamental).**

Endpoints must only accept claims from authorized claim issuers. Claim issuers therefore must endorse the claims including the claim statement and related information concerning destination and holder. The delegator must ensure that the claim issuer actually stated the claim information. Otherwise an attacker could impersonate the claim issuer and provide forged claims. **We define the third delegation fundamental as: the claim issuer must endorse the claim including all related information (claim origin fundamental).**

Data destination verifiability is also important in delegation. The claim issuer might make different claims for distinct delegators. E.g., if endpoint A that is overseen by principal Alice and endpoint B with OVP Bob both delegate the attribute validation for their overseeing principal to claim issuer Sam, they must get different answers. **Thus, the fourth delegation fundamental is defined as: a claim is only processed by endpoints that the claim issuer intends to be the destination of the claim (claim destination fundamental).**

The delegator is required to have all currently applying current claims from all relevant claim issuers. The claims of various claim issuers might be required before e.g., an attribute claim is believed. Also, it must be assured that the claim is up to date; i.e., it must be validated that the claim statement reflects what the claim issuer would say at this moment. If the claim is no longer valid and is replaced with a different claim, the new claim must be considered. **We therefore define the fifth delegation fundamental as: at the time when a claim is used,**

**the used statement must reflect the opinion of all relevant claim issuers at this moment (claim completeness fundamental).**

Attribute claims can only be effective if the delegator ensures that the claim actually refers to the holder for that the delegator requested information. Otherwise the attributes might be imputed to the wrong entity. **The sixth delegation fundamental is therefore defined as: the claim must actually relate to the holder (holder relation fundamental).**

The delegation process may sound familiar; accepting permissions from an OVP (see section 5.2.3) is a task delegation process: since the endpoint does not know the authorization rules itself, it must rely on its overseeing principal to provide them.

### **5.4.2. Delegation Tasks**

Several tasks must be conducted to comply with the delegation fundamentals and ensure that the overseeing principals' interests are protected when tasks are delegated. Since the **reference monitor fundamental** applies, delegators must ensure the authorization of the claim issuer before delegating a task and after receiving a claim.

#### **5.4.2.1. Generation**

The required input data for the task must be known to the claim issuer or securely be transmitted to it, and the output data of the task must be securely transmitted back to the delegator as the claim statement. To comply with the **claim issuer participation fundamental**, the claim issuer must validate that the delegator is authorized to send the delegation data including the input data for the task, and that it is allowed to receive the claim statement (task **D0**, see also p. 109). Also, the claim issuer must perform the delegated task before generating the claim statement that reflects the result of the task; e.g., for an attribute claim, the claim issuer must perform task **An1**.

To comply with the **claim origin fundamental**, the claim issuer must then endorse the claim, e.g., by applying a signature to it (task **D1**). The endorsement must enable the delegator to validate that the holder, statement, destination, and potential other information, such as the lifetime of the claim, actually relate to the claim and were stated by the same claim issuer.

The **holder relation fundamental** requires that attribute claims provide a verifiable link to the holder. To achieve this, these claims are bound to a *verifier*. The claim issuer thereby states that the claim statement is true for the entity (or entities) that is identified with the help of the verifier. E.g., in the physical world, photos are used as the verifier to validate the relation between a person and her passport. In computer networks, relations are usually validated with the help of cryptographic keys.

The endpoint that is the receiver of a claim uses the verifier to validate that it actually is interacting with the holder of the claim. If asymmetric keys are used, the verifier typically is the public key. The issuer's endorsement—e.g., the signature—that binds the verifier to other claim information is not a verifier. An endpoint that has a claim about a peer and a claim about the issuer may use the verifier in the issuer claim on the endorsement of the peer claim to validate that the peer claim was made by the holder of the issuer claim.

The delegation process may sound familiar; accepting permissions from an OVP (see section 5.2.3) is a task delegation process: since the endpoint does not know the authorization rules itself, it must rely on its overseeing principal to provide them. Overseeing principals that provide permissions to their devices are claim issuers. The permissions are claims. Fundamentals that apply to claim issuers therefore also apply to overseeing principals. This relation is, of course, unidirectional: not every claimant is an overseeing principal.

Figure 5.11 shows how the claim issuer issues the claim after validating that the holder actually has certain attributes and a certain verifier. Figure 5.12 shows how an attribute claim is used for task **A6**.

In security solutions, verifiers are often used as a representation of the entity. The relationship to the attributes may no longer be visible. Nevertheless, verifiers are

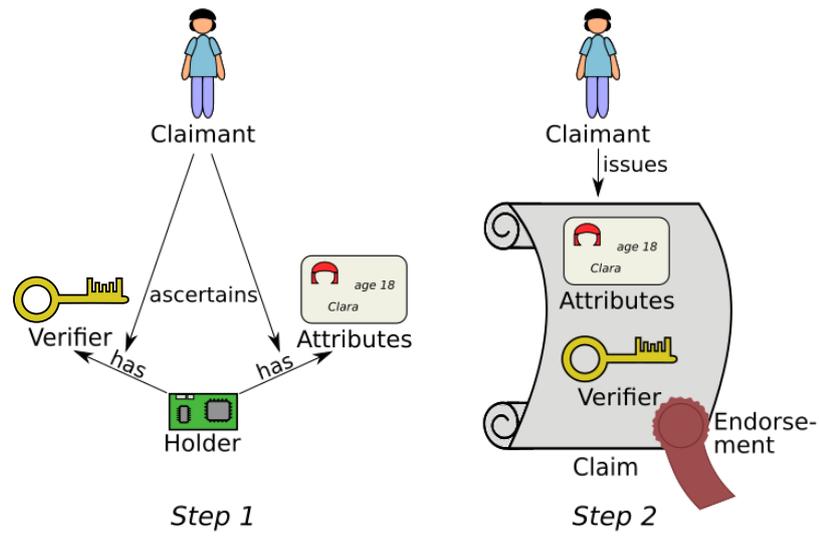


Figure 5.11.: Issuing an Attribute Claim

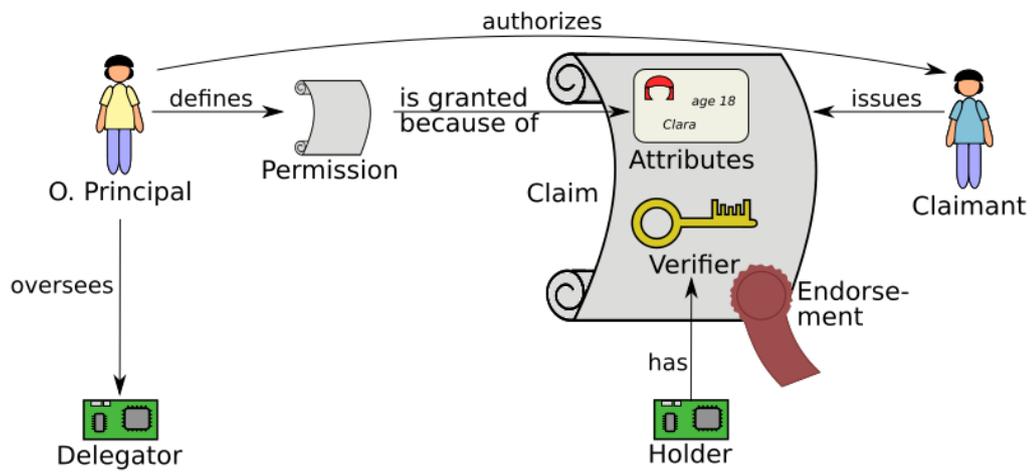


Figure 5.12.: An Attribute Claim Used for Authorization

only an abstraction of the actual attributes that are meaningful for the authorization. Verifiers are not themselves attributes: An overseeing principal may grant permissions to an employee because she works for the principal's company. If a key is assigned to her that represents her attributes, the key is not the reason for the authorization; if the key is stolen, the employee still works for the company, and the thief is not meant to be authorized.

**Verifiers must *exclusively identify* entities for which the claim statement is true. If a verifier entitles its holder to a certain authorization, all entities that have this verifier have this authorization.**

For solutions that use asymmetric cryptography the public key usually is the verifier: claim issuers bind the holder's public key to a claim statement. Endpoints validate that the claim statement is true for a certain entity by ascertaining that this entity can use the respective private key. To use a private key as a verifier of a claim, the corresponding public key must only be known to entities for which the claim statement is true, which would be very unusual.

The use of multiple verifiers might be required in an authorization process. Permissions may be directly bound to a cryptographic key. We call these constructs *authorization claims*. To generate an authorization claim, a claim issuer might rely on an attribute claim. Indirections require that the involved claim issuers and delegators know for which claim statement the verifiers stand.

#### 5.4.2.2. Provisioning

The claim including the verifier must be provisioned to the parties that require them (D2). The claim statement must reflect the current result of the task that the claim issuer performs for the delegator to ensure that the fundamentals are satisfied<sup>4</sup>. Thus, it must be assured that the claim is up to date. To comply with the **claim completeness fundamental**, the endpoint must obtain all currently valid, applicable claims.

---

<sup>4</sup>If, e.g., an attacker uses compromised keying material, the **correlation fundamental** is violated.

Like rule completeness, perfect claim completeness is not possible in practice (see also section 5.2.3). We will discuss how a solution sufficiently satisfies the **claim completeness fundamental** in section 5.8.

The actual implementation of the key distribution required for task **D2** depends on the solution. To exclusively identify an entity, the keying material that is used as the verifier may need to be kept secret: for symmetric solutions, a symmetric key that acts as a verifier must be known to the claim issuer, the holder and the endpoint. For asymmetric solutions, the holder's key pair is used. The public key must be provided to the claim issuer and the endpoint, while only the holder is allowed to know the corresponding private key. Verifiers might be pre-provisioned to an endpoint. E.g., verifiers that identify the overseeing principal might already be provisioned during the manufacturing phase (see also section 14.4.1).

#### 5.4.2.3. Validation

The delegator must validate that all necessary parts of the claim information are present, that the parts actually relate to the claim and that they stem from the same claim issuer (**D3**). The claim information includes the statement, destination, and, for an attribute claim, holder. Also, the claim information may comprise data that allows the delegator to determine the freshness of the claim, and enables it to notice if the claim is only a copy of a previously stated claim (which may be a problem if, e.g., only the newest claim is to be considered or if the claim was revoked in the meantime).

Additionally, the authorization of the claim issuer must be confirmed (**D4**): the claim issuer's permission to assert the claim must be traced back to the overseeing principal to comply with the **claim issuer authorization fundamental** and the **principal fundamental**. The authorization tasks that are included in task **D4** must be performed with the delegator as the endpoint and the claim issuer in the role of the holder.

Delegators must check if a claim was really meant for them to comply with the **claim destination fundamental (D5)**. As described above, even truthful claims from trustworthy claim issuers can otherwise be misused.

For an attribute claim the delegator must validate that the claim actually refers to the entity in question (e.g., the peer) to satisfy the **holder relation fundamental**. E.g., when an attribute validation for the peer is conducted by a third party, the endpoint must check that it really relates to the peer (D6).

The claim statement must only be used after all delegation tasks were performed. Figure 5.13 gives an overview of the delegation tasks D2 to D6. A summary of all delegation tasks is given in section 5.4.3.

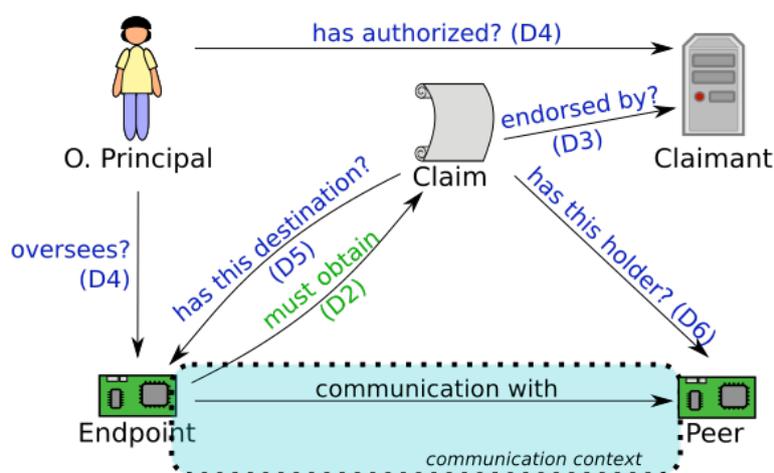


Figure 5.13.: Delegation Tasks

#### 5.4.2.4. Evaluation

To satisfy the **claim completeness fundamental**, it might be necessary to evaluate claims from various sources (D8)<sup>5</sup>. If the endpoint has several claims concerning a peer or a claim issuer, it must know how to handle them; e.g., only the most recent claim may be considered. It is also possible that an endpoint may only believe a claim that was stated by at least two authorized claim issuers.

<sup>5</sup>Task D7 was intentionally left out (see section 5.4.3).

### 5.4.3. List of Delegation Tasks

As stated above, obtaining authorization rules from an OVP is a task delegation process. **D1** to **D6** correspond to **A1** to **A6**. Task **D8** is an evaluation task corresponding to task **A8**. We intentionally left out task **D7** since task delegation does not necessarily require context validation. Task delegation also does not have an enforcement task. An authorization process that protects the overseeing principals' security objectives must enforce all of the OVP's authorization rules. In contrast, while it may be necessary for a secure task delegation that the endpoint considers certain claims, not every received claim may be required. The enforcement can therefore not be mandated.

Figure 5.14 shows an UML 2.0 activity diagram of the necessary data flows of the delegation process. The tasks are not necessarily performed in the depicted order. The claim issuer must validate the authorization of the delegator to delegate the task and to send and receive information concerning the task (task **D0**). The claim issuer must bind the claim information including the output of the task together and endorse this binding (task **D1**). Each claim concerning a subject must then in the expansion node be obtained by the delegator (task **D2**). The claim is validated in the tasks **D3** to **D5**, and, for attribute claims, task **D6**. To perform task **D5**, the endpoint must have information about itself ("me"). The authorization of the claim issuer to state the claim must be validated by conducting the authorization process for the claim issuer (task **D4**). This process equals task **A4** (see figure 5.9). If one of tasks **A3** to **D6** has a negative result, the claim is not valid and must be discarded. All resulting claims are evaluated in task **D8**. Only if all tasks were performed, the resulting statement is to be used.

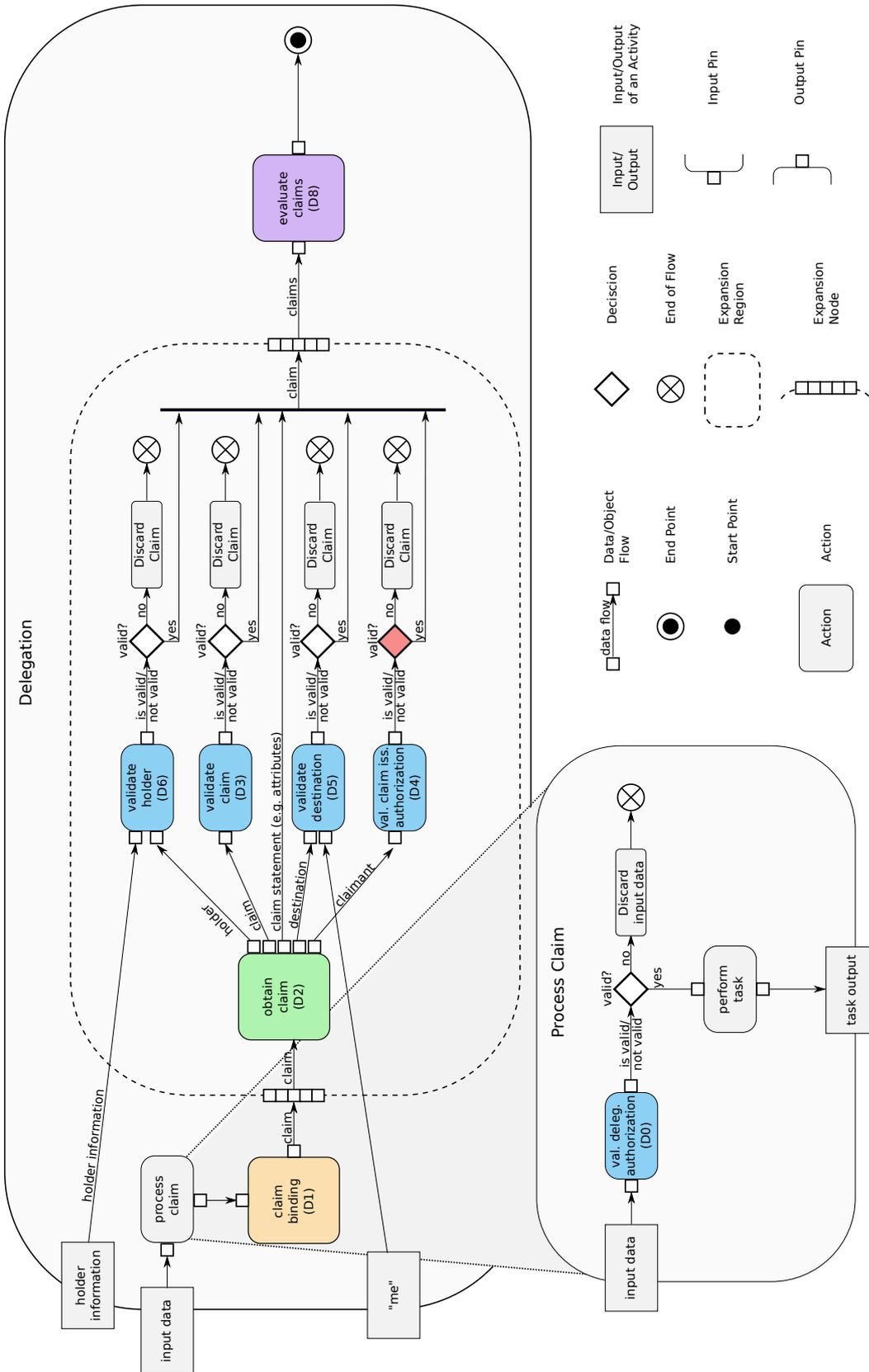


Figure 5.14.: Activity Diagram of Delegation

Delegation increases the complexity for the endpoints. For each task that is delegated, all delegation tasks need to be performed. Nevertheless, the delegation of a task might be easier to accomplish for an endpoint. E.g., it might already have a security association with a certain claim issuer, or it might be easier to establish it.

Summarizing, the tasks for delegation are:

#### Generation

- D0** Validate delegator authorization: the claim issuer must validate that the delegator currently is authorized to delegate tasks to the claim issuer, send the respective input data to the claim issuer, and receive the output data of the task.
- D1** Claim binding: the claim issuer binds the elements of the claim together and endorses them. The claim information includes a) the claim statement, b) the endpoint that is the intended destination of the claim, and c) the holder of the claim, if the claim is an attribute claim. An attribute claim must be bound to a verifier to bind the attributes to the holder. Also, the overseeing principals may need to define additional information to satisfy the delegation fundamentals, e.g., the validity period of a claim or measures against replay protection. All information relevant to the claim must be endorsed by the claim issuer.

#### Provisioning

- D2** Claim distribution: The delegator must obtain all required currently valid claims and respective verifiers.

#### Validation

- D3** Validate Claim: the delegator must validate that the statement and destination of the claim, potential other delegation-related information, and for attribute claims additionally the holder's verifier relate to each other and stem from the same claim issuer.
- D4** Validate claim issuer authorization: the authorization of the claim issuer to state the claim must be validated.

- D5** Validate claim destination: it must be validated that the claim was intended for this delegator.
- D6** Validate holder (for attribute claims): the delegator validates if a certain entity actually is the holder of the claim by ascertaining that the entity in question can be validated with the help of the verifier given in the claim.

Evaluation

- D8** Evaluate claim: all currently valid claims regarding a subject must be evaluated.

## 5.5. Securing Application Data

Application data that is exchanged between endpoints must be protected according to the overseeing principals' authorization rules. Otherwise, the authorization cannot be enforced correctly. The participating endpoints must consider their own authorization rules as well as those of their peers. The exchange of messages can therefore be regarded as task delegation: The application data is the claim statement and must be protected accordingly. The delegation fundamentals apply. For incoming messages, endpoints must satisfy the **claim issuer authorization fundamental** and ensures the authorization of the sender. For outgoing messages, endpoints must comply with the **claim issuer participation fundamental** and validate the receiver's authorization.

## 5.6. Omission of Tasks

Performing unnecessary tasks wastes system resources on endpoints. Also, such tasks may unnecessarily restrict the authorization solution. Our model therefore must only contain tasks that are strictly necessary. In this section, we will analyze the consequences that result if tasks are omitted or performed incorrectly.

The omission of authorization tasks results in flaws in the authorization process that may lead to unauthorized access. If task **A1** is omitted or not performed

correctly, the permissions may contain incorrect or incomplete information. In the worst case, the policy enforcement point may grant access to (send data to or receive data from) an unauthorized entity. The omission of task **A2** leads to incomplete information on the endpoint. As a result, the endpoint may enforce outdated rules and grant access to unauthorized entities.

If task **A3** is omitted, the required pieces of authorization information may not relate to each other or be endorsed by different claim issuers. An attacker may exploit this by introducing own pieces of information to the authorization information to gain unauthorized access. Without task **A4**, the endpoint does not check if the origin of the permission is authorized to provide it. In this case, attackers are able to issue fake permissions.

Task **A5** is necessary for data destination verifiability. Without it, the endpoint does not determine if it is the intended destination of a certain permission. It may then enforce permissions that were meant for a different endpoint. Attackers may use legitimate permissions from authorized issuers to launch man-in-the-middle attacks on the endpoint.

Without task **A6**, the endpoint does not check if the permission actually relates to a certain peer. In this case, the endpoint may communicate with an unauthorized peer. The consequences of omitting task **A7** depend on the respective context. Permissions may, e.g., be enforced for the wrong object or at the wrong time. The result is unauthorized access.

If task **A8** is omitted or not performed correctly, the endpoint's result of evaluating the permissions may be wrong, and entities may be falsely authorized. Even if all previous tasks were performed correctly, the authorization process fails if they are not enforced correctly, i.e., if task **A9** is omitted. Table 5.1 shows the consequences of omitting an authorization task.

If task delegation tasks are not performed correctly, flaws occur in the task delegation process. In this case, endpoints may use wrong, incomplete or outdated claim information. If the endpoint uses such claims in the authorization process, it may receive data from or send data to unauthorized entities.

Name	Description	Effect of Omission
<b>A1</b>	Permission Configuration	Permission may contain wrong or incomplete information.
<b>A2</b>	Obtain Permission	The enforcing endpoint may enforce outdated or incomplete authorizations.
<b>A3</b>	Validate Relations	Attackers may manipulate or forge permissions.
<b>A4</b>	Origin Authorization	Unauthorized attackers may issue permissions.
<b>A5</b>	Destination Validation	Attackers may provide permissions to the wrong endpoint (man-in-the-middle attack).
<b>A6</b>	Holder Authorization	The permission may be associated with the wrong holder.
<b>A7</b>	Context	The permission may be enforced in the wrong context.
<b>A8</b>	Evaluation	The endpoint may enforce outdated or incomplete authorizations.
<b>A9</b>	Enforcement	Unauthorized entities may access and manipulate data.

Table 5.1.: Effects of Authorization Task Omission

For task **D0**, the claim issuer must validate the authorization of the delegator to deliver input to the claim and receive the claim. If the task is omitted or not performed correctly, the claim issuer may accept data from an unauthorized entity to perform a task. In this case, an attacker may provide false data to the claim issuer, which results in a wrong claim. Also, the claim issuer may provide the claim to an unauthorized entity and thus disclose confidential information.

The effects of omitting the tasks **D1** to **D8** are similar to those described above for the corresponding authorization tasks. For the sake of completeness, we nevertheless list these effects below.

Omitting task **D1** entails that the claim is not configured correctly and may thus contain incomplete or incorrect information. As a consequence, the endpoint is not able to correctly perform the security task for which the claim is needed.

If task **D2** is not performed correctly, the endpoint may not have all currently valid claims. It may use incomplete or outdated claims. The consequences depend on the type of claim. In the case of an attribute claim, the endpoint may not have currently valid information about a peer.

The consequence of omitting task **D3** is that the necessary pieces of claim information may not stem from the same claim issuer. Attackers may exploit this flaw by introducing false information to the claim.

Without task **D4**, the endpoint does not check if the claim issuer is authorized. An attacker may exploit this behavior by providing forged claims to the endpoint.

If task **D5** is not performed correctly, the endpoint cannot be sure that it is the intended destination of a claim. An attacker may perform a man-in-the-middle attack on the delegator by presenting a legitimate claim that was meant for a different delegator to the endpoint.

For attribute claims, the endpoint performs task **D6** to determine if the claim refers to a certain peer. If the task is not performed correctly, the claim may be attributed to the wrong peer. If several claims exist and they are not interpreted correctly (task **D8**), the endpoint may use incomplete or outdated claims. An overview of the results of omitting a delegation task is given in table 5.2.

Name	Description	Effect of Omission
<b>D0</b>	Delegator Authorization	Disclosure of confidential information, wrong claim data.
<b>D1</b>	Claim Configuration	Claim may contain wrong or incomplete information.
<b>D2</b>	Obtain Claim	Critical information is missing.
<b>D3</b>	Validate Binding	Attackers may manipulate or forge claims.
<b>D4</b>	Claim issuer Authorization	Unauthorized entities may issue claims.
<b>D5</b>	Claim Destination	Attackers may provide claims to the wrong endpoint (man-in-the-middle attacks).
<b>D6</b>	Holder Authorization	Claim may be associated with the wrong holder.
<b>D8</b>	Evaluation	Critical information may not be considered.

Table 5.2.: Effects of Delegation Task Omission

Task **An1** is required for task **A6** unless entity-independent authorization is used, and is also necessary for task **D6** for attribute claims. If task **An1** is omitted or not performed correctly, the holder's attributes are unknown or wrong, and the **correlation fundamental** and the **holder relation fundamental** cannot be satisfied.

We can see that the omission of any task causes a potential vulnerability; all authorization tasks are necessary for a secure authorization process, and all delegation tasks must be performed for a secure task delegation. To achieve continuous

security for a system, all entities that participate in the system must satisfy all authorization and delegation fundamentals in every communication.

## 5.7. Minimal Set of Tasks

Endpoints that participate in protecting the security policies of their overseeing principals are responsible for the tasks **A2** to **A9**. Task **A1** is the OVPs' responsibility. As shown in section 5.4, some of the tasks required for the authorization can be delegated to entities that are authorized to perform them for the endpoint.

An endpoint that cannot enforce authorization rules is not able to participate in the protection of its OVPs' security objectives. The enforcement of the authorization rules (task **A9**) must be performed by the endpoint itself and cannot be delegated.

Other tasks can be delegated if the required input data for the task is known to the claim issuer or can be securely transmitted to it, and if the output data of the task is securely transmitted back to the delegator as the claim statement. Since task delegation increases the complexity for the delegator, it must be determined if the delegation really is useful and necessary. Delegation tasks can themselves be delegated. Endpoints are responsible for the tasks **D2** to **D8**.

Endpoints must obtain the authorization rules (task **A2**) they need to enforce. However, it is possible to obtain simplified authorization rules from a claim issuer that is authorized to provide them. The claim issuer in this case obtains, validates and evaluates authorization rules for the endpoint. The claim issuer must then bind the holder, destination, context and issuer of these simplified permissions to the claim so that the delegator can perform the remaining authorization tasks.

The validation of the permission relations (task **A3**), and validating the issuer authorization (task **A4**) can each be delegated, but it must be ensured that the issuer that is authorized in task **A4** is the origin of the binding in task **A3**. The task delegation process requires that the endpoint validates the claim (task **D3**) and that this claim issuer is authorized to state the claim (task **D4**). The endpoint

must therefore be able to validate the relations of some claim or permission, and determine by itself the authorization of some entity to state a claim or permission. The OVP as the main authority is the root of authorization chains. Therefore, endpoints must at least be able to perform the authorization tasks concerning their own OVPs<sup>6</sup>.

Endpoints are in most cases not able to perform task **An1** by themselves (see also section 5.4). The task is therefore usually delegated. The endpoint must be able to determine whether the resulting claim actually refers to the peer in question (task **D6**). Therefore, the claims resulting from delegating task **An1** always have to be attribute claims. To comply with the **claim completeness fundamental**, endpoints usually obtain the most recent attribute claim at the beginning of the communication.

As stated above, the endpoint must know with which attributes it is itself identified to be able to perform the destination validation for task **A5**. It may need to rely on a third party to find out or negotiate what these attributes are. They may even differ for different peers. But the endpoint must check by itself whether the attributes mean that it actually is the endpoint that must enforce the authorization rules. Since the delegation encompasses the claim destination validation (**D5**), the endpoint must have a common understanding about its attributes with some claim issuers, and at least with its overseeing principals.

Endpoints can delegate the context validation (task **A7**) if the type of the context does not forbid it. E.g., if the authorization rule constrains a permission to a certain time, it might be necessary that the endpoint validates by itself if the context is valid since it might take time to get a result from the claim issuer.

Summarizing, endpoints that participate in the protection of their overseeing principals' security objectives must perform the authorization tasks **A2** to **A9** for some claim issuers. Since at least the attribute validation is usually delegated, endpoints must also be able perform the delegation tasks **D2** to **D8**. As we have shown in section 5.6, omitting a task leads to vulnerabilities in the security solution. Since no task can be omitted, the burden on constrained devices can only

---

<sup>6</sup>In actual systems, the validation of the principal's authorization will often be performed by assuming that the endpoints' software and configuration files were provided by the overseeing principal.

be eased by reducing the number of security associations that the devices must establish and handle.

## 5.8. Satisfying the Fundamentals

The fundamentals cannot always be completely satisfied. **Unlike our theoretical model, an actual implementation can therefore never be perfectly secure.** A fundamental is *sufficiently satisfied* if a degree of security is reached that suffices for the application.

The **rule completeness fundamental** and the **claim completeness fundamental** require that authorization rules and claims are always up to date. In an actual application this cannot be achieved: continuous updates are inconvenient and expensive. Also, even the provisioning of the necessary data to the endpoint, and the validation and evaluation of claims and rules takes time. Nevertheless, security solutions can reach a reasonable level of security. The peril that an unauthorized entity can access data because of outdated authorization rules or claims increases the longer the rules or claims are not updated. To sufficiently satisfy the fundamentals, the damage that an attacker can cause because the endpoint does not have the current rules must be limited. **Overseeing principals must decide for their data and devices how much the security solution may deviate from the rule completeness fundamental and the claim completeness fundamental, i.e., how long the timespan may at most be until a device is informed about changes in authorization rules and claims.** Security solutions should enable OVPs to decide about these timespans and must ensure that they are enforced. Security solutions often provide configurable lifetimes or expiration dates for their permissions and claims. Revocation mechanisms enable endpoints to react to unplanned changes. **A solution where authorization rules are never updated does not sufficiently satisfy the rule completeness fundamental. Also, an approach where claims are never updated does not sufficiently satisfy the claim completeness fundamental.**

Endpoints that perform authorization usually use cryptography to satisfy fundamentals, e.g., the correlation fundamental. Cryptographic mechanisms are not

perfect. Their security depends on the underlying algorithms and the length of the used keys. Cryptographic mechanisms that are broken or use an insufficient key length are not able to sufficiently satisfy the fundamentals (see also section 10.1). Overseeing principals must decide if a certain mechanism is able to sufficiently satisfy the fundamentals in their application.

Cryptographic algorithms come with an “expiration date”: Some of the algorithms that are considered to be secure today will likely be broken in a few years. Even if attackers find no shortcuts and have to try all possible keys, faster processors are developed every year. Over time, the key length of an algorithm may no longer be sufficient. Also, new developments may suddenly render certain algorithms ineffective. E.g., if quantum computers become more powerful, asymmetric cryptography is expected to no longer be secure (see also section 10.1.2). An outdated algorithm must be changed to satisfy the fundamentals.

Given sufficient time and incentive, attackers will be able to discover the utilized keying material eventually, e.g., by trying all possible keys, by managing to compromise the device where the keying material is stored, or even due to user errors. If keying material is compromised, fundamentals such as the **correlation fundamental** and the **holder relation fundamental** can no longer be satisfied.

The potential damage and the risk that the keying material is compromised increases over time (NIST SP 800-57, p. 35). The cryptoperiod, i.e., the period where the same keying material is used, should therefore be limited. NIST SP 800-57 gives recommendations for the length of cryptoperiods of various key types (NIST SP 800-57, pp. 45–47). A security solution that does not allow regular updates of cryptographic keying material and algorithms can no longer sufficiently satisfy the authorization fundamentals after the expiration date was reached.

That an update mechanism for attribute and authorization claims exists at all is very important for the security of a solution. Adjusting the frequency of updates is usually much easier than retrofitting an entire update mechanism. As we will see in the examples below (section 5.10) and in the analysis of the DTLS profile in section 11.2, protocols are often vulnerable because update mechanisms for claims or rules are not considered at all.

Nevertheless, a more fine-grained method for determining the update frequency that an application requires would be very useful. It would help solution designers customizing validity periods and timeouts or even choosing a suitable mechanism (or combination of mechanisms) for their application, as certain update mechanisms may become too expensive if their frequency is increased.

The acceptable update latency for authentication and authorization information is influenced by the following factors:

- frequency of changes concerning the authorization (authorized claim issuers, overseeing principals, authorized peers),
- cryptographic algorithms and parameters (e.g., short keys need to be changed more frequently),
- frequency of communication,
- “attackability”: environment properties,
- attacker incentive,
- protection requirements (cost of a successful attack).

An important aspect is the the frequency with which authorized claim issuers—including the overseeing principal of the endpoint—change. Claims that were issued by these claim issuers may no longer reflect the overseeing principals’ decisions and therefore must become invalid. Also, the update frequency depends on the security of cryptographic algorithms and their parameters. The frequency with which keys need to be changed to achieve a secure solution depend on the algorithm. Also, shorter keys must be changed more frequently since attackers need less time to try every key. The security of data that is stored on an endpoint does not only depend on the security of the solution itself but also on the environment. E.g., in some scenarios the attacker may more easily perform physical attacks on a device than in others. If endpoints are compromised, their communication partners need to be informed as soon as possible. Another important aspect for the update frequency is the likelihood that an endpoint is attacked. It depends not only on the security of the system but also on the incentive of the attackers: what do they have to gain and how much does the attack cost them? And on the other side, how much protection does the system require? How much

damage can be done by an attacker and how much would that cost the overseeing principals?

A method for determining acceptable update latencies for specific types of applications is not in scope for this thesis. Since it would be very useful to evaluate the security of a solution, it is a topic for future work.

## 5.9. Model Completeness

An attacker may need only a single vulnerability to attack a system. Solution designers and implementers therefore must ensure that the security solution does not contain any vulnerabilities (Anderson 2001, pp. 4–5). To be useful, the model must cover the whole authorization process. Its quality additionally depends on the number of distinct vulnerabilities it allows to detect. We would like our model to detect every possible vulnerability in the real world. Unfortunately, it is very difficult to prove that this actually is the case. The model aims at a categorization of vulnerabilities with useful granularity: it must provide a complete checklist, which is also difficult to prove. Below, we will discuss how our model covers the authorization process.

Vulnerabilities in authorization and authentication protocols occur if the messages do not contain all information required by the endpoint. In ANAZEM, claim issuers must bind claims and permissions to all relationships that are relevant for the authorization and authentication process, and endpoints must validate all of these relationships in their own communication. **Our model is therefore complete in the sense that it covers all relationships that are relevant for a communication.** With the **context fundamental**, factors that further restrict the authorization are also covered.

In ANAZEM, each piece of information that influences the authentication and authorization must be endorsed by a claim issuer that is authorized by the endpoint's OVP. Claim issuers must correctly perform the tasks that were delegated to them. **Our model is therefore complete in the sense that each relevant information must continuously trace back to the overseeing principals as the authorities for their data and devices.**

ANAZEM demands the completeness of permissions and claims, and endpoints are required to consider all currently valid permissions and claims. **Our model is therefore complete in the sense that all relevant and currently valid information is considered in the authentication and authorization process.**

Finally, authorization must be conducted for every sent and received piece of information to satisfy the fundamentals of our model. **Our model is therefore complete in the sense that authorization must always be conducted.**

We argue that thereby all relevant aspects of the authenticated authorization process are covered. Therefore, ANAZEM is complete since its fundamentals and tasks address all relevant aspects of an authenticated authorization process.

As explained above, these considerations cannot completely dispel all doubts about the completeness of the model. We need to test the model in the real world to determine its worth. In the next section, we will test the model on known vulnerabilities of protocols. In chapter 6, we will compare our model to the BAN logic to validate if it is as capable of finding problems. In chapter 7, we will analyze the TLS protocol and show that even for established security protocols the fundamentals can point out in which areas the protocol has difficulties to satisfy the fundamentals.

## 5.10. Detection of Typical Vulnerabilities of Protocols

In this section we analyze the usefulness of ANAZEM to discover vulnerabilities in protocols. To do so, we apply our model to protocols with known flaws to show that the underlying problems are covered by the model. For our analysis we use the paper *Programming Satan's Computer* (Anderson and Needham 1995) which covers a wide range of typical vulnerabilities in protocols. Most of the protocols in the paper target authentication, which we will analyze using the delegation tasks.

To simplify the analysis, we focus on the main problem of the protocol even if several other problems can be found in its design. We do not include the attack

on the Tabayashi-Matsuzaki-Newmann scheme in our analysis because it is not caused by a protocol flaw but by the calculations required by the cryptographic mechanisms. As discussed in section 5.8, broken cryptographic mechanisms are not able to satisfy the fundamentals. However, ANAZEM cannot directly help to detect cryptographic problems.

ANAZEM not only covers all mentioned vulnerabilities, but even lists additional fundamentals and tasks that were not addressed by Anderson and Needham.

### 5.10.1. Simple Bank Fraud

The first case describes an incident with teller machines in a bank. The bank allowed its customers to draw money using a card with a magnetic strip that contained a Personal Identification Number (PIN) and the number of the account that is charged. Attackers were able to change the account number on the card and were thus able to steal money from someone else's account.

If we apply ANAZEM to this case, the teller machine is the endpoint and the customer is the peer. The card is used to prove the peer's authorization to access the account on the card. The information on the card can therefore be considered as an authorization claim. The bank did not bind the verifier, i.e. the PIN, to the account number on the card, i.e., the permission to access the bank account. Task **A1** therefore was not performed correctly. As a result, the telling machine is not able to determine if the account was specified by an authorized entity, i.e., it was not able to perform the tasks **A3** and **A4**. The protocol failed because it did not satisfy the **principal fundamental**.

### 5.10.2. Pay-Per-View Television

A wide range of attacks targets video decoders that are used to decrypt protected Pay-TV content. In one example, a certain message is used to inform the decoder that the customer stopped paying and therefore is no longer allowed to receive the service. If attackers manage to block this message, the customer can continue

to view content. From the view of our model, the decoder is not able to perform task **A2**. The stop message represents an authorization rule which the decoder did not receive. The **reference monitor fundamental** and the **rule completeness fundamental** were not satisfied.

For the other Pay-TV examples mentioned by Anderson and Needham, we have too little information to apply our model.

### 5.10.3. Wide Mouthed Frog

In the wide mouthed frog protocol, a third party S is used to securely transfer a key that is generated by A to B. S shares  $K_{AS}$  with A and  $K_{BS}$  with B. The protocol is specified as follows:

1.  $A \rightarrow S : A, \{ T_A, B, K_{AB} \} K_{AS}$
2.  $S \rightarrow B : \{ T_s, A, K_{AB} \} K_{BS}$

In ANAZEM, S performs a delegation task for B and provides it with an attribute claim. The delegation fundamentals apply. S must protect the security objectives of B's OVP to satisfy the claim issuer participation fundamental and therefore must ensure that message 1 is protected (see also section 5.5).

As already described in section 5.2.1, S is not able to detect if it is the intended destination of the message from A; the encryption is not sufficient to validate the destination because the used key is shared by A and S. S therefore cannot perform task **D5** for this message. As a result, S can be tricked into accepting its own message 2 as message 1 and reply to it with a new timestamp. Attackers could thereby keep the session key alive indefinitely. The protocol is vulnerable because the **claim destination fundamental** is not satisfied. Additionally, the protocol has no protection against replay attacks: S does not perform task **D3**, and is therefore not able satisfy the **claim origin fundamental**.

#### 5.10.4. Challenge-Response

With a challenge-response mechanism an entity can prove that it has certain keying material. An endpoint may, e.g., send a challenge that the peer must respond to before the endpoint allows it to log into a system.

The following protocol attempts to realize a challenge-response protocol for scenarios where A and B do not share a secret but each of them shares a key with S. B sends a nonce to A and only gets a correctly encrypted nonce back from S if A was able to use the keying material shared with S:

$$\begin{aligned}
 A &\rightarrow B : A \\
 B &\rightarrow A : N_B \\
 A &\rightarrow B : \{ N_B \} K_{AS} \\
 B &\rightarrow S : \{ A, \{ N_B \} K_{AS} \} K_{BS} \\
 S &\rightarrow B : \{ N_B \} K_{BS}
 \end{aligned}$$

S authenticates A for B. The delegation tasks apply. If we consider the last message to be a claim from S, we can see that essential information is missing; it does neither provide the destination nor the holder of the message. The nonce  $N_B$  is not sufficient to bind the information in message 4 to message 5, since it was transported in the clear in message 2, and therefore may be known to an attacker. Since the holder is missing in the last message from S, it is not clear to which of B's peers the message applies. B therefore cannot perform task **D6**, and the **holder relation fundamental** is not satisfied. An attacker may exploit this omission to impersonate A. The missing destination means that B cannot perform task **D5**; the **claim destination fundamental** is not satisfied. B may therefore be tricked into accepting nonces that were meant for a different endpoint.

The protocol does not enable A to authenticate B. An additional mechanism is required to achieve this.

#### 5.10.5. Denning-Sacco

Denning and Sacco specified a protocol that aimed at securely distributing a symmetric key with the help of a third party (S). S also issues certificates to endpoints.

Each certificate (CA and CB) contains the name, public key, expiration date and permissions of an endpoint, and is signed by S.  $K_A^{-1}$  depicts a signature made with A's private key. A can use the protocol to request the symmetric key for the communication with B:

$$\begin{aligned} A &\rightarrow S : A, B \\ S &\rightarrow A : CA, CB \\ A &\rightarrow B : CA, CB, \{ \{ T_A, K_{AB} \} K_A^{-1} \} K_B \end{aligned}$$

If we consider the third message to be a claim from A, we can see that the destination of the message is only specified in the certificate which is not endorsed by A. A does not correctly perform task **D1**, and B is therefore not able to perform task **D5**. The **claim destination fundamental** is not satisfied.

As discovered by Abadi and Needham, this vulnerability can be exploited by B (Abadi and Needham 1994, p. 125): A malicious B may decrypt the third message and re-encrypts it with, e.g., C's public key. Also, C replaces the its own certificate with the C's certificate to get the following message:

$$B \rightarrow C : CA, CC, \{ \{ T_A, K_{AB} \} K_A^{-1} \} K_C$$

Because of the certificate given in the message, C assumes that it can use  $K_{AB}$  to securely communicate with A: the intended holder is bound to  $K_{AB}$  because the key is signed with A's private key. But as the destination of the message is not cryptographically bound to the message, C can not use  $K_{AB}$  to perform task **A6** concerning A. As a consequence, B can use  $K_{AB}$  to impersonate A in the communication with C.

In their paper, Anderson and Needham state that B specifies its own certificate instead of A's in the message to C:

$$B \rightarrow C : CB, CC, \{ \{ T_A, K_{AB} \} K_A^{-1} \} K_C.$$

In this case,  $K_{AB}$  is no longer bound to A as the intended holder, but to B. C would thus be able to notice that the message stems from B. Also, C might use the permissions given in the certificate to determine which data B is allowed to access.

### 5.10.6. Middleperson Attack

The following protocol was designed to let two endpoints prove that they can use certain asymmetric keys. The respective certificates are expected to be exchanged prior to the communication.

$$\begin{aligned} A &\rightarrow B : \{ N_A, A \} K_B \\ B &\rightarrow A : \{ N_A, N_B \} K_A \\ A &\rightarrow B : \{ N_B \} K_B \end{aligned}$$

An attacker can impersonate A in the communication with B and act as B for A. If we apply ANAZEM, we can see that the messages are not endorsed by the claim issuer. They are therefore not integrity-protected. The endpoints are therefore not able to perform task **D3** and task **D4**. Since an attacker can impersonate each endpoint, the **claim origin fundamental** is not satisfied.

### 5.10.7. CCITT

The protocol depicted below is designed to protect the integrity of  $X_a$   $X_b$ , and the confidentiality of  $Y_a$  and  $Y_b$ :

$$\begin{aligned} A &\rightarrow B : A, \{ T_a, N_a, B, X_a, \{ Y_a \} K_b \} K_a^{-1} \\ B &\rightarrow A : B, \{ T_b, N_b, A, N_a, X_b, \{ Y_b \} K_a \} K_b^{-1} \\ A &\rightarrow B : A, \{ N_b \} K_a^{-1} \end{aligned}$$

The protocol is no authentication protocol, but if we consider the messages to be claims, we can still apply ANAZEM. Anderson and Needham state that the sender may encrypt unknown data since  $Y_a$  and  $Y_b$  are first encrypted and then signed. Applying our model, we consider  $Y_a$  and  $Y_b$  to be the claim statement. If the sender signs a claim that it did not validate, e.g., by adopting data encrypted by someone else, the sender violates the **claim issuer participation fundamental**. The task **D1** was not performed correctly. The given protocol does not indicate that such a violation has taken place.

### 5.10.8. Needham-Schroeder Protocol

The Needham-Schroeder Protocol (Needham and Schroeder 1978) aims at providing mutual authentication between two peers. The protocol was often discussed because several vulnerabilities were found over the years.

In the Needham-Schroeder protocol, a third party  $S$  performs attribute validation for both  $A$  and  $B$ . The delegation fundamentals apply. Needham and Schroeder defined a symmetric-key version of the protocol and an approach that uses asymmetric keys. Needham and Schroeder specified the symmetric version as follows:

1.  $A \rightarrow S : A, B, N_A$
2.  $S \rightarrow A : \{ N_A, B, K_{AB}, \{ K_{AB}, A \} K_{BS} \} K_{AS}$
3.  $A \rightarrow B : \{ K_{AB}, A \} K_{BS}$
4.  $B \rightarrow A : \{ N_B \} K_{AB}$
5.  $A \rightarrow B : \{ N_B - 1 \} K_{AB}$

Message 2 contains  $S$ 's claim for  $A$  and the claim for  $B$ .  $A$  can determine the freshness of the message with the help of the nonce given in the claim. The nonce must be sufficiently random and  $A$  must keep track when it sent message 1.  $A$  is then able to perform task **D2**. For  $B$ , the situation is different. The claim for  $B$  does not contain any information that would enable  $B$  to determine the freshness.  $B$  cannot perform task **D2** and the **claim completeness fundamental** is not satisfied. The protocol is vulnerable to replay attacks.  $B$  may thereby be tricked into using outdated keying material. As a result,  $B$  is not able to determine if it is actually communicating with  $A$  (task **D6**). The **holder relation fundamental** is not satisfied. The vulnerability was discovered by Denning and Sacco (Denning and Sacco 1981).

The public key protocol is defined as follows:

1.  $A \rightarrow S : A, B$
2.  $S \rightarrow A : \{ K_B, B \} K_S^{-1}$
3.  $A \rightarrow B : \{ N_A, A \} K_B$
4.  $B \rightarrow S : B, A$

5.  $S \rightarrow B : \{ K_A, A \} K_S^{-1}$
6.  $B \rightarrow A : \{ N_A, N_B \} K_A$
7.  $A \rightarrow B : \{ N_B \} K_B$

In the public key version, both claims from S do not contain any freshness information. Therefore A and B can both not determine if the claim is up to date for task **D2**. The protocol is vulnerable to replay attacks on A and B.

In message 3, A sends a nonce to B that is encrypted with B's public key. The message is not signed and B therefore cannot validate that it actually stems from A. In message 6, B sends the nonce back to A, encrypted with A's public key. The message is again not signed. A and B cannot perform task **D6** and the **holder relation fundamental** is not satisfied. The protocol is vulnerable to man-in-the-middle attacks: an attacker C can impersonate B in front of A, and act as A when communicating with B. The vulnerability was discovered by Lowe (Lowe 1996).

### 5.10.9. Decryption and Signature

In 1992, Woo and Lam present a variation of the Needham-Schroeder public key Protocol. Unfortunately, the protocol flow as cited by Anderson and Needham (Anderson and Needham 1995, p. 10) differs from the protocol defined by Woo and Lam (Woo and Lam 1992b, p. 47). In the original protocol, A is not specified in message 5 and 6,  $N_B$  is missing from message 6, and instead of  $N_A$ ,  $N_B$  is transmitted in message 7. Below, we depict the original protocol flow:

1.  $A \rightarrow S : A, B$
2.  $S \rightarrow A : \{ K_B, B \} K_S^{-1}$
3.  $A \rightarrow B : \{ N_A, A \} K_B$
4.  $B \rightarrow S : A, B, \{ N_A \} K_S$
5.  $S \rightarrow B : \{ K_A, A \} K_S^{-1}, \{ \{ N_A, K_{AB}, B \} K_S^{-1} \} K_B$
6.  $B \rightarrow A : \{ \{ N_A, K_{AB}, B \} K_S^{-1}, N_B \} K_A$
7.  $A \rightarrow B : \{ N_B \} K_{AB}$

The protocol tries to ensure with the help of the nonces that the communication partners can actually currently use their respective keying material that proves them to be A and B. S knows A's and B's public key, and A and B must obtain this knowledge from S. In message 5, B receives A's public key from S. The keying material does not contain any information that enables B to determine if it is fresh, and B therefore cannot perform task **D2**. The keying material was bound to A's identifier, i.e., the holder. The claim statement is therefore set. It is not bound to a destination, but we can assume that A's public key is the same for all entities. B therefore can perform task **D5**. S signed the first part of the message, but B can not be certain that it was actually sent by S since the protocol has no replay protection. Therefore, B cannot perform task **D3** and the **claim origin fundamental** is not satisfied. B is also not yet able to perform task **D6**, because A did not show that it is able to use the keying material in the certificate. That is the purpose of the next steps: B encrypts its message to A with A's public key. It includes the nonce  $N_B$  and the session key  $K_{AB}$ . A returns  $N_B$  in message 7 which is encrypted with  $K_{AB}$ . Since  $K_{AB}$  was first protected with  $K_B$  and then with  $K_A$ , A can only know  $K_{AB}$  if it has either of these keys. Unfortunately, B cannot determine if message 5 containing  $K_A$  was a replay.  $K_A$  can therefore not be used to validate the holder (task **D6**), i.e., B cannot authenticate A. An attacker that got hold of A's keying material may thus communicate with B even if S knows that the keying material was compromised.

A gets B's public key in message 2. The key is bound to B's identifier, but the message again does not contain a timestamp. A therefore cannot determine if the claim from S is fresh. It may be a replay, and not actually stem from S. Since A cannot believe the claim, it cannot use it to validate that message 6 actually stems from B. A therefore cannot authenticate B.

#### **5.10.10. Addressed Tasks and Fundamentals**

We have shown that all protocol vulnerabilities described in the paper are caused by the omission of necessary authorization or delegation tasks. Therefore, designers of the protocols would have been able to detect the problems if they had used ANAZEM.

## 5.11. Conclusion

In this chapter we introduced the authorization and delegation fundamentals representing the requirements that an authorization process must meet to provide perfect security. From the fundamentals, tasks are derived that must be performed to ensure that the fundamentals are considered in every phase of the authorization process. We showed that nested within the authorization process are authentication and additional authorization processes. Authorization can only be secure and thus effective if these processes work together correctly. Authorization information must be securely transmitted from the overseeing principal who defines the authorization rules to the endpoint that enforces them. Every piece of data that influences the authorization must securely be obtained, and every entity that provides such data must be authorized by an OVP, including the OVPs themselves.

The authorization and delegation fundamentals aim at perfect security. An actual implementation is not able to completely satisfy the fundamentals. The quality of a security solution depends on the degree to which it satisfies the fundamentals. A solution that does not sufficiently satisfy the fundamentals is not suitable to protect the integrity and confidentiality of application data.

For effective authenticated authorization, all authorization and delegation tasks must be performed and all authorization and delegation fundamentals must be considered. Protocols often only perform some of the tasks and rely on other protocols to perform others. In this case, the protocols must point out what additionally has to be done for a complete authorization process and should discuss how the remaining tasks can be realized. ANAZEM helps to discover hidden requirements of the protocols. The list of tasks enables solution designers to explicitly state which tasks are covered by their solution and which tasks are expected to be additionally performed. If the various building blocks are thus explicitly defined, they can be more easily combined to build a secure, gapless solution.

ANAZEM covers all relevant aspects of an authenticated authorization process. By using the tasks as a checklist, the model facilitates the detection of vulnerabilities in security protocols. We have shown for a wide range of protocol vulnerabilities that each of the problems occurred because necessary tasks were omitted

by the participants; as a result not all fundamentals were satisfied. By applying our model, solution designers and implementers are able to detect vulnerabilities and to design secure authentication and authorization solutions. The model provides us with a better understanding about detected vulnerabilities, as we can identify the cause of the problem. The terminology that we provided with our model facilitates the discussion about vulnerabilities as well as about abilities and requirements of protocols.

## 6. Comparison with the BAN Logic

In chapter 5 we introduced our authenticated authorization model ANAZEM. To show the capabilities of our model we will now compare it to the BAN logic. This logic, which is named after its authors Burrows, Abadi and Needham, is a way to determine if two entities are entitled to believe after an authentication “that they are communicating with each other and not with intruders” (Burrows et al. 1990, p. 19). To accomplish this, the BAN logic uses statements and derivation rules. It focuses on the analysis of authentication protocols and does not explicitly mention authorization. It can be compared to the delegation part of ANAZEM.

Burrows et al. analyzed the authentication protocol Kerberos as introduced by Miller et al. (Miller et al. 1987) with the BAN logic (Burrows et al. 1990). We will perform our own analysis of the protocol using our ANAZEM and compare the results. We will show that ANAZEM is more fine-grained and able to detect more problems than the BAN logic.

In Kerberos, a Kerberos Key Distribution Server called *S* is used to negotiate a shared secret between two communication partners *A* and *B* (Burrows et al. 1990, p. 25). If we apply ANAZEM, *S* is performing attribute validation (task **An1**) as a delegation task for *A* and *B*. The goal is to securely distribute  $K_{AB}$  to *A* and *B*, which will act as a verifier for *A* to *B* and as a verifier for *B* to *A*. To securely delegate a task in ANAZEM, the Kerberos clients (*A* and *B*) as the delegators must perform the tasks **D2** to **D8**. Claim issuer *S* must perform task **D0** and task **D1**, and the delegated task, in this case **An1**. The complete comparison can be found in Appendix A.

## 6.1. Discussion

The most noticeable problem of the BAN logic is that – in contrast to ANAZEM – it does not examine both sides of a message exchange: the analysis is only made for the receiver’s side. The confidentiality of messages is therefore not considered. Our model emphasizes the need to consider all participants of a communication and facilitates the detection of confidentiality problems in protocols.

ANAZEM’s fundamentals and tasks facilitate the detection of design flaws if they are used as a check list. We thereby detected that the authenticity and data destination verifiability of Kerberos could be improved. Kerberos uses symmetric keys to protect the data destination verifiability and authenticity of messages. Since symmetric keys are shared between the communication partners, it is not possible to exclusively identify an entity with a symmetric key. Additional measures are required to protect the security objectives. Kerberos often relies on the fact that messages from one communication partner always differ from the messages of the other. That these messages differ from each other is therefore a requirement. This requirement is not explicitly mentioned. The protocol may break when changed or extended, if the protocol designers or the implementers are not aware of the requirement. Explicitly mentioning the source and/or destination of the message improves the self-descriptiveness of messages and increases the security of the protocol.

Our analysis shows that some tasks that are necessary for a secure task delegation are not performed by Kerberos. The assumptions made by Burrows et al. indicate that these tasks are required. A and B must have obtained authorization rules for S from their respective overseeing principals (tasks **A2** to **A5**, **A7** and **A8**), which is similar to Burrows et al.’s Assumption 4 and 8. Also, A and B must securely have received  $K_{AS}$  and  $K_{BS}$ , respectively (task **An1**), which equals assumptions 1 and 6 by Burrows et al. S must securely have obtained the respective shared keys to communicate with A and B (task **An1**, Assumptions 2 and 7). Also, S must be able to perform the authorization tasks for A and B, which is not considered by Burrows et al.

The BAN logic requires that the receivers of a message determine if the message is fresh. With their assumptions 5, 9 and 10, they define that A and B believe

that the timestamps in the protocol are fresh. ANAZEM distinguishes between replay protection for the **correlation fundamental**, and validity which is required for the **claim completeness fundamental**. Although these goals are similar, the measures for reaching the goals differ. For replay protection, nonces may be used that make messages distinguishable. Claim completeness requires the endpoint to know at which time the claim is supposed to be no longer valid. Both goals must be considered for a secure task delegation.

ANAZEM considers requirements of authorization and delegation protocols that cannot be met by the protocols themselves. To comply with our model, entities that provide information that influences the authorization or task delegation must be authorized by the overseeing principals (**claim issuer authorization fundamental**). Kerberos requires the communicating endpoints to be time-synchronized with each other. Entities must be authorized by the endpoints' overseeing principals to provide time information.

## 6.2. Conclusion

The analysis of Burrows et al. concludes that after performing Kerberos, given the listed assumptions are true, A and B are entitled to believe that they can communicate securely using  $K_{AB}$ . Our analysis shows that Kerberos can be used to sufficiently satisfy the delegation fundamentals as long as the tasks that the protocol does not consider are conducted correctly outside the protocol, i.e. that the assumptions given in our analysis are correct.

In ANAZEM, the delegation is complete when A and B validate that their communication partner is actually able to use  $K_{AB}$ . If all fundamentals were satisfied, A can then be sure that it communicates with an entity with the attribute B and B has ascertained that it talks to the entity with attribute A.

Burrows et al. implicitly include the aspect of authorization in their logic with their control statement. If this assumption is not validated, the effectiveness of the authorization process is in jeopardy. ANAZEM therefore requires that this authority is first established: an endpoint must not just assume that an entity is

its overseeing principal but is required to first obtain this knowledge and validate it.

The analysis of Burrows et al. does not consider all fundamentals for all parts of the communication and is therefore not able to discover all problems that a protocol may have. In particular, the authors do not analyze whether an entity should be able to read certain messages. The BAN logic therefore does not cover confidentiality problems: it is not able to discover if an unauthorized entity can get hold of  $K_{AB}$ . To achieve this, the sender of a message needs to validate the authorization of the receiver to get it.

Data destination verifiability is also not considered by Burrows et al. By applying ANAZEM to Kerberos, we can see that the data destination verifiability and the authenticity of messages can be improved by explicitly mentioning the source and/or intended recipient in the message.

The BAN logic considers the freshness of authentication but does not distinguish freshness and validity. Also, problems that occur if an endpoint is compromised are not covered. Thus, the analysis does not raise awareness for these problems and is imprecise. Our analysis shows that Kerberos' ability to satisfy the **claim completeness fundamental** can be improved by only using short-lived timestamps or by introducing a revocation mechanism.

The analysis shows that ANAZEM is more fine-grained and can detect more security problems than the BAN logic. While our approach is more accurate, the BAN logic has the advantage that it is closer related to the utilized mechanism and thus may be easier to apply. Our model does not state which mechanisms can be used to satisfy the fundamentals and therefore requires deep knowledge about security mechanisms. It is not limited to certain authentication or authorization mechanisms.

Security protocols can never be 100% secure. The quality of the protocol depends on the degrees to which each authorization and delegation fundamental is satisfied. ANAZEM helps to discover implicit requirements of security protocols. E.g., in the analyzed Kerberos protocol, certain messages must differ to achieve authenticity and data destination verifiability. Also, Kerberos requires that the

participating endpoints use a time synchronization mechanism and accept time information only from authorized claim issuers. By uncovering these requirements, our model can assist developers and users in deploying a secure solution.

## 7. Example: Transport Layer Security in the Web

In chapter 6 we compared ANAZEM to the BAN logic, and showed that our approach can detect problems that the BAN logic is not able to discover. We will now use the model to perform an exemplary security analysis of a typical Web scenario. TLS is widely used for secure communication in the Web and is often used by authorization solutions, e.g., to securely transmit the required information for authorization. It provides various means for authentication. In the Web, the most common approach is to make use of the X.509 Public Key Infrastructure (PKIX, RFC 5280). Both one-sided or mutual authentication are possible. TLS aims at establishing a secure channel between two communicating parties, a client and a server, that prevents attackers from eavesdropping and undetected altering or forging of messages. The main security objectives TLS attempts to protect are the confidentiality and integrity of data (RFC 5246, p. 95). To achieve them, TLS has two layers: the Handshake Protocol where the communicating parties exchange information for the establishment of a secure channel, and the Record Protocol that uses the channel to securely transmit data.

TLS is no authorization protocol. Nevertheless, TLS is designed to prevent unauthorized access to the data that is transmitted using the Record Protocol: to provide confidentiality and integrity, the message contents must not be read or modified by unauthorized entities (see also section 5.5). TLS therefore must be able to distinguish between authorized and unauthorized entities, and must comply with the authorization and delegation fundamentals (see section 5.2.2). Only binary authorization (see section 2.4) is provided.

Snowden's disclosures in 2013 (see section 2.3) induced a reassessment of Internet security. One attempt to mitigate the threat to the privacy of users by pervasive

monitoring is to encrypt as much data as possible. The authentication of communication partners cannot always be achieved easily (cf. RFC 7435, pp. 2–3). Therefore, the so-called *opportunistic encryption* omits this step. In this case, TLS does not satisfy the **correlation fundamental**: while only the communication partners can read each other’s messages, users will not be sure that they are actually talking to the right entity. Attackers may exploit this omission to impersonate other entities. In this case, TLS will only provide limited integrity and confidentiality protection.

The following sections show how the authorization and delegation tasks are performed by TLS in a typical Web scenario where a user attempts to securely access a website on a server. The user’s browser acts as the TLS client, and the server that hosts the website is the TLS server. We do not provide a message by message analysis of TLS here since a detailed analysis of TLS security is not the focus of our work, but instead analyze how the necessary tasks are performed on each side.

## 7.1. Client-side Authorization

To perform task **A1**, the browser user as the overseeing principal must define the peer with which she wants the browser to communicate, the endpoint that is supposed to enforce her security objectives and the context for which the authorization applies. She defines which endpoint (peer) she wants her browser to contact by typing in its hostname. By using an HTTPS URI, she implicitly defines her browser to be the endpoint that enforces her security objectives. The site she wants to access represents the context of this authorization (see **context fundamental**).

In website spoofing attacks, the attacker attempts to mislead users into believing that they accessed the original website (Eckert 2013, p. 168). Phishing attacks aim at tricking users into disclosing sensitive data (Jagatic et al. 2007), e.g., by leading them to a fake website. These attacks can only be successful if the user as the overseeing principal is not careful with task **A1**.

By typing the hostname into the browser's address bar, the overseeing principal authorizes the server to receive information from and send data to the browser, thus implicitly performing task **A2**. The browser can validate the freshness and validity of the peer's permissions because the overseeing principal just typed the hostname into the address bar, thereby expressing her wish to access this resource. The browser assumes that whoever is able to type an address into its address bar is authorized to be the overseeing principal (task **A3**, **A4**). It can also assume that the overseeing principal wants it to enforce her authorization rules (**A5**). The browser needs to make sure that it requests the correct website on this server (task **A7**). Since the user of the browser is the only overseeing principal, the browser only needs to consider her most recent implicit permissions (task **A8**).

Task **A6** is a bit more difficult to solve. The browser must make sure that the website it attempts to access really is the one that the overseeing principal wants to reach. To do so, the browser must validate that the website actually has the hostname that the overseeing principal typed into the address bar (task **An1**). Since the browser cannot validate the website's hostname on its own, it must delegate this task.

### **7.1.1. Delegating the Attribute Validation**

To delegate task **A6**, the delegation tasks must be performed. In Web scenarios, usually TLS with PKIX is used for server authentication, i.e., to authenticate the server to the client (Holz et al. 2011, p. 428). In the TLS handshake, the website's X.509 certificate is transmitted to the browser. It contains the website's hostname (the claim statement) and its public key (the verifier as the reference to the holder). Certificates are the same for all delegators; the destination therefore does not need to be explicitly stated. Additionally, the certificates contain the dates that mark the start and end of the validity period (RFC 5280, p. 22). The certificate is signed by a Certificate Authority (CA), that thereby endorses that the holder of the key actually has this hostname (task **D1**). Before signing the certificate, the CA must validate that these assertions are currently true and thus perform task **An1**.

The certificate is transmitted to the browser in the TLS handshake. To perform task **D0**, the CA must be sure that the browser is authorized to get the certificate. As certificates are not confidential, everyone is allowed to receive them. The validity period in the certificate must be checked by the browser for task **D2**. Certificates are often expensive and difficult to distribute. They often expire only after several years. Endpoints and their keying material might be compromised in the meantime, which makes it difficult to comply with the **claim completeness fundamental**.

To mitigate the problem of compromised endpoints, certificate revocation lists can be used, where the issuing CAs can list revoked certificates (RFC 5280, p. 54). Another approach is the Online Certificate Status Protocol (OCSP), where clients can send certificate status queries to the issuing CA and thereby determine whether the certificate is still valid. Both proposals have the problem that the most recent revocation information must be available at the time of access. Otherwise the information will not be fresh. But if the request for revocation information takes too long, the user experience will suffer. According to Mozilla, OCSP requests “time out 15% of the time and take about 350ms even when they succeed” (Goodwin 2015). In cases where revocation information is not available, browsers usually omit the revocation check. TLS with PKIX therefore currently provides only a very low level of claim completeness. The quality of TLS can be improved by reducing the validity period of certificates. Efforts such as the project “let’s encrypt<sup>1</sup>” aim to improve the security of the Web by issuing free certificates that are only valid for 90 days. Also, browser vendors recently limited the acceptable validity period of certificates from 39 months to 825 days (CA/Browser Forum 2020, p. 42).

The browser validates the claim origin (task **D3**) by checking the signature in the certificate. The signature represents the claim issuer’s endorsement and binds the holder to its verifier. For task **D4**, the authorization of the CA to issue the certificate must be validated (see below).

X.509 certificates are the same for every delegator and are not meant for a specific entity. Therefore, the browser can be sure that the claim is meant for it (task **D5**).

---

<sup>1</sup><https://letsencrypt.org>

For task **D6**, the browser must validate that the certificate is issued to the server with the hostname that it attempts to contact. Since the certificate binds the server's hostname to the server's public key, the browser can use this key for the validation. TLS has various methods for ensuring that only the server that has the corresponding private key is able to generate the correct keying material for the communication with the client. E.g., in the RSA key exchange algorithm, the browser as the TLS client provides the server in the TLS Handshake with a premaster secret which is encrypted with the public key (RFC 5246, p. 57). Algorithms that use the key exchange mechanism Diffie-Hellman (DH, Diffie and Hellman 1976) require the server to have a suitable key pair that is either fixed (e.g., for ECDH\_ECDSA) or ephemeral, i.e., freshly generated (e.g., for ECDHE\_ECDSA). Fixed keys are provided in the server certificate. Ephemeral keys are signed by the server with the secret key corresponding to the public key in the certificate (RFC 4492, p. 6). The public key is then used in the DH key exchange. From the result of the key exchange, client and server calculate the premaster secret. The server can only determine the correct premaster secret if it knows the correct private key.

From the premaster secret, the master secret is generated. The keys that are required for the communication are derived from the master secret. The browser also generates the master secret and can then validate that it actually communicates with the holder of the certificate by checking the `finished` message. It consists of a hash value that was generated out of all handshake messages, is encrypted with a key that is derived from the master secret, and provided with a Message Authentication Code (MAC) (see also section 10.1.4). To protect the confidentiality and integrity of the subsequent communication, each message that is transmitted with the Record Protocol must use keys that are derived from the master secret for task **A6**. Also, the messages must be replay-protected. TLS uses sequence numbers for replay protection (RFC 5246, pp. 94–95).

TLS only supports the transmission of a single server certificate. Therefore, only claims of a single claim issuer are considered for the attribute validation of the server (task **D8**). Additional certificates are required to validate the CA's authorization to issue the certificate (task **D4**). To do so, the authorization tasks must be performed for the CA as we will discuss in the following section.

### 7.1.2. Validating the CA's Authorization

TLS allows the use of certificate chains. In the simplest case, the web server certificate is directly signed by a root CA; but it is also possible to use intermediate certificates as we will describe below. Browsers usually have a large number of pre-installed root certificates with which other certificates can be signed. The overseeing principal implicitly approves of the root CAs by installing and using the browser (tasks **A1** and **A2**; see also tasks **A3** and **A4** below). As all certificates, the browser's root certificates have a validity period that must be checked, and root certificates may also have long lifetimes. They cannot be revoked because they represent the issuing CA which is responsible for revoking them. Instead, browser updates must be performed to replace them. Such updates must be provided by the browser developers, and users must download and apply them. If the set of root certificates in force is updated frequently, the **rule completeness fundamental** is sufficiently satisfied (see also section 5.8).

To perform the tasks **A3** and **A4**, the browser must validate that the pre-installed certificates have been approved by its overseeing principal. But overseeing principals usually don't check the root CAs. Most users do not have the technical knowledge and the means to validate that the CAs are indeed trustworthy. To mitigate this problem, certificate authorities and browser vendors cooperating in the CA/Browser Forum<sup>2</sup> publish the baseline requirements certificate policy (CA/Browser Forum) that defines how the issuing and management of certificates must be performed. Only if CAs comply with these baseline requirements, their certificates are to be trusted by browsers. Main browser vendors such as Apple, Google, Microsoft and Mozilla are part of this alliance. Overseeing principals implicitly trust the browser vendors to allow only certificate authorities that protect their security objectives. This approach satisfies the **principal fundamental** only to a certain degree; CAs may not act in the overseeing principals' interest. This leads to a vulnerability that might render the whole authentication process ineffective.

CAs were repeatedly attacked successfully, e.g., Comodo (Comodo 2011), Diginotar (Nightingale 2011), the Agence nationale de la sécurité des systèmes d'in-

---

<sup>2</sup><https://cabforum.org>

formation (ANSSI) (Langley 2013) and the China Internet Network Information Center (CNNIC) (Langley 2015). In these cases, unauthorized certificates were issued; the responsible CA provided an attribute claim that was not specified correctly (task **D1**), and thus did not act in the interest of the overseeing principal, i.e., the browser user. It thereby violated the **claim issuer participation fundamental**. To provide a detection opportunity, Google developed a mechanism called *certificate transparency* (RFC 6962; Laurie 2014). For this approach, a public log of certificates is established that allows the detection of fraudulent certificates. If the logs contain only legitimate certificates and clients only accept certificates that appear in the log, the risk of fraudulent certificates can be mitigated. To provide certificate transparency, the TLS handshake must include a signed certificate timestamp from at least one log (RFC 6962, p. 13). Certificate Transparency may introduce new problems such as the leakage of subdomain names (Scheitle et al. 2018, p. 346).

As root certificates are pre-installed in the browser, the corresponding CAs are meant to be root CAs for this browser (task **A5**). The browser performs task **A6** by validating whether the signature on the web server or intermediate certificate was made with the CA's key as given in the root certificate. Certificates do not have contextual constraints (task **A7**). The CA is implicitly authorized to be a root CA because it is pre-installed in the browser. Therefore, only this implicit authorization must be considered (task **A8**).

To enforce the authorization rules (task **A9**), the browser must consider the information in the CA certificate concerning the CA's authorization. The CA must only be entrusted with the delegation task if its certificate is authorized to sign other certificates.

If the issuing CA's certificate is not a root certificate but a subordinate certificate, it is not pre-installed in the browser and must be provided in the TLS Handshake (RFC 5246, p. 48) (task **A2**). The browser checks if the certificate is still valid with the help of the given validity period and revocation mechanisms such as CRLs and OCSP (see also task **D2** in section 7.1.1).

CAs have special certificates that authorized them to sign other certificates. By signing the subordinate certificate, the superior CA binds the hostname to the

public key and endorses the subordinate CA's authorization to issue certificates (task **A1**).

The browser obtains the authorization rules for the subordinate CA with its certificate (task **A2**).

Authorization tasks A5 to A7 are performed as described for the root certificate (see above). The origin of the certificate and therefore the authorization rule (task **A3**) is again validated by checking the signature on the CA's certificate. If the superior CA is a root CA, the browser validates its authorization to sign the certificate and thereby provide the authorization rule for the subordinate CA (task **A4**) as described above. Otherwise yet another CA is required for validating the superior CA's authorization.

To achieve the overseeing principal's security objectives, the browser must only accept certificates from authorized CAs (task **A9**).

## **7.2. Server-Side Authorization**

X.509 certificates can also be used by the client side to provide for mutual authentication. The client certificates are then also transmitted in the TLS Handshake. However, the more common approach in the Web is to use only one-sided authentication in the handshake and perform the authenticated authorization of the client after the secure channel is established. The user is then usually prompted by the website to provide a user name and password. TLS only assists with performing task **A6**; for the other authorization tasks other means are required.

Usually, the user must register with the server before the conversation takes place. Often, the user's e-mail address is used for registration because they are unique attributes and can be used to determine the corresponding authorization data of this user in the system. Also, e-mail addresses can be validated without human interaction on the server side: The user provides her e-mail address in a registration form, and chooses a username and a password for future logins. The server then sends an e-mail with a confirmation link to this address. The user proves

that she is able to retrieve e-mails for this address by using the link and thereby confirms that the password is one of her verifiers. Attackers must not be able to read the password in the registration form or guess the confirmation link. The server has thereby validated that the e-mail address is one of the user's attributes (task **An1**) and that the password is a verifier for the user with this attribute.

After the registration, the user can contact the server using TLS. The TLS handshake provides the server with keying material for the client, but cannot provide the user's attributes. After the handshake is complete, the user types in her username and password. Since the information sent by the user is protected by the TLS record protocol, the server can validate that the attributes relate to the verifier (task **A6**). With the attributes, the server can check if authorization information is available for this user, and perform the other authorization tasks. When the user logged in successfully, the server may store cookies in the browser that can afterwards be used for authentication.

### 7.3. Secure Communication

After the handshake, the communication between client and server is protected by the TLS record protocol. As described in section 5.5, messages containing application data that is exchanged between endpoints constitute claims and must be protected accordingly, i.e., the delegation tasks must be performed on each side.

How the endpoints determine if the message is up to date (task **D2**), and if the claim issuer is authorized (task **D4**) depends on the application, as well as how endpoints evaluate multiple claims (task **D8**). TLS enables the endpoints to perform task **D6** for their peers.

The TLS record protocol may assist claim issuers with task **D1**. The application data in the message is the claim statement. To specify the intended destination of the claim, the claim issuer may encrypt the message with the receiver's keying material. The **claim origin fundamental** demands that the claim issuer endorses

the claim and all related information (see section 5.4.1). Encryption can therefore only assist with data destination verifiability if the encryption is included in the claim issuer's endorsement, i.e., it is applied before, or together with, the message authentication mechanism. The receiver of the claim then is also able to perform task **D5**. The application must use a different approach to achieve data destination verifiability if it is not provided by encryption, e.g., explicitly specify the intended destination in the message.

Other parts of the encryption may also be part of the claim, even though the relation is more subtle. Padding oracle attacks such as Lucky Thirteen (Al Fardan and Paterson 2013) or POODLE (Möller et al. 2014) exploit that attackers can modify the padding of an encrypted message, and derive information from the server's reaction. In this case, the server violates the **claim issuer participation fundamental** by accepting data (the modified padding) from an unauthorized entity, and again by afterwards providing a claim (a response or error message) that discloses information (in most cases via side-channel attacks such as timing attacks) to the unauthorized attacker. To perform task **D1** correctly and comply with the **claim origin fundamental**, the client must bind the padding to the message and endorse it. The server must ascertain that the padding stems from the same entity that sent the remainder of the message for task **D3**, i.e., the encryption must be part of the authenticated data of the message. The vulnerability could therefore be avoided if the data is encrypted before the message authentication is applied (Encrypt-then-MAC, see also Al Fardan and Paterson 2013, p. 527).

As an alternative, the side-channel attacks that allow the attacker to derive information from the server's reaction could be prevented, which would entail that the padding may stem from an unauthorized entity. In this case, client and server would also be able to perform tasks **D1** and **D3**.

TLS 1.3 no longer uses MAC-then-encrypt, but instead only Authenticated Encryption with Associated Data (AEAD) algorithms to mitigate the risk of side channel attacks (RFC 8446, p. 149). AEAD enables authentication for the encrypted data as well as for additional unencrypted data (Rogaway 2002, p. 98). Jonsson published a proof of the authenticity provided by the AEAD algorithm *Counter with CBC-MAC* (CCM) (Jonsson 2003).

## 7.4. TLS Example Conclusion

As we have seen, the model is suitable to demonstrate the various different roles of client, server and the certificate authorities. It also helps to understand the respective responsibilities of the various actors concerning the protection of the overseeing principals' security objectives. Also, the model highlights certain problems of security protocols, e.g., that claims need to reflect the current opinion of their issuer (**claim completeness fundamental**) and that the issuer must participate in the protection of the overseeing principals' data (**claim issuer participation fundamental**). If the fundamentals are already considered when the protocol is first specified, problems may be avoided or at least mitigated. Countermeasures for certain problems may already be implemented from the start. E.g., if browser vendors had limited the acceptable validity period of certificates more strictly from the beginning, TLS would have been better suited to address the **claim completeness fundamental**.

Complying with the **claim origin fundamental** and the **claim issuer participation fundamental** renders padding oracle attacks such as Lucky Thirteen or POODLE ineffective, that exploit that attackers can modify the padding of a message. This shows that the fundamentals can assist with the design of security solutions.

TLS can be used to assist with task **A6**. If certificates are used, task **An1** that is used for task **A6** is delegated to a CA, and the delegation tasks must be performed by the delegator. TLS helps with the delegation tasks, but does not specify how task **An1** is performed. For task **D4** the authorization of the CA must be validated. This task can only be performed if the certificate chain ends in a root CA that is installed in the browser. Our analysis emphasizes the weakness in the web's trust model: ascertaining the trustworthiness of every root CA is challenging even for technically versed users. The communication for which the certificates are used can only be secure if the CAs perform the necessary tasks, in particular the tasks **An1** and **D1**.

In our example, the client does not have a client certificate. On the server side, TLS can therefore only assist with performing task **A6**, but does not help with task **An1** or other authorization and delegation tasks.

In typical Web scenarios, the server acts autonomously and is often maintained by technically versed personnel. The client is directly controlled by the user who makes the authorization decisions for it. The example shows that the clients rely on the direct interaction with the user as the overseeing principal when performing the authorization tasks **A1** to **A5**.

## 8. Conclusion

In this part, we introduced our authenticated authorization evaluation model. ANAZEM specifies the authorization and task delegation fundamentals that must be satisfied for an effective authenticated authorization process. Our model shows how authorization, authentication and task delegation must interrelate; endpoints must authenticate peers and claim issuers to determine their authorization. Claim issuers that assist with authentication and authorization must be authorized by the overseeing principals. If a task is delegated, the delegation tasks must be performed.

ANAZEM describes how the whole authenticated authorization process must be protected to be effective. The fundamentals comprise the protection of the data destination verifiability, a security objective that was previously missing from the literature and therefore was easily disregarded. By requiring data destination verifiability, ANAZEM facilitates the detection of protocol flaws that may lead to man-in-the-middle attacks. We showed that all tasks must be performed for an effective overall solution; omitting a task leads to vulnerabilities that may be exploited by attackers. The fundamentals and tasks enable solution designers to identify problems and weaknesses in authentication and authorization mechanisms. Some security problems in protocols were only discovered after several years, e.g., the missing freshness in the Needham-Schroeder protocol, the problem of unauthorized certificates or the unauthenticated padding of encrypted messages that enable padding oracle attacks such as Lucky Thirteen and POODLE. Our model can assist in revealing such vulnerabilities immediately.

A protocol does not need to address each task itself, as the example of TLS in the web shows. ANAZEM allows protocol designers to describe in detail which tasks a protocol performs and how it satisfies the fundamentals. That makes it

---

easier for developers to understand how the protocol works and which services it provides. The requirements on additional protocols can more clearly be defined which allows to more easily determine which building blocks can be combined to achieve a secure, gapless solution.

The comparison with the BAN logic showed that ANAZEM is more fine-grained and thus able to detect more problems. With TLS in the Web, we gave an example how a well-known protocol performs authenticated authorization tasks today.

An endpoint cannot protect the security objectives of its principal if it does not perform the authorization tasks. However, the tasks are not always easy to implement. Some of the authorization and delegation fundamentals such as the **rule completeness fundamental** can in practice only be satisfied to a certain degree. Therefore, security solutions cannot provide 100 % security as was shown with the examples of the compromised certificates in PKIX and the validity problems with timestamps and lifetimes in the Kerberos protocol. The quality of a security solution, i.e., the level of security that a solution can provide, depends on the degree to which the authorization and delegation fundamentals are satisfied.

ANAZEM provides us with a better understanding of authorization and the secure distribution of claims. The fundamentals of our model enable us to determine the cause of a certain vulnerability. Also, the provided terminology facilitates discussions about vulnerabilities as well as about abilities and requirements of protocols.



## **Part III.**

# **Architecture Model and Protocols**

## 9. Architecture

In part I, we analyzed the special characteristics of smart objects and the environments they are used in. From this analysis, requirements for an authorization solution for constrained environments were derived. In part II, we developed the Authenticated Authorization Evaluation Model (ANAZEM) and identified the authorization fundamentals that must be satisfied for a secure and effective authorization process. Because of the constrainedness of the devices, the identified requirements and fundamentals are difficult to achieve. To develop a suitable authorization solution for the IoT, we need an architecture model that is optimized for the special characteristics of smart objects.

The OAuth 2.0 framework was designed to grant third-party applications limited access to Web content (see also section 2.5). It focuses on authorization on the server side and does not consider scenarios where multiple overseeing principals are involved. For the big Internet, User Managed Access (UMA) was developed to allow the overseeing principal of a server to grant access to the client of another overseeing principal, who is called the requesting party (Machulak and Richer 2018). Like OAuth, UMA uses a single centralized authorization server.

The use cases show that the overseeing principals participating in an IoT use case may have different security objectives and distinct, sometimes even conflicting interests (see section 4.1). Our architecture aims at considering the interests of all participating overseeing principals, and provides security to both the client and the server. Instead of static configurations that could be provided during manufacturing or deployment, our architecture allows for dynamic authorization solutions that offer more flexibility (draft-ietf-ace-actors-07, p. 3) and more security. Since it enables secure constrained-to-constrained communication it provides for a true, secure Web of Things.

In the big Internet, the REST architectural style made the Web successful (see also section 3.4) and it can be expected to also benefit constrained devices in the IoT (Bormann et al. 2012). Accordingly, the **REST Req** demands that an authorization solution for the IoT should conform to a RESTful architectural style. In this chapter we will develop an extension to the REST architectural style to offer support to constrained devices (section 9.2). In section 9.3, we introduce an architecture for authenticated authorization in constrained environments. The architecture is based on the architecture model we developed for the IETF ACE working group (draft-ietf-ace-actors-07), and that we presented in earlier work (Gerdes et al. 2015c). In section 9.4 we present an evaluation of the architecture, and analyze if the requirements from the background part are met. The findings of this chapter are summarized in section 9.5.

## 9.1. Terminology Translations

The ACE working group uses OAuth/UMA terminology. However, this terminology differs from the terms used in REST and CoAP. Also, since OAuth 2.0 was designed for certain applications in the big Web, it does not consider all necessary actors. It was designed to protect the resource server, and not the client. As stated above, both OAuth 2.0 and UMA use a single centralized authorization server.

Additionally, some OAuth 2.0 components have a different purpose; Most notably, the OAuth 2.0 authorization server is designed to issue authorization tokens and does not assist with other authorization tasks or authentication. Also, some terms have a different meaning; an OAuth 2.0 endpoint is an HTTP resource (RFC 6749, p. 18) while in CoAP entities that participate in the protocol are called endpoints (RFC 7252, p. 6). To avoid confusion and provide consistency, we continue to use REST and CoAP terminology and extend it where necessary. For better understanding, we provide a translation of the most important terms below (see table 9.1).

Our Terminology	OAuth 2.0 / UMA	Combined Terminology (Actors Draft)
Client	Client	Client
Server	Resource Server	Resource Server
Authorization Manager (AM)	/	Authorization Manager (AM)
Client Authorization Manager (CAM)	/	Client Authorization Server (CAS)
Server Authorization Manager (SAM)	Authorization Server (AS)	Authorization Server (AS)
Overseeing Principal	/	Principal
Client Overseeing Principal (COP)	Requesting Party (RqP)	Requesting Party (RqP)
Server Overseeing Principal (SOP)	Resource Owner (RO)	Resource Owner (RO)
Endpoint	/	Endpoint
Resource	Endpoint	Resource

Table 9.1.: Terminology Translations

## 9.2. Architectural Style

A *software architecture* defines the configuration of *architectural elements* and aims at achieving architectural properties that are suitable to fulfill the system requirements (Fielding 2000, pp. 7). The roles of architectural elements and the relationships between them are defined in the *architectural style*. Architectures are constrained by the definitions of the architectural style they conform to. Architectural elements encompass data, *components* that process the data and *connectors* that mediate the data between components. The term components may in some cases be confusing since it is an ambiguous term. Also, our architecture contains the principals which are logical functional entities on a more conceptual level. We will therefore use the term *actor* for the logical functional entities in our architecture and *architecture component* where a reference to REST concepts is made. Actors do not need to map devices one to one: several actors may share a device or even a piece of software (Gerdes et al. 2015c, p. 238).

REST is a composite architectural style derived from styles such as the client-server style that introduces the roles client and server, the stateless server style defining the stateless constraint, the cache style for caching information and the uniform interface style that demands the definition of interfaces instead of architecture components (see also section 3.4).

Because of their hardware and energy limitations, constrained devices will have more difficulties to perform tasks on their own (see also chapter 3). They can

benefit from delegating difficult tasks to other, less-constrained devices. To make the REST architectural style more suitable for authorization in constrained environments, we introduce the task delegation style. It comprises two actors, the delegator and the manager: the manager performs the tasks that were delegated to it by the delegator. The manager provides a service to the delegator but does not necessarily have the role of a REST server. Figure 9.1 depicts the architectural style.

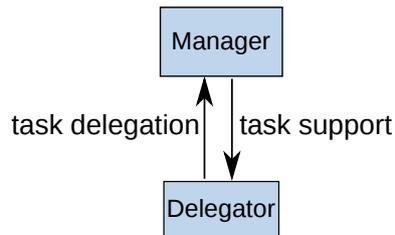


Figure 9.1.: The Task Delegation Architectural Style

The task delegation style can be used for the delegation of authentication and authorization tasks (see section 5.4) in scenarios where a PEP (see section 5.2.3) needs support to enforce the security policies of its overseeing principal. By coupling the PEP with an Authorization Manager (AM), the enforcement of authorization policies can be separated from the configuration and management of authorization rules and from making authorization decisions.

The AM performs authorization and authentication tasks for its delegators. It prepares and endorses authentication and authorization data and provides it to the delegator. The delegator uses this information to enforce the overseeing principals' authorization policies. AMs must have the overseeing principals' approval to be claim issuers for delegators. The delegation fundamentals (see section 5.4.1) must be satisfied for a secure authorization task delegation. Figure 9.2 shows the authorization task delegation.

To meet the **Distinct Interests Req**, the security policies of all participating overseeing principals must be considered. An endpoint can only participate in the protection of data if it can satisfy the authorization fundamentals. If an endpoint is not able to perform the authorization tasks, it must delegate them. As an additional constraint for the authorization delegation architectural style, PEPs that

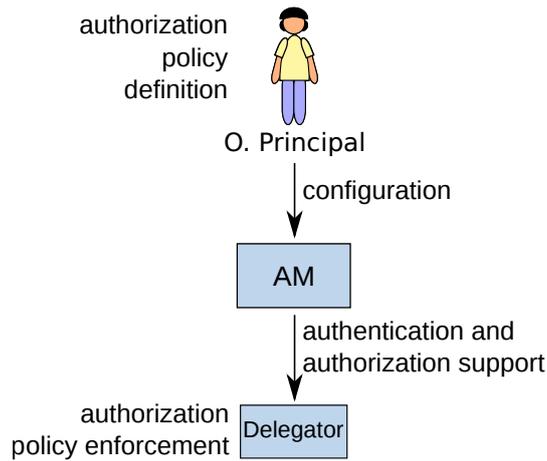


Figure 9.2.: Task Delegation for Authorization

are not able to perform the necessary authorization tasks and satisfy the authorization fundamentals on their own are required to have a security association with an AM and must be able to perform the delegation tasks for it.

### 9.3. Architecture Model

The scenario we want to address with the authenticated authorization architecture is the following: A client (C) and a server (S) want to communicate securely. C wants to request access to a resource (R) that is hosted by S. C and S did not necessarily know each other prior to the communication and have no security association. Since the authorization solution must support secure communication between constrained devices (**C2C Req**) and between constrained and less-constrained devices (**C2L Req**), the client, the server or both may be constrained devices. The basic scenario is depicted in figure 9.3.

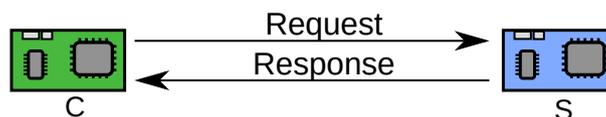


Figure 9.3.: Basic Scenario

Servers and clients must be able to determine if their overseeing principal authorized their communication partner. Since client and server may be acting autonomously (**Autonomous Server Req** and **Autonomous Client Req**), mutual authenticated authorization is required (**Mutual Authorization Req**, see also section 4.2.8): S must be able to validate that its overseeing principal allows C to access R as requested (objectives of type 1). C must validate that its overseeing principal authorizes S to be a server for R (objectives of type 2). For objectives of type 1, authorization on the server side is required. Objectives of type 2 require client side authorization.

### 9.3.1. Actors

From the scenario described above, we can derive the actors in the architecture. A logical functional entity that can be assigned to a constrained device is part of the *constrained level*. Although a constrained level actor may be implemented with a less-constrained device, a constrained device suffices. An actor that requires more capabilities is located on the *less-constrained level* (draft-ietf-ace-actors-07, p. 4). Overseeing principals, i.e. individual human beings or companies, belong to the *overseeing principal level*.

On the constrained level, C and S want to communicate securely. They are controlled by their overseeing principals on the principal level. C and S not necessarily have the same overseeing principal: we distinguish between the Server Overseeing Principal (SOP) and the Client Overseeing Principal (COP) (see figure 9.4).

The architectural style requires that each endpoint that is not able to perform the authorization tasks on its own has an authorization manager (see section 9.2). The authorization managers on the less-constrained level assists the constrained devices with authentication and authorization tasks. They provide them with control information such as authorization information, attributes of clients and servers and the associated verifiers. Authorization managers are able to influence the authorization decision and must act in the overseeing principal's interest. To enable the principals to keep the control over their endpoints (**Separate Authority**

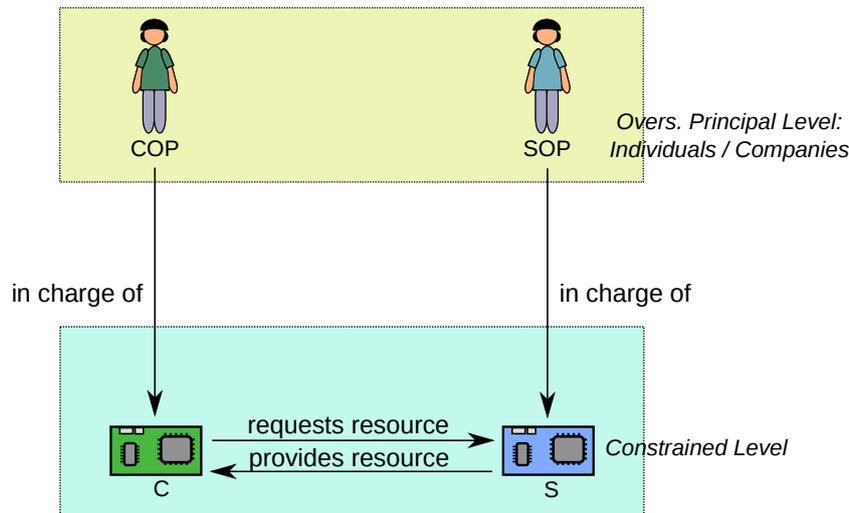


Figure 9.4.: Endpoints and Their Principals

**Req**), the AM is overseen by the endpoint's overseeing principal: COP oversees C and C's Client Authorization Manager (CAM), SOP oversees S and S's Server Authorization Manager (SAM) (see figure 9.5). Thus, the interests of all principals can be considered (**Distinct Interests Req**).

On the less-constrained level, endpoints have the ability to use unmodified protocols of the big Internet. They can, e.g., use HTTP and TLS. On the constrained level, endpoints may be too constrained to deploy a full protocol stack with common Internet protocols. They will, e.g., speak CoAP and DTLS. A communication where one device is a constrained device is a constrained-level communication: the limited abilities of the constrained device must be considered.

The communication between clients and servers as well as between constrained nodes and their authorization managers may be transported over the Internet and thus require protection (see section 2.3).

### 9.3.2. Architecture Variants

The constrained device is the most limiting factor in a communication. If a task can be performed by a constrained device, a less-constrained device is also able to perform it. If an architecture provides for secure communication between two

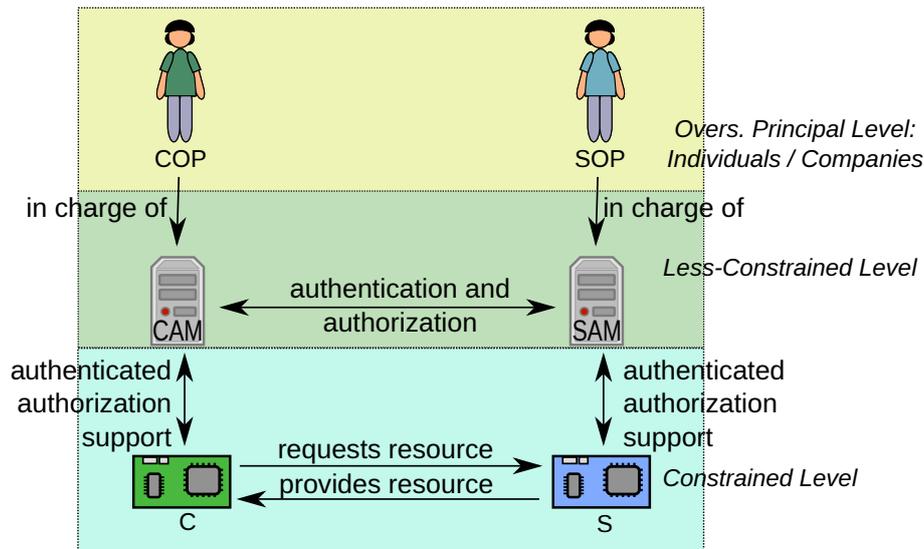


Figure 9.5.: The complete architecture

constrained devices, it will also enable less-constrained devices to communicate securely. The most general architecture therefore is the four-corner architecture depicted in 9.5, where each constrained device has its own authorization manager.

The architectural style allows for variants in the architecture. Constrained Level actors may have the same overseeing principal. In this case, they are in the same security domain and may share an AM (see figure 9.6).

Constrained devices need to be able to communicate with less-constrained devices to meet the **C2L Req**. Less constrained clients and servers can perform authorization tasks on their own and do not require the assistance of an AM. We can think of them as having an integrated AM. As less-constrained devices, they are located on the less-constrained level (see figure 9.7).

### 9.3.3. Intermediaries

REST introduces intermediaries which can be used as caches, resolvers or tunnels (see also section 3.4). In some cases, intermediaries may be required to modify the data of a transmission, e.g., if a translating intermediary is used to translate

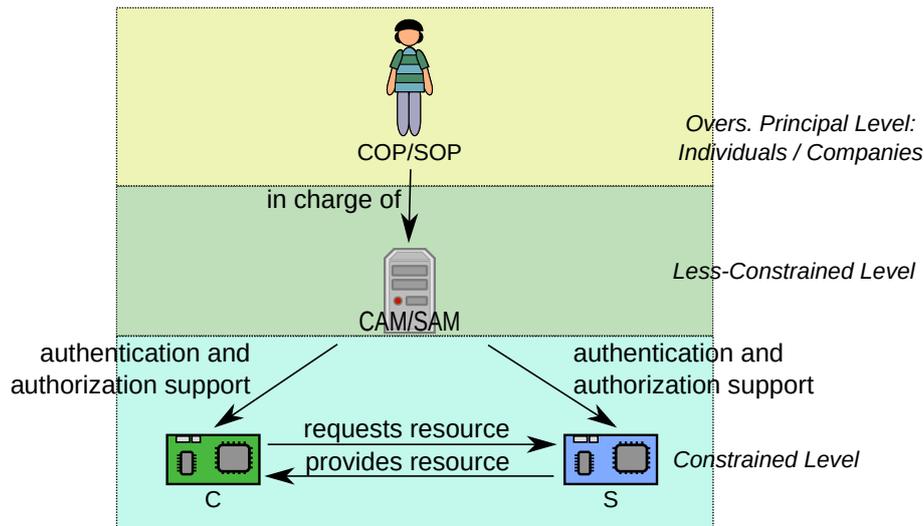


Figure 9.6.: Combined Authorization Manager

between DTLS and TLS or CoAP and HTTP. Caching intermediaries may store a resource representations as long as it is fresh. To do so, they must know at which time a piece of data must be considered stale and needs to be thrown away. Also, caching intermediaries must be able to decide if a request targets a certain cached resource representation. To do so, they may need access to at least parts of the message.

The secure transmission of data over caching or the translating intermediaries described above has some difficulties. We can roughly distinguish cases where client and server use the security association between each other for the secure transmission, cases where they instead establish a security association with the intermediary and hybrid cases.

Cases where client and server have a security association with each other but not with the intermediary are especially difficult. Untrusted intermediaries must only have access to data that does not need confidentiality or integrity protection. Caching intermediaries may store protected content if they are able to identify the content, and as long as it does not become stale. If the availability of the data is required, the reliable delivery of its representation must be assured. Translating proxies must be able to read and modify certain parts of a message. For untrusted intermediaries this is only possible if the data is not confidential and does

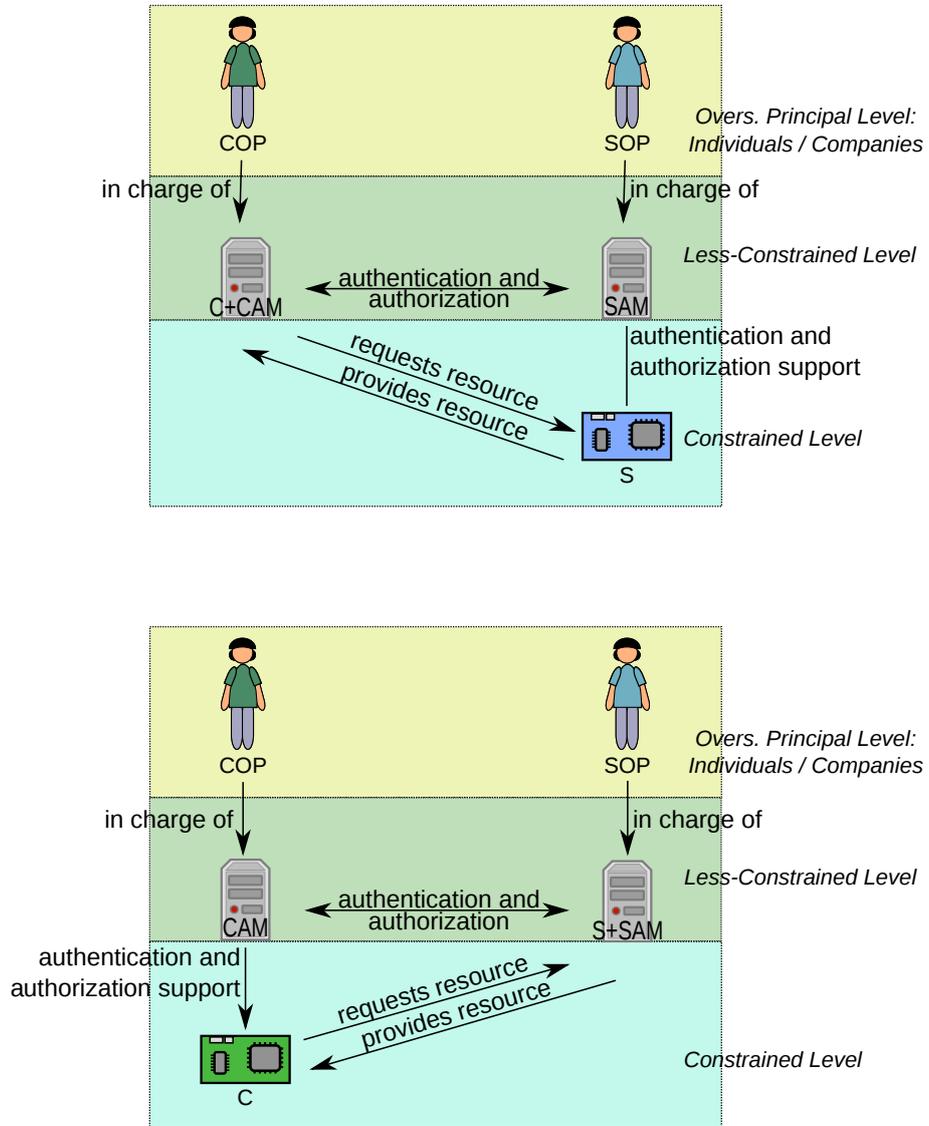


Figure 9.7.: Less Constrained Clients and Servers

not require authenticity and integrity protection. If untrusted proxies are used, protocol designers and implementers must be extremely careful to not violate the authorization and delegation fundamentals; the **reference monitor fundamental** requires that an endpoint validates the authorization of its communication partner before sending and after receiving a message. The proxy must not modify information that is relevant for the authenticated authorization.

If the security mechanism works on a lower layer, the upper layer content is usually completely protected and it is not possible for the intermediary to read or modify parts of the message. In these cases, only trusted caching and translating intermediaries are supported.

If a security association is formed with the intermediary, the data can be securely transmitted between server and intermediary and between intermediary and client. The intermediary must provide the necessary protection for the data: it needs to determine the origin of the data if integrity and authenticity are required, and must decrypt and encrypt confidential data. Decrypted data is vulnerable until it is encrypted again; there is no end-to-end security between client and server in this case.

In hybrid cases, intermediaries may be trusted only to a certain degree; they may not have the same authorization as the client and the server. Client and server both need to determine whether the intermediary is authorized to perform a certain action, i.e., read or modify (parts of) the message. If the authorization of the intermediary is not confirmed, the integrity and confidentiality of the transmitted data are at risk.

## 9.4. Evaluation of Architectural Properties

The architecture is based on REST and therefore shares a lot of its characteristics and features. In section 9.4.1 we will discuss how the introduction of the authorization managers changes the design and thereby influences REST properties.

Afterwards, we will analyze if the architecture is able to meet the requirements we defined in the background part: section 9.4.2 summarizes the evaluation of

hardware requirements defined in chapter 3; in section 9.4.3, 9.4.4 and 9.4.5 we discuss if the architecture meets the requirements for authorization in constrained environments that we introduced in chapter 4.

Although the architecture does not impede conformance with the lifecycle requirements, it cannot directly participate in meeting them. This type of requirements is therefore omitted.

### 9.4.1. REST

The architecture design aims at satisfying the **REST Req** and to thus profit from REST features and characteristics. We will now analyze how the use of authorization managers changes the design and influences properties such as the scalability, the separate evolution of architecture components and the ability to deal with intermediaries.

A design principle for the Web are loosely coupled architecture components (see also section 3.4). The interfaces between the components need to be defined and stay the same but the architecture components themselves are not defined. Since this feature allows the components to evolve individually, it facilitates fast progress because the Web is developed to work over organization boundaries. Separate evolution is hindered if the architecture components need knowledge for the communication that is not obtained in the communication. In actual applications interface definitions usually change over time. Architecture components need to at least use the same protocols to be able to communicate.

If a less-constrained device performs authentication- and authorization-related tasks for a constrained device, the security of the constrained devices depends on the security association to its less-constrained device. The constrained device must store authentication and authorization information about its manager. The authorization manager must be able to perform tasks for the constrained device in accordance with the overseeing principals' security policies and thus needs to store additional data. These devices are therefore more closely coupled.

The constrained devices and their authorization managers belong to the same security domain and will usually be administrated by the same individual or

company. The closer coupling between these actors is therefore less harmful. Nevertheless, protocol designers must consider that the endpoints, especially the constrained devices, must only store as much information about their managers as necessary, and gain as much of the required information from exchanged messages as possible. Fortunately, explicitness and self-descriptive messages also help to improve the security (see chapter 5).

That both the constrained device and the AM need to store authentication and authorization information about each other also affects the stateless constraint. Without the authorization manager, a constrained device might not be able to securely communicate at all since it might not be able to establish a security association with its communication partner; or it needs to be pre-provisioned with the authentication and authorization information for each possible client which would violate the stateless constraint even more. Scalability can be improved by simplifying the architecture components and by decentralizing interactions (Fielding 2000, p. 32). By delegating the performance of the more difficult security tasks, client and server components are simplified, and the scalability is thereby improved. That the delegators, i.e, the constrained devices, store information about their authorization managers therefore is a reasonable trade-off.

As described above, security mechanisms do not work well with intermediaries. In most cases, clients and servers must trust intermediaries at least to some degree. This affects the scalability since clients and servers cannot use random intermediaries and must have or obtain knowledge about those which are trusted. Realizing caching and translating intermediaries in secure communications is a general security problem and not specific for our architecture.

The architecture supports REST features where possible, and offers the chance for a reasonable solution where trade-offs between REST functionality and the security for the communication of constrained devices are necessary.

#### **9.4.2. Hardware Requirement Evaluation**

The architecture is specifically designed to support constrained devices and therefore meets the **Class 1 Req** requirement. The design may also facilitate saving

energy (**Energy Req**), since difficult tasks can be delegated to less-constrained devices. Nevertheless, the protocols used to implement the architecture must reduce traffic and avoid producing unnecessary overhead because the sending and receiving of messages is particularly expensive.

The introduction of authorization managers allows to simplify client and server components since difficult tasks can be delegated to a trusted entity. The architecture therefore enhances scalability where security is required (**Scalability Req**).

The architecture allows for direct secure communication between constrained clients and servers. CoAP and Observe are therefore supported by the architecture (**CoAP Req** and **Observe Req**).

### 9.4.3. Environment Requirements

Authorization managers act as a mediator between constrained devices and their overseeing principals. As the managers are less-constrained, overseeing principals can interact more easily with them, and the AMs are more capable to securely obtain and store authentication and authorization information. The architecture therefore helps to meet the **Absent Principal Req**.

Constrained devices need their authorization managers to communicate securely. The **Offline Req** demands that constrained nodes are able to communicate even if they are not able to reach the Internet. This requirement can be met if local authorization managers are used, e.g., the overseeing principals' smartphone or a router.

For the **Multiple Hops Req**, the communication may need to be securely transported over multiple hops. Intermediaries are not part of the architecture, especially if they only forward packages. Architectures that implement the task delegation architectural style do not prohibit the use of intermediaries.

The four-corner architecture supports both constrained to constrained and constrained to less-constrained communication (**C2L Req** and **C2C Req**) as shown in section 9.3.2.

The task delegation architectural style can be applied to group communication scenarios. In this case, each device can have its own manager, or multiple devices can share an AM (**Group Communication Req**). The architecture cannot contribute to providing protection against capture (**Capture Req**), but it enables constrained devices to react more quickly if their peers were compromised.

#### 9.4.4. Authorization Rule Requirements Evaluation

Since each constrained device has its own authorization manager that assists it in performing difficult authentication and authorization tasks, the overseeing principals keep the control over their endpoints even if both client and server are constrained (**Separate Authority Req**). With the possibility to deploy an authorization manager for each constrained device, the security policies of all participating overseeing principals can be considered (**Distinct Interests Req**). If a constrained device does not have an own authorization manager, the architecture does not contribute to protecting the interests of the device's OVP.

The authorization managers enable the constrained devices to enforce fine-grained authorization rules (**Fine-Grained Authorization Req**). Without this assistance, the constrained devices might not be able to authenticate and validate the authorization of other devices at all.

The architecture cannot directly influence the **Simple Configuration Req** that demands that the configuration of authorization rules must be simple. But that the authorization servers act as mediators between overseeing principals and constrained devices helps to achieve this goal. As less-constrained devices, they likely are less energy-constrained, and therefore do not need to sleep so much. They may have user interfaces and displays, and, because they have more processing power and better networking capabilities, will likely have better response times. The architecture can therefore facilitate the simple configuration of authorization rules. Also, it enables the principals to preconfigure authorization rules (**Preconfiguration Req**).

Authorization managers may in some cases be able to assist with interpreting context-based authorization rules as demanded by **Context-Based Authorization**

**Req.** Also, since the constrained nodes are relieved from the more difficult authorization tasks, they have more capacities to process context-based authorization rules themselves.

If overseeing principals delegate their permissions to others as demanded by the **Permission Delegation Req**, endpoints may be required to evaluate more complex authorization rules. The AM can assist the constrained device in this task.

As described above, the architecture supports cases where both the client and the server are constrained and need to communicate autonomously (**Autonomous Server Req** and **Autonomous Client Req**). By introducing an AM for each delegator, the architecture enables devices, that may even belong to different owners, to authenticate each other and validate each other's authorization (**Mutual Authorization Req**).

#### 9.4.5. Authorization Rule Update Requirements Evaluation

Authorization Managers facilitate authorization updates on constrained devices because they act as a mediator between the overseeing principal and the constrained device. The AM assists constrained devices in authenticating the origin of a message and validate its authorization. Thereby, the AM enables dynamic updates (**Dynamic Updates Req**). Constrained devices need to be able to access their AM to receive authorization updates (see also **Offline Req** evaluation in section 9.4.3).

For the same reasons, the use of less-constrained authorization managers can make the revocation of permissions easier (**Revocation Req**). Authorization managers can act as a contact for their constrained devices, and are aware for which of their devices revocations are relevant. If a constrained device cannot reach its AM, revocation messages may not reach it.

Authorization Managers can assist their constrained devices by handling temporary permissions for them and provide them with simplified authorization rules (**Temporary Permissions Req**).

## 9.5. Conclusion

In this chapter, we enhanced the REST architectural style by the introduction of less-constrained devices that relieve the constrained devices from performing more difficult tasks. The task delegation architectural style can be used for authenticated authorization by assigning an authorization manager to each constrained device that handles sensitive data and is not able to perform authentication and authorization tasks on its own.

We introduced our architecture for authenticated authorization in constrained environments where client and server may have distinct authorization managers. We showed that the requirements defined in the background part of this work are met by the architecture model where possible. The architecture follows REST principles, as far as this is feasible while providing for secure communication, and thus provides scalability and allows for separate evolution. A protocol based on the architecture enables constrained devices to perform mutual authenticated authorization with other constrained devices or with less-constrained devices.

Authorization solutions must implement the task delegation architectural style to enable secure authentication and authorization. We showed that the four-corner architecture is the most general architecture since it allows for constrained to constrained and machine-to-machine communication even if the devices have distinct overseeing principals. Architectures with only a single authorization manager offer less support for constrained and autonomous devices and can only address a subset of scenarios. Our architectural style and the four-corner authorization architecture are the basis for a true Web of Things.

## 10. Solution Design Considerations

Authenticated authorization solutions must satisfy the authorization fundamentals that we introduced in chapter 5. The fundamentals may be satisfied in various ways, and an authenticated authorization solution must determine which approach is best suited for the application scenario it is developed for.

Authenticated authorization solutions require the secure transport of authentication and authorization information. Transport Security solutions such as TLS and DTLS can be used with various types of algorithms. We will analyze in section 10.1 how certain types of cryptographic algorithms assist in satisfying the fundamentals.

The **rule completeness fundamental** and the **claim completeness fundamental** require the validation that a certain piece of information still reflects the opinion of its issuer. We will discuss the validity of authorization information and claims in section 10.2.

To satisfy the **reference monitor fundamental** and the **correlation fundamental**, the endpoint must assure that a message actually stems from a certain peer and is not a reinserted copy. This problem can be addressed by solutions that assure the freshness of a piece of information: they provide the receiver with means to detect whether a message has already been received previously (Burrows et al. 1990, p. 20). We will analyze approaches for solving this problem in section 10.3.

### 10.1. Cryptographic Algorithms

The strength of a cryptographic algorithm depends on the algorithm itself and the key length used (NIST SP 800-57, p. 52). The stronger the algorithm, the

more effort is required to break it. A cryptographic algorithm is practically secure if the effort and cost for breaking it exceed the resources of potential attackers and the anticipated gains (Eckert 2013, p. 321). Since computing power increases over time, the effort and cost of performing cryptanalysis on a system decreases. Also, shortcuts may become known that reduce the effort required for an attack, e.g., the number of keys that the attacker has to try. Cryptographic systems that are sufficiently secure today may be broken in the future, and then need to be replaced. The authorization fundamentals cannot be satisfied in a system if weak cryptographic mechanisms are used. IoT devices may have a very long lifetime (see also section 3.2), which exacerbates the problem.

To participate in the protection of their overseeing principals' security objectives, the communication partners must know if the entity that is able to use certain keying material is authorized to send or receive certain data (**reference monitor fundamental**, see section 5.2.2). For a secure authenticated authorization process, there must be a continuous link between the communication partner and the holder that the permission relates to (**correlation fundamental**). TLS and DTLS assist endpoints in performing tasks **A6** and **D6**: by finishing the handshake, an entity can prove that it is able to use certain keying material (see also chapter 7). An endpoint that obtained authorization information that is bound to certain keying material can thereby determine the peer's permissions. The permission may be directly bound to the keying material in an authorization claim, or it may only contain certain attributes of the holder. To establish a relation to the keying material in the latter case, an attribute claim is additionally required (see also section 5.4). Figure 10.1 shows how the relation between the authorization rules and the peer is made using claims and verifiers.

In the following, we will analyze how cryptographic algorithms assist in satisfying the fundamentals. Section 10.1.1 and 10.1.2 describe symmetric and asymmetric algorithms, respectively. In section 10.1.3 we show how forward secrecy assists in satisfying the **correlation fundamental**. Transport and object security solutions may also provide data destination verifiability and thus assist in satisfying the **destination fundamental** and the **claim origin fundamental** as we will discuss in section 10.1.4. Section 10.1.5 gives an overview of object security solutions. We discuss our findings in section 10.1.6.

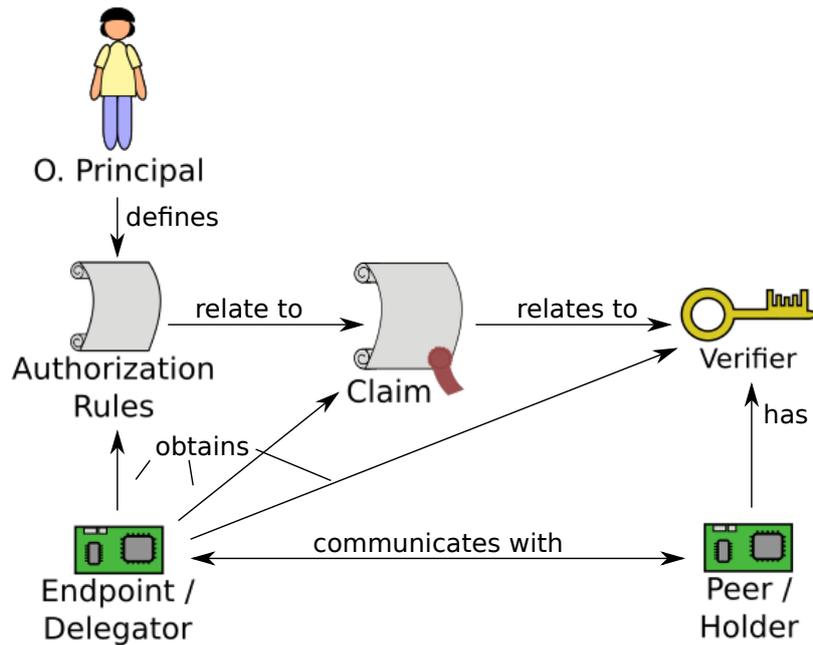


Figure 10.1.: Relation between Authorization Rules and the Peer Using Claims

### 10.1.1. Pre-shared Keys

To satisfy the **holder relation fundamental**, claim issuers that state a claim about a certain holder must provide a verifier that binds the holder to the claim (see section 5.4.2.1). The verifier must exclusively identify entities for which the claim statement is true. Symmetric approaches use the same key for encryption and decryption, i.e., all parties that participate in the same communication must share a key. The confidentiality of the keying material must be protected. If a claim is bound to a symmetric key, the key must usually be known to the holder, the delegator and the claim issuer. The keying material alone therefore cannot exclusively identify an entity, which has an impact on the authorization process. The verifier alone is therefore not sufficient to ascertain that a message actually stems from the holder; the delegator additionally must ascertain that it did not send the message itself, and it must also trust the claim issuer not to impersonate the holder.

As the claim issuer knows the keying material, it may read data exchanged between endpoints that is encrypted with the keying material. It may therefore be

able to perform a passive attack, whereas with an asymmetric solutions it would have to actively impersonate an entity. As in every case where a claim issuer influences the authorization, overseeing principals must authorize the claim issuer (**claim issuer authorization fundamental**).

For achieving data destination verifiability with symmetric algorithms encryption alone is not sufficient; the receiver of a message must also ascertain that it did not send the message itself (see also section 10.1.4).

In an authorization claim, the permission is directly bound to the verifier. Every entity that can use this verifier has this permission. Symmetric keys can therefore only be used for group communication if all members of the group have the same permissions.

In contrast to asymmetric approaches, symmetric algorithms are believed to be resistant against quantum computing if longer key lengths are used (NIST IR 8105, p. 3). Quantum attacks are unlikely to be feasible in the near future, but constrained IoT devices may be operated for a very long time.

### 10.1.2. Asymmetric Keys

Asymmetric cryptography makes use of *one-way functions* that are easy to compute, but hard to reverse, as the necessary computations require too much effort (Schneier 1996, p. 29). To make one-way functions useful for cryptography, they require a *trapdoor*: reversing the function is hard unless a certain secret is known. Performing asymmetric cryptography is more challenging for a system than symmetric cryptography (Ferguson et al. 2010, p. 28), because the operations used in asymmetric algorithms require much more effort to compute. Symmetric algorithms are therefore usually preferred if large amounts of data need to be encrypted. For asymmetric TLS and DTLS approaches, hybrid solutions are deployed, i.e., regardless of the keying material used in the handshake, the record protocol uses symmetric keys to protect the communication.

There is no mathematical proof that one-way functions really exist; researchers might discover an easier way for reversing a one-way function and thus break

the cryptographic algorithm or make it less secure (Schneier 1996, p. 29). Shor showed that the problem of finding discrete logarithms and factoring integers can be solved in polynomial time on a quantum computer (Shor 1999, p. 306). Asymmetric approaches such as RSA and key-agreement schemes such as Diffie-Hellman may need to be replaced if quantum attacks become practical (NIST SP 800-57, pp. 52–53).

In asymmetric solutions, the private key is only known to its holder. The keying material can thus exclusively identify a single entity and therefore is useful as a verifier. The keying material can be used to satisfy the **correlation fundamental**, and to thus protect the confidentiality, authenticity and integrity of messages. Data destination verifiability can be achieved if the message is first encrypted and then signed (see also section 10.1.4)

#### 10.1.2.1. Certificates

X.509 certificates comprise the public key of an entity and additional mandatory information such as the subject (holder) of the certificate, its issuer and its validity period (RFC 5280, p. 18). The certificates are described using an Abstract Syntax Notation One (ASN.1) structure with Distinguished Encoding Rules (DER) encoding that represent each element with tag-length-value encoding (RFC 5280, p. 16).

In the public key infrastructure that is usually used in the web today, certificates are issued by Certificate Authorities (CAs, see also section 7.1.1). The CAs have the duty to validate that the holder of the certificate actually has the attributes and keying material (task **An1**) that are then included in the certificate (task **D1**). To validate a certificate, the CA's certificate is required. It is again issued by a CA, unless it is the root of the certificate chain. In TLS, all certificates of the certificate chain must be transmitted in the handshake. The only exception is the root certificate which may be omitted (RFC 5246, p. 48).

The certificates may already contain much of the required information for performing the delegation tasks, e.g., the claim statement that contains some of the holder's attributes, e.g., the common name, and the issuer's endorsement (see

also section 5.4). A difficult problem is satisfying the **claim issuer authorization fundamental** which requires that the overseeing principal authorized the claim issuer, in this case the issuing CA. As described above, certificate chains end in root certificates; they are assumed to be already known to the communicating endpoints (RFC 5246, p. 48). To satisfy the **principal fundamental**, overseeing principals must approve of the root CAs. In the Web, this is difficult to achieve even for technically versed users, especially considering the vast amount of root certificates<sup>1</sup>. Instead, overseeing principals that install a browser on their system implicitly authorize the browser vendor to choose the root certificates. This satisfies the **principal fundamental** only to a certain degree (see also section 7.1.2). The trust model has fundamental weaknesses (cf. (Roosa and Schultze 2010), (Kasten et al. 2013)), and attacks on CAs repeatedly led to the fraudulent issuing of certificates (e.g. incidents at Comodo (Comodo 2011) and DigiNotar (Nightingale 2011)), thereby undermining the security association between client and server. Centralized trust roots are not available for the big Internet, and will be difficult to achieve (and are not desirable anyway) in the IoT. Constrained devices will have difficulties to store large numbers of root certificates.

DTLS with certificates may assist with task **A6** and **D6**, task **An1** and most of the delegation tasks, but an authorization solution that uses certificates must solve the problem of ensuring the authorization of the root certificates for task **D4**. To be useful for the authorization process, the authorization rules specified by the overseeing principals must refer to entities with attributes that are mentioned in the certificates. Otherwise, additional claims must be provided that relate the verifier to the permission. Today, authorization solutions often provide an endpoint with an authorization claim where the permission is directly bound to a verifier (e.g., OAuth 2.0 proof-of-possession tokens). In this case, the information in the certificate is not necessary link the permission to the communication partner.

Devices that use X.509 certificates must have a clock and use a time synchronization mechanism. Otherwise, they are not able to check if, according to the certificate's validity period, the certificate is still valid. Certificates are therefore

---

<sup>1</sup>E.g., in September 2020, the browser Firefox comprised 138 root certificates (see also <https://ccadb-public.secure.force.com/mozilla/CACertificatesInFirefoxReport>)

not suitable for constrained devices with only a very limited ability to measure time. They require a different mechanism.

Because of the encoding and the required fields, X.509 certificates are relatively large and may become even larger if extensions are used. The certificate chains therefore require a lot of storage capacity. If DTLS is used in the certificate mode, the certificate chains are transmitted in every handshake, which increases the network load. Having to store a vast number of root certificates would provide an additional strain. Hummen et al. state that the certificate-based handshake has static RAM requirements of about 6 KiB (Hummen et al. 2014, p. 3), which is more than half of the available RAM on class 1 devices (see section 3.1). Constrained devices with limited energy, RAM and storage space are therefore hard-pressed if they need to use X.509 certificates. Also, issuing a certificate to every smart object is expensive and requires a lot of effort. As shown above, certificates may constitute only unnecessary overhead in the authorization process for constrained devices.

#### **10.1.2.2. Raw Public Keys**

If DTLS is used with Raw Public Keys (RPKs), only the SubjectPublicKeyInfo structure of the X.509 certificate is used. It comprises the public key and information about the algorithm with which the key is used (RFC 7250, p. 25). A raw public key therefore is much smaller than an X.509 certificate.

Unlike X.509 certificates, raw public keys (RPKs, RFC 7250) are not bound to administrative information. They can be used as a verifier, similar to symmetric keys, and thus help with task **A6** and **D6**. The other delegation and authorization tasks must be performed additionally.

#### **10.1.3. Forward Secrecy**

For a protocol that has Forward Secrecy (FS), compromising the long-term keying material does not compromise previously derived session keys (RFC 4949, p. 218).

FS can, e.g., be achieved by using a Diffie-Hellman (DH, Diffie and Hellman 1976) key agreement algorithm.

Protocols without FS are less able to satisfy the fundamentals: if an unauthorized entity gets hold of a key, the authorization rules do not refer to this entity and the **correlation fundamental** is violated. The data that the key is supposed to protect is no longer secure: its confidentiality is breached.

#### 10.1.4. Encryption and Message Authentication

Message authentication, i.e., validating the source of a message, can be performed by generating a signature or a Message Authentication Code (MAC) of the message. To sign a message, an entity performs the signature algorithm defined for its private key on the message (or a hash of the message). MACs are generated from the message with a symmetric key and a MAC function (Ferguson et al. 2010, p. 26). MACs can be generated using block ciphers, e.g., with the cipher block chaining (CBC) mode. An alternative is keyed-hashed Message Authentication Code (HMAC, RFC 2104) which uses a symmetric key as an input to a cryptographic hash function such as SHA-256 to compute a MAC.

Message authentication assists with task **A6** and **D6** because it enables the receiver of a message to determine if the sender is able to use certain keying material. The receiver can thereby determine if the sender is the holder of certain attributes or permissions. Message authentication can also be used for tasks **A4** and **D4**. Additionally, it may help with tasks **A3** and **D3**, because it provides integrity and thus binds the pieces of information in a message to each other.

Encryption can be used for data destination verifiability, e.g., to define the destination of a claim. To satisfy the **claim origin fundamental**, the relevant information about the claim, i.e., the claim statement, the holder of the claim and the claim destination, must be bound together. That means that if encryption is used to achieve data destination verifiability, messages must first be encrypted, before the MAC or signature is generated. Otherwise, the encryption does not provide data destination verifiability. Without additional protection, the protocol then is susceptible to man-in-the-middle attacks.

Authenticated Encryption with Associated Data (AEAD) combines encryption and message authentication (RFC 5116, p. 3). It therefore provides data destination verifiability. AEAD reduces the amount of computation and allows for simpler interfaces (RFC 5116, p. 4). Also, a single key is used for both encryption and MAC generation.

As described in section 10.1.1, symmetric verifiers on their own are not sufficient to exclusively identify an entity. Modern protocols usually have own symmetric communication keys for each direction. E.g., in TLS and DTLS the *client write MAC key*, the *server write MAC key*, the *client write encryption key*, and the *server write encryption key*<sup>2</sup> are derived from the master secret (RFC 5246, p. 25). Each side therefore has its own keys for encryption and generating a MAC. For AEAD algorithms, clients and servers use their respective write encryption keys for encryption and MAC generation. The MAC keys are not used (RFC 5246, p. 24). The distinct keying material for each direction enables endpoints to determine if they send a message themselves, and if they are the intended destination of a message, unless the peer intentionally uses the wrong keying material. This approach thus assists with the tasks **A5** and **D5**, tasks **A6** and **D6**, tasks **A3** and **D3**, and tasks **A4** and **D4**.

### 10.1.5. Object Security

As an alternative to DTLS, an object security solution might be used. It protects the data per packet on the application layer. This approach can help in cases where no uninterrupted DTLS channel can be established between the communication partners, e.g. because the communicating endpoints use different underlying protocols (e.g., TLS and DTLS), or different application layer protocols (e.g., HTTP and CoAP) that need to be translated by intermediaries. Since DTLS protects the whole upper layer content, intermediaries are not able to access selected parts of the transmitted data (see also section 9.3.3). The secure communication therefore must end on the intermediary. In this case, only trusted caching and translating intermediaries can be supported.

---

<sup>2</sup>Some cipher suites additionally provide a client write IV and a server write IV, but this is not relevant for our work.

Object security solutions can be used to protect the complete CoAP message (i.e., the complete payload of the UDP datagram). But they can also enable the access to selected application level data. We will call this approach *object security with invasive intermediaries*.

One of the most difficult parts in building an object security solution is to determine which parts of a message can be accessed by intermediaries without violating the authorization fundamentals and compromising the security. Untrusted intermediaries with which the endpoints have no security association must only access data that do not have confidentiality, integrity or authenticity requirements. Untrusted proxies may also not be reliable, which may compromise the availability of the data that is handled by them. Object security solutions are mainly useful in cases where proxies are trusted to some degree and may access parts but not all of the transmitted data. In this case, endpoints need to establish security associations with the intermediaries.

The security of object security solutions is not in scope for this work. Where necessary, we will define which requirements underlying security solutions must meet to be suitable for the protection of data. A detailed analysis of how object security approaches that enable intermediaries to access sensitive data must be defined to comply with the authorization fundamentals is beyond the scope of this thesis and is a topic for future research.

CBOR Object Signing and Encryption (COSE, RFC 8152) is a solution for securing data objects in constrained environments. COSE describes the generation and processing of data objects with signatures, MACs and encryptions. An important element is the COSE key structure for describing cryptographic keys. COSE uses CBOR, which was specifically designed for devices with very limited hardware resources (see also section 3.3.4), as encoding format. COSE messages are structured by CBOR arrays. The message type is identified by a CBOR tag, e.g., a COSE signed data object has the CBOR tag 98, and a cose encrypted data object has the tag 96 (RFC 8152, p. 9). COSE also provides data structures to transport information about the content such as the utilized algorithms or keys. Figure 10.2 shows a COSE MAC object<sup>3</sup>.

---

<sup>3</sup>Based on RFC 8152, p. 113

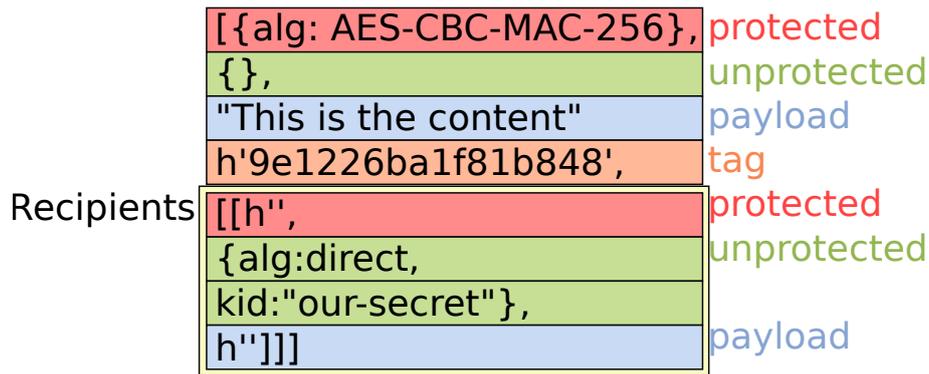


Figure 10.2.: COSE MAC Object

### 10.1.6. Discussion

As described in the previous sections, both symmetric and asymmetric keying material can be used as verifiers and thereby assist in satisfying the authorization and delegation fundamentals. The necessary claims must be provided to the communication partners in conformance with the authorization and delegation fundamentals. Claim issuers that influence the authorization must be trustworthy. If the **principal fundamental** is not satisfied, the whole authorization process may become ineffective. This problem manifests in the trust model Web applications use today, where certificates are issued by CAs. As we discussed in section 10.1.2.1, the trust model has fundamental weaknesses, and attacks on CAs repeatedly lead to the fraudulent issuing of certificates.

Semantic information in the X.509 certificates, e.g., the domain name, is not always helpful for the authorization. The certificates may therefore not contain the required information but provide only unnecessary overhead. X.509 certificates can only be used by devices that have an RTC and use a time synchronization mechanism. Otherwise the devices are not able to check if the certificate is still valid. Considering all these disadvantages, certificates are not a promising approach for constrained environments.

Asymmetric keying material has the advantage that it can exclusively identify the holder of a claim. Also, only asymmetric approaches can be used for third-party verifiability. But the necessary operations are expensive. Constrained devices

may be equipped with hardware security modules that perform difficult cryptographic operations. The modules increase the cost and energy consumption but may be useful for some applications.

Symmetric algorithms are less expensive and usually have a shorter key length than comparably secure asymmetric approaches. In contrast to asymmetric approaches, symmetric algorithms are expected to be comparably resistant against quantum computing. Because of these advantages, symmetric solutions are a good fit for constrained environments. If asymmetric solutions are used, raw public keys are preferable, as they have less overhead than certificates. Both symmetric keying material and raw public keys require that the necessary semantic information, i.e., the claim for which they are the verifier, is generated, provided and validated in compliance with the authorization and delegation fundamentals. Security solutions must consider this in order to be effective.

## 10.2. Validity of Authorization Information

In section 5.2.3, we described that the enforced permission must reflect how the overseeing principals' authorization decision would be at this precise moment to comply with the **rule completeness fundamental**. Similarly, the **claim completeness fundamental** demands that the claim statement reflects the opinion of the relevant claim issuers at this moment. Although real applications can satisfy these fundamentals only to a certain degree, they must sufficiently satisfy the fundamentals to be secure (see also section 5.8).

Claim issuers (including overseeing principals) are not always present during the communication. And even if they were, it would be inconvenient for them to continuously express their opinion. If claims are obtained using a network connection, the respective messages might get lost. Also, the transmission of messages always takes time and claim issuers might change their minds in the meantime.

Endpoints might be notified of changes by updates and revocations. But there is no guarantee that update and revocation messages reach their destination. The

connection between endpoint and claim issuer might be disturbed or messages might be intercepted. The endpoint will not know if it missed such a message, and will not be able to consider the new claims, unless endpoints contact the claim issuers regularly. In this case, the endpoint must have a means to measure the passing of time.

An alternative to revocation is to provide a *timestamp* with the claim, maybe in combination with a *lifetime*. The timestamp declares when the claim was generated and assists the endpoint in determining how fresh the claim is. The lifetime can additionally inform the endpoint how long the claim is supposed to be valid. Without a lifetime, the endpoint must decide by itself whether a claim is still valid, likely by using a timeout. The use of lifetimes allows for a more flexible definition of validity periods. However, the claim issuers still need to predict how long the claim needs to be valid at the time when it is provided to the endpoint. Unexpected circumstances, e.g., that a peer becomes compromised, are difficult to consider. Additionally using a revocation mechanism may mitigate this problem.

The use of timestamps requires the communicating endpoints to have synchronized clocks. The more the clocks diverge over time, the less rule completeness can be achieved. Time synchronization messages might get lost or be blocked by attackers. Also, time synchronization protocols may be attacked, which may even allow an attacker to alter the target's time (see, e.g, Malhotra et al. 2017). **Claims that assist an endpoint with time synchronization influence the authorization and therefore must stem from an authorized claim issuer.**

For all approaches that satisfy the **rule completeness fundamental** and the **claim completeness fundamental** the following holds: **An endpoint must be able to measure the passing of time in some way to be able to participate in the protection of the security objectives of its overseeing principals. Also, the completeness fundamentals cannot be sufficiently satisfied if the endpoint is not able to (directly or indirectly) communicate with a trusted claim issuer regularly.** The endpoint must at least be able to get either revocation/update messages or time synchronization messages. How often an endpoint must communicate with a trusted claim issuer depends on the accuracy of its clock and on the requirements of the application scenario.

For a constrained device it may be difficult to satisfy the **rule completeness fundamental** and the **claim completeness fundamental** since it may only have a limited ability to measure time (see section 3.1.4).

If multiple sets of authentication and authorization information exist, the endpoints must know how to handle them. In many cases, only the most recent claims are valid. In this case, the endpoints must be able to determine which the most recent claim is. This can be achieved if timestamps or sequence numbers are bound to the claim.

### 10.3. Nonces and Time

Another fundamental where time is important is the **correlation fundamental**. To achieve it, endpoints must be protected against *replay-attacks* where an attacker re-inserts previously sent data into the communication. These attacks can be prevented by ensuring that a piece of data has only been received once. In the literature, this problem is often associated with freshness (see, e.g., Burrows et al. 1990, p. 20; Anderson 2008, p. 66).

The usual approach to protect against replay attacks is to include a *nonce* in the message. Nonces are random or non-repeating values (RFC 4949, p. 200). Fresh nonces must be used for every message, and the receiver must validate that the nonce is unique in this context. Nonces must be bound to the content of the message they are meant to protect. The integrity of these messages including the nonce must be validated.

One way to generate a nonce is to use a unique random value. The receiver must store all received nonces at least as long as the same keying material is used; a message can be replayed if the nonce is no longer remembered. The more nonces an endpoint needs to store and the longer the nonces are, the more storage space is required. Nonces may also be used to determine if a response is fresh: endpoint A includes a nonce in the message to endpoint B which B must reflect. If A receives the nonce back during a certain timespan, it knows that the response is fresh.

A different approach is using timestamps as nonces. In this case, the receiver can easily determine if the message is fresh and does not have to store old nonces. But sender and receiver are required to have synchronized clocks which may be difficult to achieve, especially for constrained devices (see also section 3.1.4).

Without synchronized clocks, timestamps may be used as incrementing nonces (counters). Counters do not have the clock requirement, and endpoints have to store only a few nonce values. Solutions with counters must specify how overflows are avoided, e.g., by defining when and how counters are resetted. Also, endpoints may need to deal with situations where the counters run out of sync. TLS and DTLS use counters in the form of sequence numbers to protect against replay attacks (see RFC 5246, pp. 94–95, RFC 6347, p. 13). If counters are used, receivers are able to decide if a message is more recent than another, but without additional measures they cannot determine how old the message is.

## 10.4. Conclusion

In this chapter, we analyzed approaches for satisfying the authorization and delegation fundamentals. We showed that both symmetric and asymmetric algorithms may assist in performing tasks **A6** and **D6**, **A3** and **D3**, and **A5** and **D5**. X.509 certificate mode may additionally provide the necessary information for performing the delegation tasks, but the common name in the certificate may not be meaningful for the authorization. They therefore may provide only unnecessary overhead. Because of their encoding and structure, X.509 certificates are relatively large. Constrained devices are hard-pressed if they are required to handle certificate chains and store vast amounts of root certificates. Also, the X.509 certificates require constrained devices to have a clock and use a time synchronization mechanism to be able to check the validity of the certificate.

Raw public keys produce less overhead but, like pre-shared keys, require that semantic information is securely bound to the key. Only asymmetric approaches can be used for third-party verifiability. But asymmetric algorithms will no longer be secure if quantum computing advances. Quantum attacks are likely not feasible in the near future, but constrained devices will often be operated for a very

long time, especially if they are installed in places where they are difficult to update and maintain, e.g., in walls (see also section 3.2).

Symmetric keys require that the confidential key is transmitted or negotiated securely between the communication partners. Symmetric keying material typically has a shorter key lengths than asymmetric keys with comparable security. The operations that are required for symmetric algorithms are easier to compute. Even if asymmetric keying material is used for the authentication, solutions usually use symmetric keys for the subsequent communication. Symmetric keys therefore are a good fit for constrained devices.

To comply with the **rule completeness fundamental** and the **claim completeness fundamental**, endpoints must check if authorization rules and claims are up to date. Timestamps and lifetimes enable endpoints to determine the expected validity period of a claim, but unexpected circumstances may require claims to become invalid before their time. Revocation mechanisms may help in this case, but revocation messages must reach the endpoints. If multiple sets of authorization rules exist, endpoints must know how they are to be handled. If only the newest set of authorization rules are valid, endpoints must be able to determine which set is the most recent.

In general, endpoints can only participate in the protection of their overseeing principals' data if they have some means to measure the passing of time. Also, endpoints can only sufficiently satisfy the fundamentals if they regularly communicate with a trusted claim issuer to receive authentication- and authorization-related updates and/or time synchronization messages.

To satisfy the **correlation fundamental**, messages must be protected against replay attacks. Nonces enable endpoints to determine if the message is fresh. In some cases the same mechanism can be used for checking the validity and the freshness of a message.

*"But I don't want to go among mad people", Alice remarked. "Oh, you can't help that", said the Cat: "we're all mad here".*

– Lewis Carroll (Alice's Adventures in Wonderland)

## 11. ACE DTLS Profile for Constrained Environments

In the previous chapter, we introduced the authentication and authorization architecture for constrained environments. To enable constrained devices to perform authenticated authorization, the architecture must be implemented in a protocol.

As described in section 2.5, the OAuth 2.0 framework is widely used in the big Internet, but is insufficient for building a secure, interoperable implementation; profiles and extensions are required to provide the necessary information. Since devices in the IoT may have various limitations (see also chapter 3), they might have difficulties to use OAuth 2.0 profiles and extensions for the big Internet. Therefore, the ACE working group defines a new framework for authentication and authorization (draft-ietf-ace-oauth-authz-35) that is loosely based on the OAuth 2.0 framework.

In this chapter we will introduce our DTLS profile for the ACE framework, and evaluate its usefulness for constrained environments. The profile uses DTLS for the secure transmission of data on the constrained level. IETF drafts are work in progress. In the general description, we will use the version 35 of the ACE framework (draft-ietf-ace-oauth-authz-35) and version 13 of the profile (draft-ietf-ace-dtls-authorize-13), unless stated otherwise. The evaluation we provide in this chapter was performed on earlier versions as described in section 11.2.

The chapter is organized as follows: in section 11.1 we introduce our DTLS profile. In section 11.2 we evaluate the profile by analyzing if it satisfies the authorization fundamentals (see also chapter 5). Section 11.3 summarizes the findings of this chapter.

As already described, the OAuth terminology differs from REST and CoAP, and some terms even have a different meaning. To avoid confusion, we continue to use CoAP and REST terminology and extend it where necessary. A comparison to OAuth 2.0 terminology can be found in section 9.1.

In the Delegated CoAP Authenticated Authorization Framework (DCAF, see chapter 12), the client's authorization manager can provide information about the server to the client as we will describe in section 12.2.4.2. This information was called client information. The ACE framework picked up this concept, and renamed it first to RS information and later to access information. To achieve consistency, we use the term *client information* throughout this work for information about the server that is meant for the client. Similarly, the framework adopted the AS information message from DCAF, which was later renamed to AS request Creation Hints. To achieve consistency, we will use the term SAM information for data about the server's authorization manager that is provided to C by the server.

## 11.1. DTLS Profile Overview

Like OAuth 2.0, the ACE framework only addresses cases where a single authorization manager is used (CAM is assumed to be integrated in C) and thus only implements one of the four possible architectures of the task delegation architectural style (see also section 9.3). The most general four-corner architecture is not supported.

Our profile describes how a client obtains access tokens from the server's authorization manager to establish a secure DTLS session and gain access to a resource on the server. Access tokens in the profile are proof-of-possession (pop) tokens (draft-ietf-ace-dtls-authorize-13, p. 3): a symmetric or asymmetric key is bound to the token, and entities perform proof of possession by showing that they are able to use this key.

Access tokens that allow C to access resources on S are issued by S's authorization manager SAM. To determine how SAM is contacted, C may send S an unau-

thorized request (see section 11.1.1). S will decline the message, and reply with SAM's address.

After obtaining the address, C contacts SAM to request an access token. If SAM decides that C is to be authorized, it generates an access token response, and transmits it to C. This information can then be used by C to establish a DTLS channel with S. The communication flow is depicted in figure 11.1. Messages in square brackets are optional.

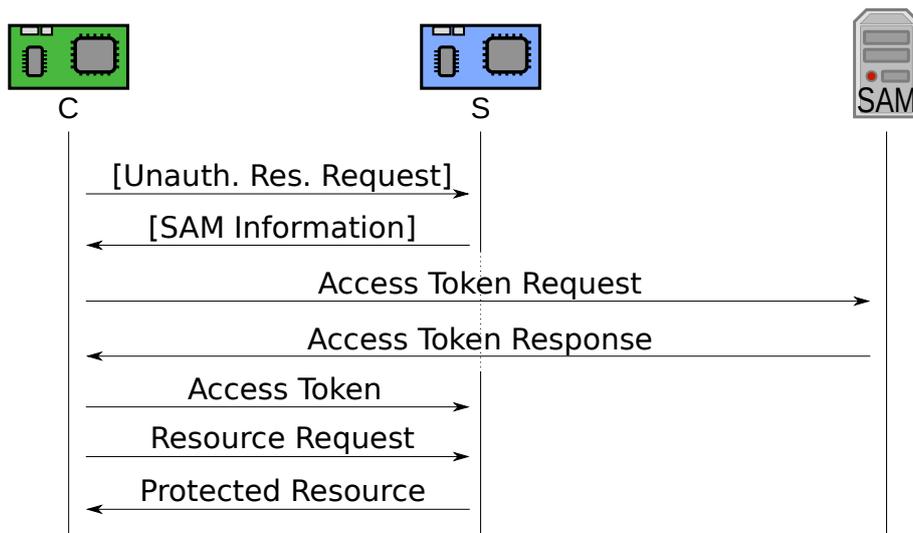


Figure 11.1.: DTLS Profile Flow

### 11.1.1. Obtaining SAM Information

Clients may obtain information about SAM from S using an unauthorized request message as described above. S replies to an unauthorized resource request with the SAM information message (draft-ietf-ace-dtls-authorize-13, p. 4). S may add a nonce to the message to prevent replay attacks (see also section 11.1.5).

At this point, C and S are not yet able to communicate securely. Since C cannot validate that the message actually stems from S, it cannot be sure that the provided SAM information is correct.

Instead of sending an unauthorized request message, C might obtain information about SAM from a resource directory (see section 3.5.2). How this is accomplished is not part of the profile.

### 11.1.2. Obtaining the Access Token

C obtains access tokens from SAM. C must only communicate with SAM if a security association exists between them or can be established. A constrained client might not be able to establish security associations on its own. The communication between C and SAM must be integrity-protected and encrypted (draft-ietf-ace-oauth-authz-35, p. 45). Also, the access token must be integrity-protected (draft-ietf-ace-oauth-authz-35, p. 41).

To obtain the token, C sends an access token request to SAM that may contain information about the S with which C wants to communicate in the `audience` field (draft-ietf-ace-oauth-authz-35, p. 22), and the client's keying material in the `req_cnf` field (draft-ietf-ace-dtls-authorize-13, p. 7).

The profile defines two modes for the establishment of the DTLS channel between C and S. The pre-shared key mode uses symmetric keys that must be securely provided to the communicating parties prior to the communication. The raw public key mode deploys an asymmetric approach where an entity provides only its RPKs. The client must indicate in its request to SAM which mode it wants to use. SAM validates that C is authorized and then generates an access token for C. Additionally, SAM may add client information to the token, e.g., to transmit a symmetric key.

The OAuth 2.0 framework uses access tokens to transmit information about the authorization to the server. A token type for the OAuth 2.0 framework is the JSON Web Token (JWT, RFC 7519). It is encoded in JavaScript Object Notation (JSON), and used to transport claim information such as the issuer, the audience, i.e., the entity that is intended to process the claim, and the expiration time (RFC 7519, p. 9).

As an alternative for JWTs, the CBOR Web Token (CWT, RFC 8392) format was developed. It is optimized for constrained devices and uses CBOR (see also section 3.3.4), which allows for a more compact content encoding than JSON. In the ACE framework, both the JWT and the CWT format may be used (draft-ietf-ace-oauth-authorized-35, p. 10). CWTs are protected using COSE (see section 10.1.5).

Access tokens in the ACE framework comprise an OAuth 2.0 `scope` and a confirmation (`cnf`) field (draft-ietf-ace-oauth-authorized-35, p.36). The `scope` indicates authorization rules (Hardt 2012, p. 49). The `cnf` structure contains information concerning the proof-of-possession key, i.e., the key itself or a reference to it. The token must be validated by the server before granting access to the client (draft-ietf-ace-dtls-authorize-13, p. 16). Only if the token is valid, a DTLS session is established. For each request that the client sends in the DTLS connection, the server must check if the requested resource is covered by the authorization rules that it can determine with the help of the access token.

### 11.1.3. Pre-Shared Key Mode

In the pre-shared key mode, a symmetric key is used for the DTLS session between C and S. The information for the establishment of the DTLS channel is provided by SAM together with the access token.

To obtain the token, C sends a token request to SAM. C may include an identifier for a symmetric key in the request to indicate which key SAM should use in the access token. This can, e.g., be useful for authorization information updates; C and S can continue to use the existing DTLS channel instead of having to establish a new session. The key must have been previously used in a communication between C and S. The confidentiality, integrity and authenticity of the communication between C and SAM must be protected (draft-ietf-ace-dtls-authorize-13, p. 6).

C must then send the token to S. C can either transmit the token to the resource `/authz-info` that is reserved for the upload of access tokens, or it can provide

the token in the `psk_identify` field in the DTLS handshake (draft-ietf-ace-dtls-authorize-13, p. 15). The authorization information on S can be updated by uploading a new access token to the `/authz-info` resource on S (draft-ietf-ace-dtls-authorize-13, p. 18).

#### 11.1.4. Raw Public Key Mode

In the RPK mode, C sends its raw public key or an identifier for that key to SAM. If C's request is authorized, SAM returns an access token that includes the key or its identifier. As in the PSK mode, the confidentiality, integrity and authenticity of the communication between C and SAM must be protected.

To transmit the access token to S, C sends a `POST` request to `/authz-info` on S. If S sends a positive response, C starts to establish the DTLS channel. It must use its raw public key in the DTLS handshake to perform proof of possession for the access token (draft-ietf-ace-dtls-authorize-13, p. 9).

#### 11.1.5. Resource Access

With the information provided by SAM, C can establish a DTLS channel with S. C uses the key that is bound to the access token in the DTLS handshake. Thus, S can validate that the presented access token was actually issued to C, and that C is authorized to access resources covered by the access token.

## 11.2. DTLS Profile Evaluation and Improvements to the ACE Framework

An endpoint that participates in the protection of data must satisfy the authorization fundamentals (see chapter 5). The degree to which the authorization fundamentals are satisfied determines the quality of the authorization solution. In the following we provide a summary of the authorization fundamentals analysis of

the DTLS profile. Many important details are specified in the ACE framework's main document, which is therefore included in the analysis where necessary. The complete analysis can be found in Appendix B. We previously published an abbreviated version of the DTLS profile analysis (Gerdes et al. 2018a).

Since IETF working group drafts are work in progress, new versions of the DTLS profile and the ACE framework were developed during the writing of this thesis. The complete analysis in Appendix B was performed using draft-ietf-ace-dtls-authorize-01 and draft-ietf-ace-oauth-authz-06. We present a summarized analysis of the draft versions draft-ietf-ace-dtls-authorize-04, draft-ietf-ace-oauth-authz-16 and draft-ietf-ace-oauth-params-00 below. After our analysis, we provided a review with the most important issues in the ACE framework to the ACE working group<sup>1</sup>. Many of the identified issues were fixed by the authors in draft versions draft-ietf-ace-oauth-authz-17 and draft-ietf-ace-oauth-params-01 as we will show below. We will also describe the status of the drafts draft-ietf-ace-dtls-authorize-13, draft-ietf-ace-oauth-authz-35 and draft-ietf-ace-oauth-params-13, which were the most recent at the time of writing.

The goal of the authenticated authorization solution is to enable C and S to communicate securely (see also section 9.3). To achieve this, C and S must be able to authenticate each other and validate each other's authorization. A device that participates in the protection of data must satisfy the fundamentals and thus be able to perform the authorization tasks (see section 5.2.3). Constrained endpoints that are not able to perform authorization tasks on their own must delegate those tasks to other devices, the authorization managers. Both C and S may be constrained (draft-ietf-ace-oauth-authz-35, p. 9) and they may have distinct principals (draft-ietf-ace-oauth-authz-35, p. 12).

One important observation we made during our analysis is that the information that is required to build a secure solution is scattered over several documents and therefore difficult to find. For the analysis we specifically searched for details on how the authorization and delegation tasks are meant to be performed. Software developers may miss important details and thus may make wrong design decisions that lead to vulnerabilities. Even worse, a lot of important details

---

<sup>1</sup><https://mailarchive.ietf.org/arch/msg/ace/z5xH0-uzsObLplsC1mBRc89THJU>

are missing in the analyzed versions of the documents. Even though it may be possible for an experienced implementer to correctly fill the gaps and implement a secure solution, the quality of the ACE framework and DTLS profile would increase considerably if the gaps were filled by the specifications.

As a general advice, we indicated in our review that the security of the solution in many cases relies on responses being securely bound to requests, e.g., if SAM provides client information to C, the server's attributes are only specified in C's request to SAM. The receiver of a response must be able to validate that a response actually belongs to a certain request. The ACE framework addresses this issue since version 17 (draft-ietf-ace-oauth-authz-17, pp. 37–38; draft-ietf-ace-oauth-authz-35, pp. 45–46).

### 11.2.1. Server Side Delegation

The main purpose of the ACE framework is the delegation of authorization tasks from the server to the server authorization manager. SAM must obtain (actively or passively) authorization information for C from SOP and provide them to S. For a secure task delegation process, SAM and S must perform the delegation tasks. S must validate C's authorization with the assistance of the information provided by SAM. How S and SAM establish a security association with each other is out of scope for the ACE framework and the DTLS profile.

Although the setup of the security association between SAM and S is out of scope, the ACE framework and the DTLS profile should specify how the security association is validated. The framework and profile should at least specify that SAM and S are expected to perform mutual authorization. Also, the documents must specify how SAM and S perform task **A6**, respectively, i.e., how they must protect their communication. Up to version 16, the framework and the DTLS profile did not require the authenticity of the token, and S did not check if the token stems from a SAM that is authorized by SOP. **Attackers would thus have been able provide their own access tokens to S.** In our review of the ACE framework, we proposed that the server must check SAM's authorization to provide access

tokens (item “Authorization of the AS on the RS side”). After our review, the authors included this requirement in the document (draft-ietf-ace-oauth-Authz-18, p. 33; draft-ietf-ace-oauth-Authz-35, p. 37).

The ACE framework does not clearly specify which pieces of information the access token must contain. S must be able to determine the claim statement (the authorization rules defined in the scope), the destination (given in the audience) and the holder (specified in the `cnf` structure). **If required data is missing, S cannot perform the necessary validations and the authorization is ineffective.** S must validate that this information is bound together and was provided by the same authorized SAM. In our review, we proposed that the framework describes how S must react to missing fields and missing authenticity in the access token. The authors mitigated this problem by providing a section on verifying an access token (draft-ietf-ace-oauth-Authz-17, pp. 33–34; draft-ietf-ace-oauth-Authz-35, pp. 37–39). Unfortunately, the authors only specify how the server must verify data that is contained in the access token, but do not define how the server must react if data is missing. There is no indication that the access token must contain a scope (see, e.g., draft-ietf-ace-oauth-Authz-35, pp. 35–36). How the server has to react if the scope is missing is not specified. This gap may, e.g., lead to situations where SAM does not specify access permissions because it believes S to already have them, while the server assumes that the missing scope means that the client has unlimited access. The framework states that it is important that the token contains an audience (draft-ietf-ace-oauth-Authz-35, p. 43), but does not state how the server must react if the audience is missing. In the worst case, the server may accept an access token that was meant for a different server. The framework by default uses proof-of-possession tokens (draft-ietf-ace-oauth-Authz-35, p. 15), which may indicate that the `cnf` structure is assumed to be present in the token. How the server must react to a missing `cnf` parameter is not specified.

Symmetric keying material that is not confidentiality-protected is no longer secure after it was transmitted. The server should therefore decline and discard access tokens with a symmetric keying material that is not encrypted. The ACE framework does not address this problem in version 35. The **holder relation fundamental** is not satisfied: the keying material cannot be used to identify the client. Since SAM must encrypt symmetric keys (draft-ietf-ace-oauth-Authz-35,

p. 42) and the server must check if the token stems from an authorized SAM, this problem likely only occurs if SAM made a mistake or was compromised. Nevertheless, S should check if symmetric keying material actually was encrypted and not solely rely on the AM.

How SAM obtains authorization rules in behalf of S from SOP (tasks **A1** to **A5**) is out of scope for the ACE framework and the profile. To obtain a grant a client can use various protocol flows that are defined as the grant types. They may specify how SOP and SAM perform the authorization tasks **A1** to **A5**. E.g., the “device flow”, i.e., the OAuth 2.0 device authorization grant (RFC 8628), was designed for clients that do not have a convenient interface for user interaction. It specifies how SOP can use a less-constrained device such as a smartphone to authorize a certain client to access her resources. The analysis of the grant types is not in scope of this work.

Even though obtaining the authorization rules is out of scope, the framework must specify how SAM ensures that the authorization rules that SOP defined for C must actually refer to this C, e.g., by using certain keying material. Otherwise the **correlation fundamental** and the **principal fundamental** cannot be satisfied. The framework specifies that the client must present a valid grant for the requested scope (draft-ietf-ace-oauth-authz-17, p. 19; draft-ietf-ace-oauth-authz-35, p. 22). Since version 17, the framework also demands that C and SAM share a secret or know each other’s public keys to establish secure communication (draft-ietf-ace-oauth-authz-17, p. 37; draft-ietf-ace-oauth-authz-35, p. 45). But, the document does not state that the access token scope must reflect SOP’s permissions and must be bound to the keying material of the client that has these permissions. **If SAM does not set the access token correctly, S may communicate with an unauthorized C.**

S must validate that the access token is up to date to comply with the **claim completeness fundamental**. Not all of the three methods that the ACE framework defines to solve this problem are effective. In the first approach, the token contains an expiration date. It can only be used by devices that have a clock and regularly synchronize their time.

In the token introspection approach, S asks SAM if the token is still valid. The

framework does not state if and how the server checks that the introspection response is valid (see also section 11.2.1.2). Also, the approach does not enable S to determine the validity of the token. With only the `exp` parameter, i.e., the lifetime of the token, S does not know when the token introspection response was generated. To make this approach work, S could, e.g., set a timeout when sending the token introspection request, and accepts a response only until the timeout is reached. However, S must be able to measure the time until the timeout is reached. If S turns off its clock during sleep, S must not sleep during this time or it cannot use this mechanism. **If no improvements are made to the token introspection approach, S may be tricked into accepting outdated access tokens.** At the time of writing, this issue was not addressed by the ACE framework. Token introspection is also vulnerable to DoS attacks, as we will discuss in section 11.2.1.2.

Up to version 16, the framework specified a sequence number approach, where S remembers the most recent sequence number and only accepts tokens within a certain range of this number (draft-ietf-ace-oauth-authorization-16, p. 34). This approach is not suitable to satisfy the **claim completeness fundamental**, since S cannot determine if it has the most recent access token. **This approach is particularly harmful because the client may benefit from withholding new, more restrictive access tokens from S. S will then continue to use the old access token. Therefore, this approach must not be used unless there is an additional communication channel between S and SAM that regularly informs S about the most recent sequence number.** In version 17, the sequence number approach was removed from the ACE framework. Instead, the token may now contain the lifetime of the ticket and a sequence number (draft-ietf-ace-oauth-authorization-35, p. 41). Since the server does not know when the token was created, the lifetime starts when S first receives the token. The token thus is only valid for a certain time. But, a client can still withhold the token. A malicious client may thus, e.g., request several tokens from SAM while it still has certain permissions, and provide them to the server after these permissions were revoked.

With the SAM discovery, the ACE framework also adopted a nonce mechanism from DCAF that assists the server in determining if the access token is fresh

(draft-ietf-ace-oauth-authz-35, p. 19). We will describe the original mechanism in section 12.1.7.

#### 11.2.1.1. Access Tokens

Up to version 16, the ACE framework did not describe how the server is supposed to manage access tokens and the `/authz-info` resource to which the tokens are uploaded.

When C initially transfers the access token to S, C and S do not yet have a security association, and S cannot validate that the access token was transmitted by an authorized C. Sending an access token to S takes considerably less effort than it takes S to receive and validate the token. An attacker therefore might use the `/authz-info` resource to perform DoS attacks on the server. An attacker that eavesdrops on the unprotected channel can easily obtain access tokens and replay them to S. Even though we described this threat in our review, the ACE framework still does not mention it in version 35.

If tokens are not replaced or otherwise revoked, the server must store them as long as they are valid, even if C never contacts S. S therefore unnecessarily wastes storage space. A malicious C might try to flood S by requesting various access tokens for varying resources. This problem can be mitigated if S keeps unused access tokens only for a short period of time. This problem is still not mentioned in version 35 of the ACE framework.

If access tokens are transmitted in the DTLS handshake, the server can discard the token if the client cannot finish the handshake. This approach therefore wastes less storage space. Also, DTLS offers more protection against DoS attacks. Transmitting the access token in the DTLS handshake therefore is preferable for unauthorized clients.

New access tokens might replace former access tokens, but it is also possible that the authorization rules are combined. To comply with the **rule completeness fundamental**, S must know if former authorization rules are invalidated by new authorization rules. If access tokens are replaced by new tokens, S must know

which token is the most recent. Otherwise an attacker might trick S to use a reinserted older token. Since version 17, the ACE framework recommends that only one access token is stored per key (draft-ietf-ace-oauth-Authz-17, p. 32; draft-ietf-ace-oauth-Authz-35, p. 36). Unfortunately, it is not clear how the server determines which token is the most recent. For the wall-time clock approach, the token contains an expiration date (draft-ietf-ace-oauth-Authz-35, p. 40). On its own, the expiration date is not sufficient for determining which ticket is the most recent since a newer ticket may have a shorter lifetime. For the token introspection approach it is also not clear how the server identifies the most recent ticket, as they may only contain a lifetime. **The server therefore may use outdated tokens.** The ACE framework should state for every proposed validity option how the order of the tokens is established.

#### 11.2.1.2. Token Introspection

In the ACE framework, the server may use token introspection to obtain additional information for an access token from SAM (draft-ietf-ace-oauth-Authz-35, p. 7). Token introspection must comply with the delegation fundamentals, because SAM provides information in the token introspection response that influences the authorization and therefore is a claim issuer (**claim issuer authorization fundamental**). SAM and S therefore must perform the delegation tasks. A detailed analysis of token introspection is out of scope for this thesis.

S must check if SAM is authorized to be an introspection endpoint for S. According to the framework, SAM must check the server's authorization (draft-ietf-ace-oauth-Authz-35, p. 31), but it does not require S to check that SAM is authorized. **S may thus request introspection from an unauthorized SAM and receive wrong information about the access token.**

Token introspection puts additional strain on the server because S must send and receive extra messages. As described in section 3.1.3, data transmission is particularly expensive for a constrained device. Performing the delegation tasks requires additional effort. If the server must use token introspection to validate newly received access tokens, the risk of DoS attacks therefore considerably increases. Access tokens should be self-descriptive and contain all information that S requires

for the authorization process. The server should avoid using token introspection to validate newly received tokens. Due to our review, the ACE framework now describes the threat in the security considerations (draft-ietf-ace-oauth-authz-17, pp. 39–40; draft-ietf-ace-oauth-authz-35, p. 48)

The server might also use token introspection to validate from time to time if a token is still valid. This might be an alternative to a revocation mechanism for satisfying the **rule completeness fundamental** or **claim completeness fundamental** if the delegation fundamentals are sufficiently satisfied. In particular, S must validate that the token introspection response is fresh and stems from a SAM that is authorized by SOP.

### 11.2.2. Client Side Delegation

Until our review, the ACE framework did not define the relationship between C and SAM, and did not clearly specify how these devices must protect their communication. To provide S with authorization information about C that represent SOP's decisions, SAM must be able to continuously link SOP's authorization rules for C to the credentials that C uses in the secure communication with SAM. **Without this validation, SAM cannot be sure that it is communicating with an authorized C.** Since version 17, the ACE framework defines security requirements for the communication between C and SAM, and demands that C and SAM exchange a secret or their public key (draft-ietf-ace-oauth-authz-17, pp. 37–38; draft-ietf-ace-oauth-authz-35, p. 45).

In the RPK mode, C specifies the RPK in the access token request that it wants SAM to use in the access token (draft-ietf-ace-oauth-params-13, p. 3). Until our review, the ACE framework did not require SAM to check if C actually has this RPK. **Without this check, an attacker may trick SAM into issuing tokens for a different C.** After we pointed this problem out to the working group<sup>2</sup>, this requirement was included in the ACE framework additional parameters draft (draft-ietf-ace-oauth-params-01, p. 3; draft-ietf-ace-oauth-params-13, p. 3).

---

<sup>2</sup>See <https://mailarchive.ietf.org/arch/msg/ace/uEJYGOOTEqpfhTy1pXHO99-Mp1g/>

C must validate the client information, i.e., it must perform the delegation tasks D2 to D8. To perform task D4, C must determine if SAM is authorized by COP. **If the security association between C and SAM is not established and validated correctly, C might accept client information from an unauthorized entity and then communicate with the wrong S.** Figure 11.2 shows the mutual authenticated authorization between C and SAM. Since version 17, the ACE framework demands that C checks if SAM has the authority to issue access tokens for S (draft-ietf-ace-oauth-authz-17, pp. 37–38; draft-ietf-ace-oauth-authz-35, p. 45).

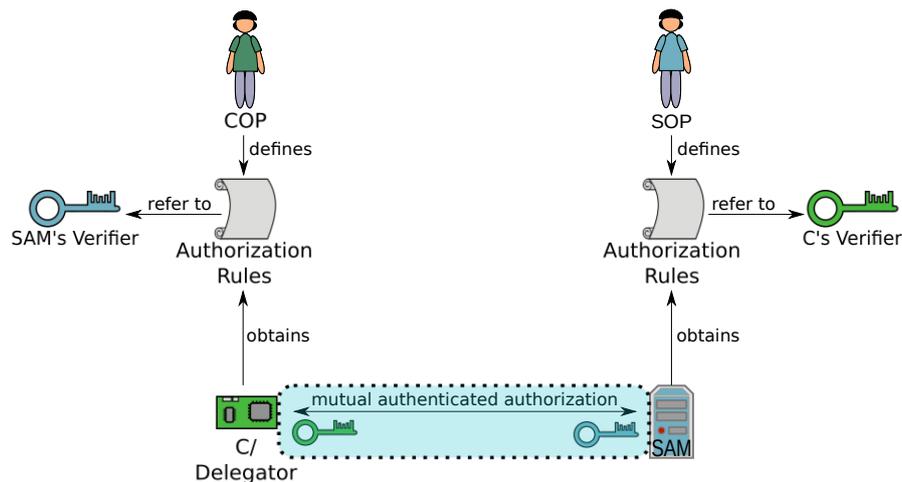


Figure 11.2.: C and SAM Use Their Respective Keying Material For Mutual Authenticated Authorization

Until our review, the ACE framework did not specify that C must validate if the client information is up to date (task D2). The access information for the client could contain an `expires_in` field, which specifies the validity period of the access token in seconds from its generation. But it was not clear if the server's keying material has the same validity period as the access token. Also, the `expires_in` field is optional. Without it, the client information is valid indefinitely. **A client that cannot determine when the client information expires may use outdated (and maybe compromised) keying material to communicate with the server<sup>3</sup>.**

Since version 21, the ACE framework contains a key expiration section (draft-ietf-ace-oauth-authz-21, pp. 39; draft-ietf-ace-oauth-authz-35, pp. 41–42). The section

<sup>3</sup>see also <https://mailarchive.ietf.org/arch/msg/ace/NrQWetugoy0TWp9eg3lwtSictc8>

specifies that the client must assume that the keying material expires with the access token. Additionally, the client must either know a default validity period for the tokens, or SAM must inform C with the help of the `expires_in` parameter about the token's lifetime. Unfortunately, these mechanisms do not really enable C to determine when token and keying material expire, because C does not know when the client information was created. An `iat` (issued at) field is not specified for SAM's response to C.

**At the time of writing, the ACE framework still provides too little detail about how the client determines the validity of the client information.** To improve the proposed mechanisms, C may, e.g., start a timer when it sends the access token request to SAM. When the access token response arrives, C then must check the timer and subtract the elapsed time from the token lifetime. A constrained C that does not measure time during sleep must stay awake until the response arrives. The ACE framework should specify the details of the proposed mechanisms.

The ACE framework does not require SAM to check if the server's keying material actually is valid as long as the access token. SAM must consider the security of symmetric keying material when it specifies the validity period of the access token. If raw public keys are used between C and S, SAM must ascertain that the server's RPK does not expire before the access token. This is especially important if tokens are valid for a very long time. At the time of writing, this problem was not addressed in the ACE framework. **The keying material therefore may be long expired before the access token becomes invalid, and C then uses outdated keying material for the communication with S.**

SAM and C must have a common understanding about the identity of the server with which C wants to communicate. C specifies the server in the token request message with the `audience` parameter which is encoded as a text string (draft-ietf-ace-oauth-authz-35, p. 18). Up to version 19, the ACE framework did not explain how C and SAM identify the server. Example audiences were (and still are) human-readable, e.g., `tempSensor4711` or `tempSensorInLivingRoom`. **If SAM does not understand with which server C wants to communicate, it might provide C with the wrong credentials and access token. C will then communicate with the wrong server without being able to notice the mistake.** The process where SAM provides C with the wrong credentials for S is depicted in figure 11.3.

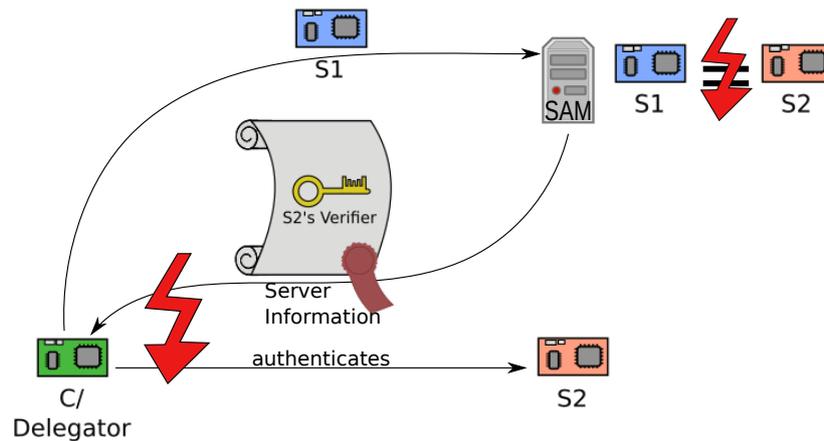


Figure 11.3.: SAM Provides C with Credentials for Wrong S

Since version 20, the authors include a description of the flaw described above in the ACE framework (draft-ietf-ace-oauth-authz-21, pp. 43–44; draft-ietf-ace-oauth-authz-35, p. 48). They state that the client may be configured with the respective audience values or that the client may look them up in a directory. They demand that such a directory would need to provide integrity and ensure the correctness of the data. The relationships and assurances that are required for an effective authorization solution are a bit more complicated than that. **To satisfy the correlation fundamental, C must validate that COP authorized the S with which C communicates.** To achieve this, COP and SAM must have a common understanding how S (or any server with certain attributes) is identified, and COP must provision C with authorization rules for S that refer to the server's identity. C must specify a server that COP allows in the `audience` field of the access token request. SAM must identify the server that the `audience` field describes and ensure that the respective keying material is provided to C in the access token response. If any of these interrelations are not implemented correctly, C may communicate with an unauthorized server. The ACE framework should explain these requirements to provide a secure solution. **At the time of writing, the framework does not even require SAM to ensure that the access token and keying material that it provides to C refer to the server that C requested.** As stated above, this may lead to the client communicating with the wrong server without even realizing it.

The ACE framework introduced an `audience` field for the SAM information

message that indicates to the client how the server is addressed (draft-ietf-ace-oauth-authz-35, p. 17). **Unfortunately, this approach provides no security and may even be harmful. Since the SAM information message is not protected, attackers may change it so that it contains their own server as the audience.** Therefore, C must not rely on the audience that is provided in the AS information.

C must be able to determine that it is the intended receiver of the client information (**claim destination fundamental**). Since the client information does not contain a field to directly specify for which C it is intended, the destination must be indirectly bound to the client information: the access token request and the corresponding client information provided by SAM must first be encrypted and then integrity-protected (see also below), or an AEAD algorithm must be used. In the ACE framework, the communication between C and SAM must be confidentiality-protected (draft-ietf-ace-oauth-authz-35, p. 45), but that the confidentiality-protection must be applied before or together with the integrity-protection is not mentioned. It is not clear how data destination verifiability for C is accomplished in the ACE framework. **Without data destination verifiability, man-in-the-middle attacks are possible: C might be tricked into accepting client information that was intended for a different C and as a result communicate with the wrong server.** While AEAD algorithms are usually used today, it is still necessary to mention this requirement in the ACE framework.

#### 11.2.2.1. Client Registration

The ACE framework expects the clients to be registered with SAM. The ACE framework refers to OAuth 2.0 dynamic client registration (RFC 7591) for the registration process (draft-ietf-ace-oauth-authz-35, p. 25), where a client or client developer requests registration from SAM. For the communication between client and SAM, RFC 7591 suggests the use of TLS (RFC 7591, p. 28). The client must check SAM's certificate (RFC 7591, p. 28). Since constrained clients will often have difficulties to do so, they are not able to perform the registration by themselves. An alternative mechanism is required. **In the worst case, overseeing principals need to manually register their constrained clients with every potential SAM:**

**they must provision the necessary keying material for each SAM and the corresponding client ID to the client.**

In the ACE framework, SAM is supposed to obtain the client's credentials that enable it to authenticate the client during the registration. Claimants that provide information for authenticating C (task **An1**) influence the authorization and thus must be authorized by SOP (**claim issuer authorization fundamental**). In the OAuth 2.0 dynamic client registration, clients specify client metadata in the registration request such as the client's name or its logo. SAM returns the client id and, in some cases, also the client credentials to the client. SAM must treat the client metadata as self-asserted (RFC 7591, pp. 30–31). Rogue clients may impersonate legitimate clients by using, e.g., their name and logo, unless SAM somehow manages to counteract this. It is not clearly defined how SAM is supposed to do so. OAuth 2.0 dynamic client registration may not enable SAM to securely gain credentials that authenticate C.

C and SAM must know each other prior to the authorization process. SAM is required to store the information about C that it obtained during the client registration. Also, C must store information about SAM such as SAM's credentials. C and SAM therefore are closely coupled.

Establishing an initial security association with a less-constrained device is difficult for constrained devices. They often do not have user interfaces and displays that would allow their overseeing principals to directly interact with them. Also, some smart objects might be installed in places where they are difficult to reach, e.g., inside a wall. Network communication may be the only way to interact with these devices. Overseeing principals then must use a less-constrained device to interact with their constrained devices and configure them. To avoid user interaction on the client side, the client registration for constrained clients should involve a less-constrained device on the client side, i.e., CAM.

In the four-corner architecture, C and S are closely coupled to their AMs. Since the potentially constrained device and its manager belong to the same security domain, the close coupling does not prevent separate evolution and communication across organization boundaries (see also section 9.4.1). In the ACE framework, C is closely coupled to SAM which not necessarily belongs to the same

security domain. The required client registration makes communication across organization boundaries difficult and complex which goes against the idea of an open Web. Client registration also is not scalable; the effort of registering every potential client with every potential SAM may be very high in the IoT that consists of billions of nodes. The **Scalability Req** is not met.

#### **11.2.2.2. OAuth 2.0 Device Authorization Grant**

As mentioned in section 11.2.1, the device authorization grant (RFC 8628) was designed for clients that do not have a convenient interface for user interaction. To use this grant type, clients must be able to use HTTP and TLS (RFC 8628, p. 3), which may be difficult for constrained clients. Also, the client must display certain information such as a URI to SOP, which requires certain hardware on the client or an additional communication channel with SOP. For the device flow to work, the client must poll SAM repeatedly until it responds with the access token (RFC 8628, p. 5). Since this behavior puts strain on SAM, use of the device flow is discouraged for autonomous clients.

The device authorization grant was designed to provide SOP's authorization rules to SAM (tasks **A1** to **A5**). It does not help the client to authenticate SAM or S and validate their authorization.

#### **11.2.2.3. Client Side Authorization**

Most of the authorization tasks that C and COP must perform (tasks **A1** to **A5** and task **A7** and **A8**), are out of scope for the ACE framework. Also, it is not clearly specified how C performs the delegation tasks including the validation of SAM's authorization. An important question in IoT scenarios is how the client knows with which server it is supposed to communicate and which resources COP wants C to access on that server. A client must at least perform binary authorization (see section 2.4) if the communication data requires protection.

COP's and C's perspective is mostly omitted from the ACE framework. How COP provides authorization rules to C clearly is out of scope. COP is not involved in the authorization process described by the framework.

**If C does not know which data requires protection, it may disclose sensitive information in the unprotected unauthorized resource request. Unless S rejects the unauthorized resource request, C may even continue to communicate with an unauthorized S, send sensitive data and process data received from S without validating its authorization. C may not even protect the communication, which allows everyone to eavesdrop. COP's and even SOP's security objectives are in jeopardy if C is not enabled to actively participate in their protection by using an authenticated authorization solution.**

For authorization on the client side, we can distinguish scenarios where C is directly controlled by an authorized overseeing principal and scenarios where C is acting autonomously. If COP controls C directly, e.g., because C is located on a laptop or smartphone, COP specifies via C's user interface which resources on which server C accesses. As we showed in the TLS example (see section 7.1), it would thereby be possible for COP and C to perform tasks **A1 to A5, A7 and A8**. COP may then also be able to assist in validating SAM's authorization if additionally a security solution such as TLS or DTLS is used between C and SAM. If implemented correctly, C may then also be able to perform the delegation tasks.

But IoT devices often must communicate autonomously: their overseeing principals may not be present at the time of the communication or it may be inconvenient for them to interfere, and the constrained devices may not have user interfaces (see also section 4.2.3). In these scenarios, COP and C need a different approach for performing the authorization tasks.

As we described in section 3.5.2, clients can autonomously discover services provided by servers using, e.g., multicast CoAP or resource directories. Devices are thus able to find their communication partners themselves which reduces the configuration effort for the overseeing principals. The question is how a client can determine if its overseeing principal wants it to communicate with a certain discovered server.

One possible solution for enabling COP and C to perform the authorization and delegation tasks would be to manually configure on the client with which server and which SAM the client is supposed to communicate. Hard-coding the authentication and authorization data into the client seems like an easy solution,

but COP must ensure that the data on the client is always up to date to comply with the **rule completeness fundamental** and **claim completeness fundamental**. Manual configuration is not scalable (the **Scalability Req** is not met): it is not feasible for scenarios where COP oversees a large number of clients, as can be expected in the IoT. Also, it will be difficult for a constrained client to communicate with a large number of servers, because the client needs to store information for all servers with which it may want to communicate and for their respective authorization managers. But even for scenarios where the client communicates with only a few servers the solution is very inflexible. Each time the server or its SAM changes, the client must be reconfigured, which is especially difficult if it happens on short notice, e.g., because the server broke down. According to Fielding, anarchic scalability, which is one of the requirements for the world wide Web's architecture, comprises that "clients cannot be expected to maintain knowledge of all servers" (Fielding 2000, p. 69). As the ACE framework does not specify an alternative to manual configuration, it currently does not support autonomous clients.

An approach where COP's authorization rules are automatically provided to C would be much more convenient for COP. The strain for the client lessens and the scalability increases if C only needs to store authentication and authorization information for communication partners that it is actually currently communicating with. This can be securely achieved if the process of provisioning the necessary information to the client is integrated into the process of provisioning the authentication and authorization information to the server. We will describe in chapter 12 how DCAF combines client and server provisioning.

### 11.3. Conclusion

The main focus of the ACE framework and the DTLS profile is how S delegates authorization tasks to SAM. The documents explain how SAM generates the access token, how the access token is validated by S, and how S then performs the authorization tasks for C.

A general problem in earlier versions of the ACE framework was that important details such as the requirements for the underlying security solution or the management of the access token were not clearly specified. With our review, we made the ACE working group aware of the existing vulnerabilities. Many of the open issues were addressed after our review. The ACE framework and the DTLS profile therefore will be able to help with the respective tasks.

Some open issues are still not addressed. Important details such as how the server must react to missing fields in the access token, or how the server determines which is the most recent access token are not specified. The burden for filling those gaps is left to the implementers or even the overseeing principals. While it may be possible to generate a secure implementation in these cases, the room for mistakes is unnecessarily large.

The use of token introspection for validating messages may be difficult for constrained devices. The additional messages that must be sent between S and SAM increase the effort for S and heighten the risk of DoS attacks. Also, it is not clear if token introspection complies with the authorization and delegation fundamentals and securely provides S with the necessary information. Where possible, all required information should be provided in the access token, i.e., the token should be self-descriptive.

A pressing concern in the ACE framework is the client support. The ACE framework and the DTLS profile describe how SAM provides the necessary keying material to C for securely communicating with S. If specified correctly, the ACE framework and DTLS profile define how SAM assists C with task A6 concerning S, and how SAM and C perform the respective delegation tasks. But this part has conceptual problems that must be addressed, most prominently the problem how C and SAM get a common understanding of how S is identified. Without it, C may communicate with the wrong server without being able to notice it. Also, C may not know when the client information expires: the lifetime of the keying material alone is not sufficient. The ACE framework offers only very little support for securing the client.

Registering every client with every potential SAM is not a scalable solution. The closer coupling between C and SAM conflicts with the Web's anarchic scalability

requirement and makes communication across organization boundaries difficult. To avoid the necessity for manual registration of constrained nodes, an additional client registration protocol is required which increases the burden on constrained clients.

How autonomous clients are supposed to enforce COP's security objectives is not addressed by the ACE framework. Manual configuration and maintenance of authorization rules on the clients is tedious work for overseeing principals and infeasible for IoT scenarios with large numbers of devices. An additional authorization solution is needed for this purpose which requires extra code and memory and additional message exchanges. The strain on constrained clients increases. Simply ignoring how C participates in the protection is not an option: the client may then expose confidential data and accept data from unauthorized servers. **The overall security of the Internet of Things will suffer if the security of autonomous clients is not properly addressed.**

Summarizing, many important questions concerning the support of constrained devices remain unresolved such as how constrained servers can be protected against DoS attacks, how clients determine the validity period of client information, or how they validate that a communication partner is authorized by COP. The three-corner architecture model of the ACE framework limits the scenarios where the solution can be used. COP's interests are not considered by the framework. In the following chapters we will show that the four-corner architecture is better suited to support autonomous clients and servers and thus enables them to build a true Web of Things.

## 12. Delegated CoAP Authenticated Authorization Framework

In chapter 9 we described the Task Delegation architectural style, where each node that is not able to perform authenticated authorization on its own must have a security association with an authorization manager that helps with the necessary tasks. The most general architecture that implements this style is the four-corner architecture, where each constrained device has its own authorization manager.

If overseeing principals provide authorization rules only to the server, the client cannot participate in the protection of the security objectives. Relying solely on the server for the protection is not a secure solution: C then may send confidential messages to S without protection, which discloses their content to eavesdropping attackers. Also, C may communicate with unauthorized servers and exchange data with them. We therefore postulate that **each endpoint that handles sensitive data must participate in its protection. It must securely be provisioned with the authorization rules that it must enforce.**

The ACE framework (see chapter 11) provides only an authorization manager on the server side and offers only little support for clients. An additional mechanism is required to provide clients with the necessary information for secure communication.

In this chapter we describe the Delegated CoAP Authenticated Authorization Framework (DCAF) that we designed to fully implement the proposed four-corner architecture: C and S may belong to distinct security domains with different overseeing principals, and may each have their own authorization manager.

Thus, not only servers but also autonomous clients are enabled to participate in the protection of their overseeing principals' security objectives. The authorization information that C requires is provided within the protocol flow; no additional messages are needed.

DCAF supports the task delegation architectural style and thus can be adapted to individual scenarios; if a client does not require its own authorization manager, e.g., because it is directly controlled by its overseeing principal, only a single authorization manager may be used.

Constrained devices are not required to have a security association with authorization managers from other security domains, but only with their own AM. The burden on constrained clients is thereby reduced, and the solution offers more scalability than OAuth-based approaches, which require clients to register with the SAM of each server with which they want to communicate. As challenging tasks can be offloaded to the authorization manager, the client's design can be more simple. As a result, less software updates are required. Also, the clients can be deployed in more application scenarios, and be assigned to other scenarios more easily.

As we described in chapter 10, symmetric keys are a good fit for constrained environments, but for some use cases raw public keys may have some advantages. DCAF enables the participating endpoints to use raw public keys and symmetric key cryptography. To perform the delegation and authorization tasks, endpoints not only require the keying material itself (the verifier) but also the corresponding semantic information (the claim that the verifier binds to the holder, see section 5.4.2.1). In DCAF, the access ticket and client information are designed to securely transport all necessary pieces of semantic information to the client and the server. DCAF therefore provides for the secure distribution of keying material and corresponding semantic information for both symmetric and asymmetric solutions.

The DCAF version provided in this thesis is based on our earlier work on the topic (draft-gerdes-ace-dcaf-authorize-04; Gerdes et al. 2014). DCAF was designed before the ACE working group was created and prior to the ACE framework and the DTLS profile.

We introduce important design decisions for DCAF in section 12.1. The normal DCAF protocol flow is described in section 12.2. DCAF implements the Task Delegation architectural style and thus can be applied to various architectures as we will describe in section 12.3. If unforeseen changes in the authorization rules occur, e.g., if an endpoint was compromised, authorization managers might need to react quickly. In section 12.4, we introduce a revocation mechanism for DCAF. Some IoT applications also benefit from using group communication; as DCAF allows for attribute-based authorization, it can be used to distribute the required keying material for these scenarios. In section 12.5, we outline a suitable key distribution mechanism. If clients are not able to obtain an access ticket, e.g., because they cannot reach authorization managers in the Internet, servers may instead request authorization for them. In section 12.6, we will outline a protocol flow for these scenarios and explain why such an approach should be avoided if possible. Section 12.7 discusses how DCAF implements REST design patterns. We developed an open source implementation of our protocol as a proof of concept; an overview of our implementation is given in section 12.8. The findings of this chapter are summarized in section 12.9.

## 12.1. Protocol Design Overview

The effectiveness of an authorization solution for the IoT depends on its ability to fulfill the various requirements that arise from the characteristics of the devices and their application scenarios. While it is possible to define an authorization protocol with a single protocol flow, such an approach does not optimally address every scenario. DCAF therefore allows for multiple protocol flows, some of which we will describe later in the document. The architecture, i.e., the participating actors, as well as the order of the messages exchanged between the actors may be adapted to the scenario. The following section focuses on DCAF's basic protocol flow in the four-corner architecture.

In the basic protocol flow, C obtains an access ticket from SAM with the assistance of its own CAM. The ticket contains the authentication and authorization

information required by S. To obtain the ticket, C must first discover SAM's address, e.g., by sending an unauthorized resource request to S (see also 12.2.1). If the requested resource is protected and S has no authorization information for C, S will decline the request and reply with SAM's address.

After obtaining SAM's address, C contacts its own CAM and securely transmits SAM's address as well as the resources and actions that C wants to perform on S. CAM uses the information sent by C to contact SAM. CAM and SAM mutually authenticate each other and check each other's authorization (mutual authorization). If SAM decides that the client belonging to CAM is allowed to access the resource as requested, it generates an access ticket for C. The ticket comprises two parts: the client information that is meant for the client, and the ticket face meant for the server. The client information contains data that the client requires to authenticate S, in particular the necessary keying material. The ticket face holds the authorization information required by S, including the permissions that are granted to C. SAM protects the ticket face with the keying material it has for S.

The complete ticket is sent to CAM, which may add permissions defined by C's overseeing principal COP to the client information. CAM then sends the ticket to the client. C keeps the client information and sends the ticket face to S. With their respective parts of the ticket, C and S can establish a secure channel and validate each other's authorization. Figure 12.1 shows the normal DCAF protocol flow.

Clients need to determine with which server they are supposed to communicate. In section 12.1.1, we discuss server discovery. DCAF requires that certain security associations already exist. Section 12.1.2 describes the security associations that are a prerequisite for DCAF. This chapter mainly describes DCAF over DTLS, but DCAF can also be used with other security solutions such as object security. The underlying security solution must meet the requirements that are specified in section 12.1.4. DCAF can be used with symmetric or asymmetric cryptographic algorithms. How the keying material must be managed is specified in section 12.1.6. An important problem in constrained environments is how the devices are enabled to validate the freshness and validity of access tokens. Section 12.1.7 introduces validity options to support constrained devices with varied limitations.

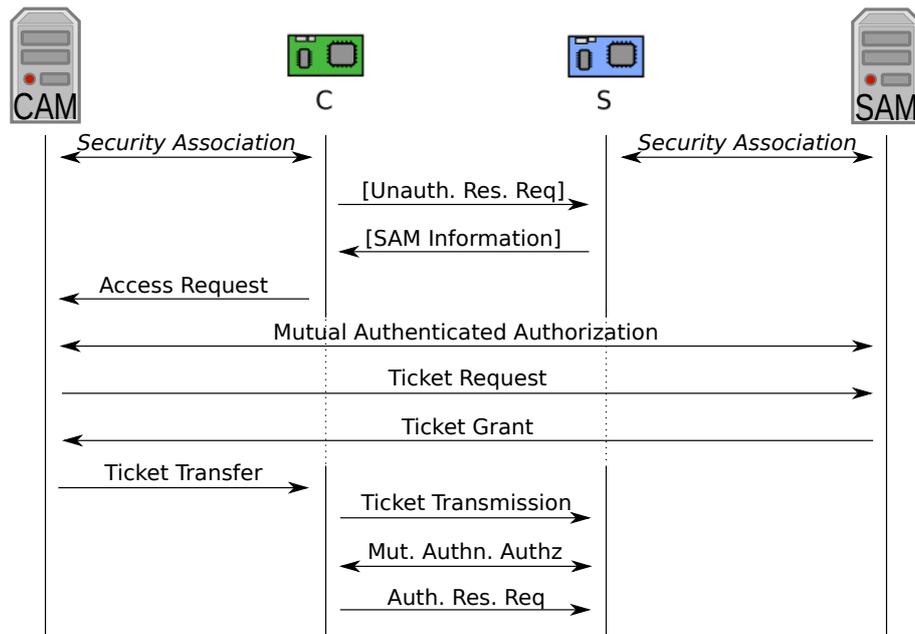


Figure 12.1.: DCAF Protocol Flow

### 12.1.1. Server Discovery

In the big web, overseeing principals directly control their browser to navigate to a certain server (see also section 7.1). In contrast, autonomous clients in the IoT cannot rely on the direct interaction with their overseeing principal to learn with which server they are supposed to communicate. The client is therefore faced with two problems: 1. finding a server that provides a certain desired service and 2. ascertaining that this server is authorized to provide the service to the client.

A seemingly easy solution is for the overseeing principal to configure the client with the necessary information. But this solution has multiple problems: the client may not have convenient user interface, and interacting with it may be slow. Also, overseeing principals may have a large number of clients and manual configuration is not a scalable solution. Additionally, unforeseen changes are difficult to address, i.e., the solution is not dynamic.

CoAP assists constrained clients with discovering servers and their resources (see section 3.5.2). CoAP resource discovery enables clients to find servers that offer resources of a certain type, e.g., the outdoor temperature. Clients may also use

resource directories to discover servers. These mechanisms assist clients with accomplishing the first of the steps above. The information that is provided by these mechanisms often is not secured and may not stem from an authorized claimant. In these cases, it must not be used for authorization (see **claim issuer authorization fundamental**). For some cases a secure server discovery may be necessary where clients can rely on the provided information about the server.

In DCAF, the client authorization manager is able to provide C with all information necessary to validate the server's authorization. C may specify the attributes of the server with which it wants to communicate in its request to CAM and as response receive keying material and—for fine-grained authorization—authorization information for this server. Also, DCAF allows clients to ask their CAM with which server they are supposed to communicate. CAM then specifies all necessary information about the server in the response.

In DCAF, authorization information is therefore provided dynamically when needed. If unforeseen changes occur, e.g., if a server becomes unavailable, the client can discover a new server or ask its CAM to find one.

### 12.1.2. Security Associations

In this section we will describe security relationships that are expected to already exist between certain actors and need to be established outside the DCAF protocol flow. They are necessary to satisfy the **principal fundamental**, as we will discuss in section 13.1. Details about the establishment of security associations between a constrained device and its manager are out of scope for DCAF. We will describe in chapter 14 how security associations can be established and managed during the whole life cycle of a device.

The setup of security associations with previously unknown devices requires some effort. In OAuth 2.0 and the ACE framework, each client must be registered with the server's AM. The registration of constrained devices with a foreign authorization manager requires an additional mechanism. In most cases, human interaction is also necessary (see also section 11.2.2.1). OAuth 2.0 clients

are expected to authenticate SAM using certificates (RFC 6749, p. 58), which puts additional strain on constrained devices: e.g., in TLS and DTLS, the required certificates—with the exception of the root certificate—are transported in the handshake (see section 7.1.2), which may lead to large message sizes. Also, clients require a clock and a time synchronization mechanism to interpret the validity period given in the certificates, which may be difficult for constrained clients (see also section 10.1.2.1). In the ACE framework, clients need to manage security associations for each SAM of the servers with which they want to communicate. In DCAF, constrained devices only require a security association with their own AM. This approach significantly decreases the effort for the constrained devices and their owners.

The setup of the security associations that are described in this section requires that the participating endpoints securely obtain information about their respective peer. In DCAF, *securely obtaining* authentication or authorization information includes that the endpoint must validate the respective information as follows: the endpoint must ascertain that it obtained and evaluated all currently valid information, that the information was provided by claim issuers that are authorized by the endpoint's OVPs, that the endpoint is the intended destination of the information, and that the information applies to the current communication context.

In DCAF, authorization managers act as an authority for their domain: they vouch for the attributes and keying material of the constrained devices for which they are responsible (see figure 12.2). COP and SOP make the authorization decisions for their respective domain. They inform their own AMs which other AMs are authorized. They also decide for which attributes a certain AM may vouch: they may, e.g., only permit the AM to speak for clients that belong to a certain overseeing principal. The required authorization rules and keying material must securely be obtained by the AMs. To enforce their OVPs' decisions, authorization managers are expected to mutually authenticate each other and validate each other's authorization (see also section 12.1.3).

With the help of the attributes, OVPs can define fine-grained authorization rules. A SOP who owns a door lock may, e.g., define permissions for devices which

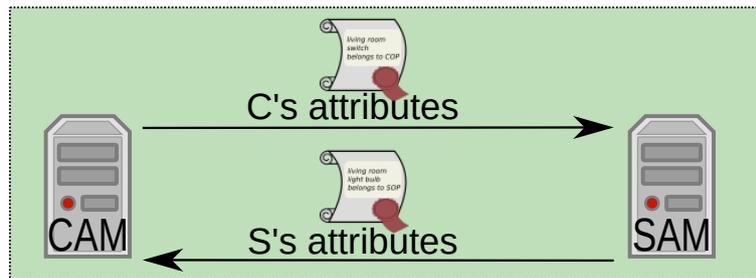


Figure 12.2.: AMs Vouch for Their Constrained Devices' Attributes

are keys and belong to a certain owner. COP and SOP must have a common understanding how attributes are interpreted. SOP must bind the authorization rules to the respective attributes and securely provide them to the respective AM. The AM must validate the OVP's authorization to provide the authorization rules for the respective constrained device. To reduce the burden on the constrained devices, the AM should evaluate the received authorization rules if necessary and provide simplified rules to the constrained device.

Using authenticated attributes for authorization has the advantage that the solution is very flexible: clients can communicate with any server that has the demanded attributes. If a certain server breaks down, the client can search for a new server with the desired attributes and—with the assistance of the AMs—establish a security association with it. Nevertheless it may be necessary for some scenarios to restrict the communication to certain devices. In DCAF, overseeing principals can authorize individual devices by specifying a unique attribute of the device in the permissions, e.g., a device identifier. But, overseeing principal should be aware that attackers can use unique device identifiers to track devices and create behavioral profiles which is a problem for scenarios where privacy is required. By avoiding the use of unique identifiers, the privacy of users can be increased.

The constrained devices and their authorization managers are expected to have an existing security association prior to the communication, and to securely have obtained the required keying material and related semantic information: on the client side, C was securely provisioned by COP with keying material for its CAM. Also, COP securely provided CAM with the keying material for the clients for

which CAM is responsible, and informed CAM about authorization policies for C and C's attributes. On the server side, SOP securely provisioned S with the keying material for its SAM and SAM's address. SAM was provided with the keying material, attributes, and authorization information for each of its servers. Also, in some cases SAM and S must have been provided with a common understanding how S is identified (see also section 12.1.6). The communication between the authorization manager and its constrained device is protected with the obtained keying material. Figure 12.3 shows how COP and SOP provision their devices with the necessary keying material and authorization rules.

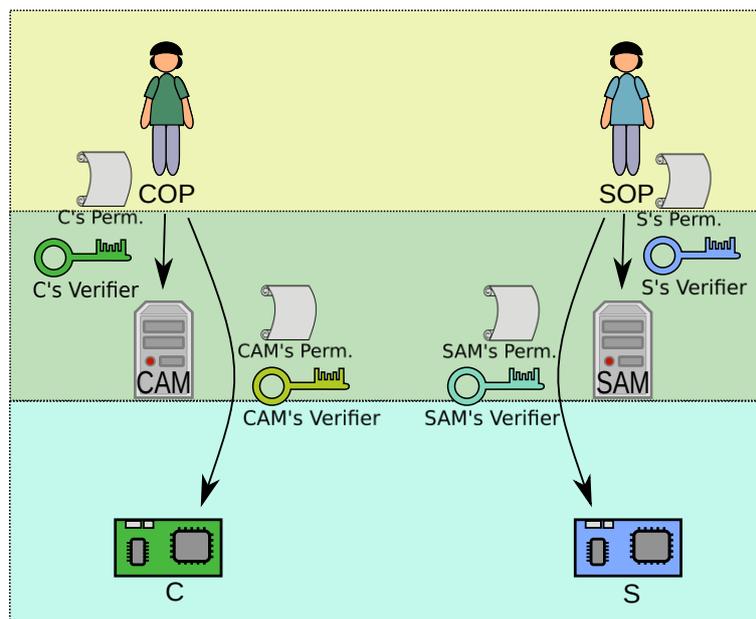


Figure 12.3.: Provisioning

C or CAM must securely obtain the attributes of the server with which C is supposed to communicate from COP, so that they can specify them in the access request message and the ticket request message. If raw public keys are used in the communication between C and S, C may obtain the server's keying material outside of the DCAF protocol flow. The keying material then must securely be obtained together with the server's attributes.

All security associations described in this section must be up to date: the authorization rules and associated keying material must be updated regularly to satisfy the **rule completeness fundamental** and the **claim completeness fundamental**.

Also, the attributes of their devices must be kept up to date on the respective AM. Figure 12.4 shows the security associations that DCAF expects to exist between the actors.

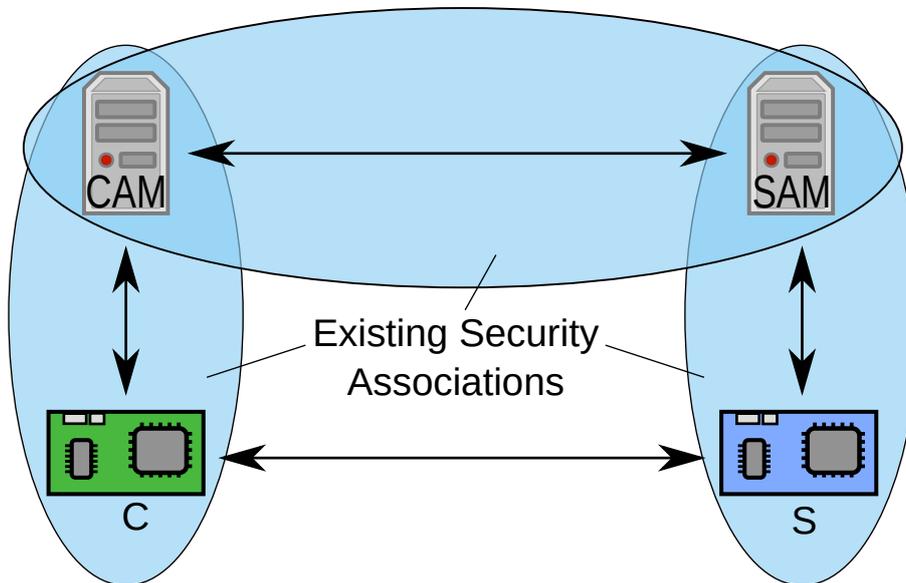


Figure 12.4.: Existing Security Associations

### 12.1.3. Mutual Authenticated Authorization Between CAM and SAM

As described in section 12.1.2, CAM and SAM must be able to mutually authenticate each other and validate each other's authorization. Each AM must securely have obtained authorization rules concerning other AMs from their overseeing principal which must be kept up to date (see figure 12.5).

How the authorization managers get authorization information from their overseeing principals is not specified in DCAF. Since they are less-constrained, they may, e.g., have a Web interface that the overseeing principals can use to configure authorization rules.

An authorization information entry on an AM must contain the following pieces of information which must be bound to each other and endorsed by the OVP:

- the AM that is the holder of the permission,

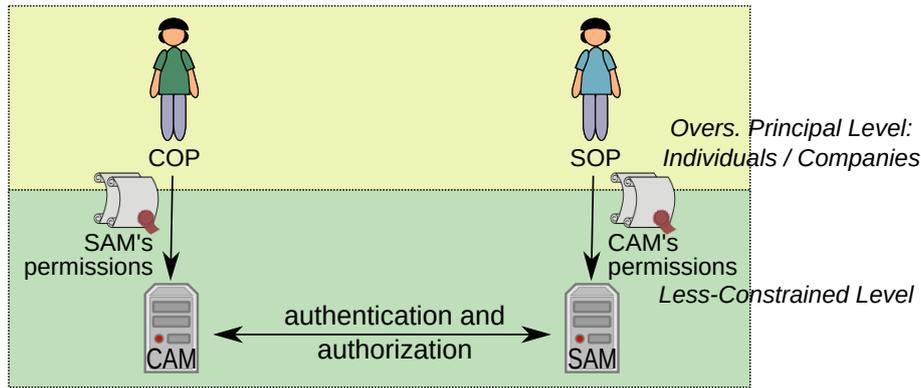


Figure 12.5.: AMs receive authorization rules from their overseeing principals

- the permission, i.e., the resource and the action that the holder and its constrained devices are entitled to use, and the attributes that the constrained devices may have,
- the destination of the permission, i.e., the constrained device that is meant to enforce the authorization,
- the generation time and lifetime of the permission.

If the OVP wants to define fine-grained authorization rules, the entry must also contain the attributes of the holder's constrained devices; the permission is then limited to devices with these attributes.

An example for an authorization information entry is shown below. It informs SAM that clients of the CAM with the hostname `cam.example.com` are allowed to access the `light` resource on a certain server. The access is limited to clients that are located in the living-room and have the identifier 12345.

```
Holder: cam.example.com,
Attributes: location:living-room, id: 12345,
Permission: server.example.com/light, 5,
Destination: server.example.com,
```

The AMs must obtain keying material that identifies the holder of the permission from an authorized claimant. This authentication information needs to be kept up to date. The security solution that is used between CAM and SAM must allow

them to ascertain that they are communicating with the intended holder. They may, e.g., use TLS in certificate mode with client certificates to protect their communication. To validate the relation between peer and permission in this case, the holder of the permission must match the common name in the X.509 certificate. The certificate must be signed by an authorized CA.

In some cases it may additionally be necessary that CAM communicates only with SAMs that are authorized by the server's SOP. CAM then must ascertain that the resource where it is supposed to request the access ticket is hosted by a SAM that is authorized by SOP (see also section 12.2.1), and that this is the SAM that is authorized by COP. If X.509 certificates are used for the authentication, the common name in the certificate must match the holder of COP's permission and the SAM that hosts the resource.

#### 12.1.4. Requirements for the Underlying Security Solution

The messages that are exchanged between C and CAM, and CAM and SAM, respectively, are expected to be protected by an underlying security solution such as DTLS, TLS or object security. To be suitable for the use with DCAF, the security solution must meet the requirements that we list below. Protocols on lower layers than DCAF, e.g., CoAP, may have additional requirements on the security solution that are not discussed here.

As SAM and S do not communicate directly, their communication is secured differently. We will describe in this work how SAM protects the access ticket for S (see section 12.2). This mechanism is independent of the underlying security solution.

The security solution must protect the integrity, authenticity and confidentiality of message contents and DCAF header fields. To satisfy the **correlation fundamental** and **holder relation fundamental**, the solution must offer replay protection.

In some cases, DCAF also relies on the underlying security solution to provide data destination verifiability (see section 5.2.1): the sender defines the intended

receiver of the message by encrypting the data with the receiver's public key or a key that sender and receiver share. In this case, the encryption is part of the claim information and must be bound to the claim for task **A1** and task **D1**. To achieve this, the underlying security solution should use an AEAD mechanism (see also section 10.1.4). Where this is not possible, an algorithm must be used where messages are first encrypted, before message authentication is applied. Otherwise, the **destination fundamental** and the **claim destination fundamental** cannot be satisfied.

Also, the solution must enable the communication partners to determine that a message belongs to the communication context, e.g., that a certain message was a response to a certain previous request. Thereby, claim data that was provided in the request is bound to the response, which is useful to reduce message sizes.

The CoAP Max-Age Option may be used in DCAF to indicate the lifetime of the access ticket including the client information to CAM and C. Endpoints then must be able to detect if an intermediary altered the value of the Max-Age Option.

DCAF is used to securely distribute authorization information and related keying material to C and S. For an effective authorization, the subsequent communication between C and S must be protected by the security solution.

### 12.1.5. Authorization Information

The purpose of DCAF is to enable the authorization managers to provide authorization information and required keying material to constrained devices. The authorization information instructs the endpoint how it must protect its overseeing principals' data. It is therefore important that the endpoint is able to understand the authorization information correctly.

Authorization information may become quite complex, especially if a lot of different rules are required. Endpoints, especially constrained devices, may have difficulties to interpret and evaluate complex rules. In DCAF, the Authorization Information Format (AIF, draft-ietf-ace-aif-00) is recommended for encoding the authorization information. An AIF object consists of a resource identifier in form

of a URI, and the permission set that identifies the actions in form of CoAP methods. We recommend that only one set of permissions per resource is defined in an authorization information.

If the entitlements of a peer do not or only seldom change, it may be useful to provide them once and to only update the authorization to use them. In these cases the access ticket does not need to contain the permission, but must still convey all data that is required to validate the ticket: the intended destination and the holder of the ticket as well as validity information. Additionally, the endpoint must be able to identify the corresponding stored entitlements of this holder. Establishing this relation with only the keying material in the ticket is not always possible or useful. Keying material, especially symmetric keys, need to be exchanged frequently, which would then entail that the authorization information that is stored on the endpoint also needs to be changed. It is therefore more useful to provide additional information about the holder with the access ticket. This would e.g., allow overseeing principals to provide authorization information to a light bulb which states that switches in the living room are allowed to turn it on and off. The access ticket would then need to contain the attributes “switch” and “living room”. DCAF enables the AMs to specify the holder’s attributes in the parts of the access ticket.

If an authorization protocol does not explicitly convey authorization information and does also not specify the intended behavior of the receiver for this case, the security of the solution may be breached. The claim issuer may assume that the delegator already has authorization information for this specific holder, while the receiver may assume that the authenticated holder is supposed to have all possible entitlements (binary authorization, see section 2.4). Authorization solutions can avoid such gaps, e.g., by requiring the authorization information to be present or by specifying the expected behavior of the receiver if information is missing. For DCAF we therefore specify that the ticket face must either contain the permissions of the holder or its attributes or both. The holder of the client information must have the attributes that CAM specified in the ticket request message and may additionally contain the server’s permissions. Tickets are not required to contain client information. How clients perform authenticated authorization for the server in this case is out of scope.

In theory it is possible to omit the holder's keying material for authorization claims: the access ticket then must contain the holder's attributes as the verifier. The recipient of the ticket must obtain or already have keying material for the peer with these attributes. For attribute claims this approach would not work, however: The attributes can either represent the verifier or the statement, but not both. The treatment of these tickets may lead to errors on the recipient. To make the solution less complicated we require that the keying material or at least information about the keying material of the respective holder must be present in both parts of the access ticket.

### **12.1.6. Keying Material**

DCAF recommends the use of symmetric cryptography if constrained devices participate in the communication, which not only concerns the communication between C and S, but may also apply to the communication between C and CAM, and S and SAM, respectively. Since the deployment of asymmetric solutions can be useful in some cases, DCAF is also usable for the distribution of raw public keys. We discuss advantages of symmetric and asymmetric solutions in section 10.1.

If C and S use raw public keys to communicate, each AM must have obtained the RPK of its constrained device, and the authorization rules defined by the OVP must refer to the device with this key. The AM secures the communication with the constrained device using this RPK to ensure that it is actually communicating with the correct device.

Symmetric keys that are used in DCAF between the constrained device and its authorization manager must not be shared with other devices that are not also authorized AMs. Otherwise it would be possible for an unauthorized entity to issue access tickets or client information. The access ticket therefore must be encrypted if it contains a symmetric key. The encryption can also be used to achieve data destination verifiability because it enables S to determine that it is the intended destination of the access ticket.

If C and S use raw public keys for the communication and the access ticket does not contain other sensitive data, SAM does not need to encrypt it. SAM may need to explicitly inform S that it is the intended recipient of the ticket. Therefore, SAM and S must then have a common understanding how S is identified. Both SAM and S must have obtained the attributes that identify S from their overseeing principal.

### 12.1.7. Freshness and Validity

Authorization managers in DCAF are expected to measure time accurately, i.e., they have local clocks which are frequently updated by a time synchronization mechanism. Constrained devices may only have a limited ability to measure and keep time (see also section 3.1.4). Nevertheless, the constrained devices must validate that the information that they get from their authorization managers is up to date to satisfy the **rule completeness fundamental** and the **claim completeness fundamental**. C must check the client information and S must check the ticket face concerning its freshness and validity. They must be able to validate received authentication and authorization information concerning the following aspects:

1. The authentication and authorization information is not a replay.
2. The authentication and authorization information is still valid.
3. If multiple sets of authentication or authorization information were received, which of them is the most recent.

DCAF uses sequence numbers for access tickets and client information (see also section 12.1.7.4). The sequence number enables servers and clients to determine which access ticket and client information is the most recent and enables revocation (see also section 12.4).

In this section, we will describe three validity options. Devices that have wall-clock time can use validity option 1. But not every constrained device has such a reliable clock. The very first DCAF draft therefore already included a rough description of a mechanism where S defines a nonce in the response to an unauthorized resource request which it later uses to validate the ticket (draft-gerdes-core-dcaf-authorize-00, p. 9). The ACE framework later adopted this approach.

A constrained device must at least be able to use option 3 to securely participate in the protection of its overseeing principal's security objectives.

Overseeing principals must inform their AMs how long authentication and authorization information and thus access tickets and client information may at most be valid. SOP and COP should configure a `MAX_LIFETIME` on the AMs for each of their constrained devices. CAM and SAM must set the lifetimes in the ticket face and the client information accordingly. For validity option 3, access tickets should only have a short lifetime because of the device's very limited ability to measure time.

In CoAP, the Max-Age option specifies the maximum of time a response may be cached until it is no longer fresh (see also section 3.5). DCAF uses the Max-Age Option to specify the lifetime of the client information.

The ACE framework defines the `exi` parameter for the lifetime of the ticket. Unfortunately, this parameter specifies the remaining lifetime of the ticket from the time when the server first sees it (draft-ietf-ace-oauth-Authz-35, p. 41). The time between generation of the ticket and receiving it thus cannot be determined. The framework correctly points out that lifetime of tokens can be extended by withholding them. The approach is therefore not usable without additional measures that will likely make the whole mechanism dispensable. As it is not possible to relate the lifetime to a timestamp, the `exi` parameter also cannot safely be used for other purposes. It would be more useful if the ACE framework would follow the specification of the OAuth 2.0 `expires_in` parameter, which describes the lifetime of the access token in seconds from the time of its generation. In this work we use the parameter `expires_in` more generally to specify the lifetime of a claim relative to its generation time.

#### 12.1.7.1. Option 1

The constrained device has an RTC (see also section 3.1.4), and the authorization manager and the constrained device use a time synchronization mechanism. In this case, fields that require wall clock time such as timestamps, expiration time and issuing time can be used by all actors. SAM specifies the ticket lifetime, i.e.,

the length of time that the ticket is supposed to be valid, and a timestamp containing the current time, in the ticket face and the client information. The client information lifetime should equal the lifetime of the ticket face. If necessary, CAM may change the lifetime of the client information to include COP's authorization policies (see section 12.2.6).

If SAM does not provide C with keying material for S, it does not define client information. If COP specified authorization rules for C and S, CAM adds client information including an own timestamp.

Both constrained devices calculate the remaining lifetime (RL) from the current time (CT), and the lifetime (LT) and timestamp (AM\_TS) specified by the AMs, in their respective part of the ticket:

$$RL = LT - (CT - AM\_TS)$$

If a constrained device uses validity option 1 and its part of the ticket does not contain a timestamp, the constrained device discards the ticket.

#### **12.1.7.2. Option 2**

The constrained device has no RTC but counts during sleep, and stores time synchronization points in non-volatile memory to keep the time during reboot. The device thus does not have the actual current time, but continually takes note of the passing of time. Without an RTC, devices suffer from long-term deviation for which overseeing principals may compensate, as we will describe in section 12.1.7.5.

On the server side: the server generates a timestamp that contains the current time—as the server understands it—and a random nonce. The nonce is sent with the SAM information message. S stores the timestamp together with the nonce. SAM reflects the nonce and adds the ticket lifetime to the access ticket. When the server receives the access ticket, it checks the nonce and retrieves the timestamp. If no matching stored nonce is found, S discards the access ticket. Otherwise, S

checks if the ticket is still valid by calculating the remaining lifetime of the ticket using the timestamp ( $S\_TS$ ), the ticket lifetime specified by SAM and the current time as shown below. If the access ticket is no longer valid, S discards it.

$$RL = LT - (CT - S\_TS)$$

If C already knows CAM's address, it may directly contact CAM. Without a fresh nonce in the access ticket, S is not able to determine its validity. In this case, S discards the ticket. SAM should decline ticket requests without a server nonce if it knows that S uses validity option 2 or 3.

The random nonces are used to prevent an attacker, e.g., a malicious client, from tricking SAM to generate an access ticket for a time in the future. S therefore must generate the nonces in a way that an attacker cannot guess future nonces.

S deletes stored nonces and timestamps when it receives an access ticket with the nonce. To save storage space, S should delete unused nonces and timestamps after a certain time, e.g., the expected round-trip time to SAM and back. This behavior provides some protection against attacks where an attacker floods the server with (maybe unauthorized) access requests to drain the server's resources.

On the client side: Before sending the access request, the client stores a client timestamp that contains the time of the access request as the client understands it. C can determine with the assistance of the underlying security solution if a message is the response to a certain request (see section 12.1.4). C therefore does not require a client nonce. SAM provides a lifetime that indicates how long the client information, including the keying material for S, is valid (see also section 12.2.4.1). The client information lifetime should match the lifetime of the ticket face. CAM may change the client information lifetime to include COP's authorization policies, if necessary (see section 12.2.6). After determining that a received response is the answer to a certain request, C uses the current time, the ticket lifetime as specified by SAM or CAM, and the stored client timestamp ( $C\_TS$ ) to calculate the remaining lifetime of the client information (see below). If the client information is no longer valid, the client discards it.

$$RL = LT - (CT - C_{TS})$$

Option 2 enables C and S to determine when the access ticket expires: an attacker who intercepts the communication and withholds a message, e.g., the SAM information message, to reinsert it at a later time, cannot achieve that the ticket expires later than intended.

### 12.1.7.3. Option 3

The constrained device has no RTC and does not count during sleep, but does measure time when it is awake. On the server side: the server includes a random nonce in the SAM information message and sets a short timeout. It stays as awake as necessary to measure time until the timeout is reached. If no access ticket arrives during this time, the server discards the nonce and goes to sleep. If SAM receives a ticket request containing a nonce, it reflects the nonce in the access ticket and adds a lifetime.

When S receives the ticket face, it checks if it has a stored nonce that matches the nonce in the ticket face. S then uses the time that has passed since the timeout was set (PT) and the ticket lifetime to calculate the remaining lifetime of the ticket as described below. If S does not have a matching nonce, the ticket is discarded. If the ticket does not contain a nonce, S also discards the ticket and treats subsequent requests as unauthorized resource requests (see also option 2).

$$RL = LT - PT$$

For this approach, the server must store a nonce and stay awake for a certain amount of time. An attacker might exploit this behavior by continuously sending false access requests to drain the server's resources. If possible, option 1 should be used instead. Shutting off the devices from the Internet by protecting them with a firewall may mitigate these attacks, but this solution is very inflexible and thus not a good solution for the IoT (see also section 4.2.6). As an alternative, a mechanism similar to Secure Wake on Radio Nudging (SWORN, draft-bormann-t2trg-sworn-03) may be useful: the router adjacent to the server only forwards

unauthorized resource requests to the server if the client presents it with a permission granted by SOP.

On the client side, the client sets a short timeout and then sends the access request. C must stay awake until the timeout is reached. If no corresponding ticket transfer message arrives during this time, the timeout is discarded and the client can go to sleep or try again. SAM specifies the lifetime of the ticket in the client information, which may be modified by CAM (see section 12.2.6). If the client receives a ticket transfer message it relies on the underlying security solution to determine that the message is a response to the previously sent access request. C then calculates the remaining lifetime from the ticket lifetime and the time that has passed since the timeout was set (see server side).

If a device does not have the ability to measure time during sleep, it cannot use a lifetime to determine if an access ticket is expired. For those cases, the ticket might be used only while the device is awake, and discarded afterwards. If this is not feasible and the device has a defined duty-cycle, the device may count the sleep phases it went through and discard the access ticket after a certain number of them.

#### **12.1.7.4. Sequence Numbers**

Sequence numbers are used in DCAF for three purposes: for determining the most recent access ticket, for updating a ticket, and to enable revocation. These mechanisms are important to achieve rule and claim completeness: If an endpoint cannot determine the most recent access ticket, it may use deprecated tickets and keying material. Without revocation, endpoints may communicate with compromised endpoints or use compromised keying material. We will describe a revocation mechanism in section 12.4.

The sequence numbers are specified by the responsible AMs: SAM specifies the sequence number for the ticket face, CAM that for the client information. Each AM is expected to have an own number range for each of its constrained devices to reduce the risk of overflows. Usually, the number range starts with 0. Sequence numbers must be unique for this AM its constrained device, i.e., if two clients

request access tickets from the same SAM, the tickets will never have the same sequence number.

S must check the sequence number in the ticket face to ascertain that it uses only the most recent information (see section 12.2.8). The client does not require the sequence number to determine the most recent client information, but instead may use the generation time (for validity option 1), a stored timestamp (for validity option 2) or the passed time since the access request was sent (for validity option 3) to determine the most recent ticket.

For the management of access tickets, endpoints must consider that each AM defines own sequence numbers. E.g., a server that receives an access ticket must check if it already has an access ticket with this or a more recent sequence number from this AM. If the security association between the constrained device and its AM is altered or terminated, i.e., if the AM's keying material changes, the constrained device must discard all tickets that were issued by this AM, maybe after a transition phase (see also section 14.3.2).

If many tickets are issued, overflows may still occur eventually. To deal with them, constrained devices must notice if they reach the upper bounds of the sequence number range. They then remove all tickets and terminate all communications. If the responsible AM reaches the upper bounds, it should revoke all access tickets for this server before resetting the number range.

For validity option 1, it is possible to use the issuing time instead of the sequence number. Both allow an endpoint to determine which ticket is the most recent. To realize revocation with the issuing time, the AM must ascertain that it only issues a single access ticket per second.

When using options 2 or 3, C and S require the sequence number to determine the most recently created access ticket. The timestamps and nonces that they created themselves are not suitable for that purpose.

CAM informs SAM in the ticket request about the sequence number of the client information. If SAM wants to receive revocation information concerning C from CAM, it must store CAM's sequence number. SAM provides the sequence number of the ticket face to CAM in the ticket grant message.

### 12.1.7.5. Long-Term Deviation

In addition to not having the actual current time, clocks on constrained devices suffer from long-term deviation (see also section 3.1.4): over time, the discrepancy between the time that actually passed, and the time that the constrained device thinks has passed, increases. Frequent updates by time synchronization mechanisms enable devices to keep the correct time. But without a time synchronization mechanism, the long-term deviation may become a problem. For the validity options two and three it may therefore be necessary to counterbalance the effects of deviation. Below, we introduce a simple compensation mechanism that can be used in DCAF. Its goal is to prevent that access tickets have a longer validity period than intended.

Manufacturers usually specify accuracy parameters such as the frequency stability of their clock generators. Overseeing principals may store the current accuracy of the constrained devices on their authorization managers. The AM uses this value to calculate the current deviation and adapts the lifetimes accordingly. If, e.g., the clock on the constrained device deviates up to 10 s per day, the lifetime is reduced by this amount per day. To determine the remaining lifetime (RL), the deviation ( $d$ ) is subtracted from the `MAX_LIFETIME` (ML) (see below).

$$RL = ML - d$$

Constrained devices need to be able to communicate for a certain amount of time to fulfill their purpose. The deviation mechanism assumes the constrained device to be behind the actual time by the maximum deviation. If the clock generator instead is too fast, access tickets expire too soon, which is made worse by the mechanism. In the worst case, the constrained devices may no longer be able to function. The overseeing principals therefore should configure a `MIN_LIFETIME` on the AMs for each of their constrained devices, that defines the minimum lifetime of an access ticket or client information. The lifetime of an access ticket should not fall below the `MIN_LIFETIME`, i.e.,  $MIN\_LIFETIME \geq (MAX\_LIFETIME - 2 \times \text{deviation})$  should hold. **If the lifetime that remains after subtracting the maximal deviation ( $2 \times \text{deviation}$ ) from `MAX_LIFETIME` is smaller than**

**MIN\_LIFETIME**, the constrained device may not have the necessary ability to securely process the ticket.

The AMs subtract the deviation only for the constrained devices for which they know the accuracy. In the normal DCAF protocol flow, SAM should consider the deviation for the ticket face, but not for the client information. In this case, the lifetimes differ for the parts of the ticket. CAM may adapt the lifetime of the client information to consider the client's expected deviation.

#### 12.1.7.6. Summary

All validity options presented above help to achieve the **rule completeness fundamental** and the **claim completeness fundamental**. Endpoints can thereby determine if and for how long claims that they received are still valid. Attackers are not able to extend the lifetime of a claim over the intended period, e.g., by withholding messages.

Validity options 2 and 3 offer solutions for constrained devices that only have a limited ability to measure time. Thus, the devices do not require an RTC to communicate securely. If long-time deviation is considered for the definition of ticket lifetimes, its impact on the authenticated authorization can be mitigated.

Storing the necessary nonces and staying awake means more effort for the devices. If necessary, the risk of DoS attacks may be mitigated by additional measures, e.g., by deploying an additional protection mechanism on routers adjacent to the devices.

## 12.2. Protocol Flow

We will now describe the normal protocol flow, where both the client and the server are constrained devices. Since this protocol flow is not always applicable, variations are described in section 12.3. Also, messages not necessarily need to be exchanged in the order that is given below; e.g., CAM may request an access ticket from SAM before C asks for it.

The focus of the description is on the security of the protocol: we aim at explaining the necessary steps for achieving an effective authenticated authorization protocol. We perform an evaluation of DCAF in chapter 13. It includes an authorization fundamental evaluation where we applied our model to the protocol. To minimize redundant explanations we avoid references to the model in the protocol description.

We use the CBOR Web Token (CWT, RFC 8392) format to describe the required parameters. Where necessary, we extend it to transport the required information. A summary of the parameters that are used in DCAF is given in Appendix D. The CoAP content format option specifies which payload format is used. DCAF allows the constrained devices to choose which content format they use. In this thesis, we describe the content type with the media type `application/dcaf+cbor`. This media type is not yet registered.

The messages that are exchanged for the protocol flow are described in the sections 12.2.1 to 12.2.12. DCAF defines a mechanism for deriving symmetric keys from the access ticket, which is described in section 12.2.13. Section 12.1.3 gives an overview of the authenticated authorization process between CAM and SAM.

### 12.2.1. SAM Discovery

As stated above, clients require SAM's address to request an access ticket. Clients may obtain this information from S with an unauthorized request message: if S receives a request that is not covered by authorization information, e.g., because S did not yet receive an access ticket for C, S declines the request and provides SAM's address with the response in the SAM information message. If S uses validity option 2 or 3, it additionally adds a nonce to the message (see also section 12.1.7). At this point, C and S do not yet have a security association. C therefore must not send confidential data in the unauthorized resource request.

In the DCAF payload format, SAM's address is identified by the label `sam`. The ACE framework specifies the parameter `cnonce` for the nonce generated by the server to validate the freshness of the access token. Although naming the parameter `cnonce` (client nonce) is very misleading as the client has nothing to do with

the nonce, we use it here for compatibility reasons. An overview of the DCAF data structure and the used identifiers can be found in Appendix D. The message below shows an example for a response to an unauthorized request.

```
4.01 Unauthorized
  Content-Format: application/dcaf+cbor
  {
    sam: "coaps://sam.example.com/authorize",
    cnonce: h'50a93002fa9e71'
  }
```

A minimal SAM information message that only contains the SAM address takes up 2 bytes of fixed data to which the additional bytes for the string must be added. The example message above that also contains a server nonce has 46 bytes.

In some cases it may be difficult for C to obtain SAM's address, e.g., because of its limited system resources. CAM therefore may obtain SAM's address for C.

There may be application scenarios where SOP requires the confidentiality of ticket request messages. SOP may not want others to know the requested resources and actions, or the characteristics of the client, e.g., its type of device. In these cases, C and CAM must securely be informed where they must request access tickets; the respective SAM and its resource to which requests must be sent must be provided by a claim issuer that is authorized by SOP, e.g., a protected resource directory. How this is accomplished is out of scope for DCAF. SOP should ensure that in this case the server does not provide SAM's address, e.g., by omitting it from the SAM information message.

#### **12.2.1.1. AM Information Resource**

If C attempts to contact S over an unprotected channel, it must not send confidential data to S. If COP or SOP require the confidentiality of the request data, C must not reveal which resource it attempts to access. To solve this problem, we define a special SAM information resource. The server responds to requests to this resource always with a SAM information message. Providing a SAM information resource helps protecting the overseeing principals' privacy.

C may discover the name of the SAM information resource using the server's `.well-known/core` resource (see also section 3.5.2). The CoRE link format allows clients to discover resources with certain resource types using the `rt` attribute (RFC 6690, p. 11). For this purpose, we define the resource type `am-info`.

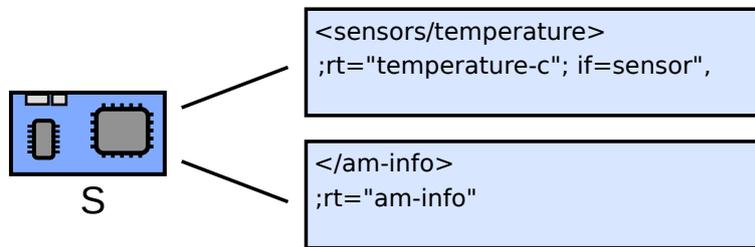


Figure 12.6.: AM-Info Resource on the Server

C discovers the SAM info resource by sending a `GET` request to `.well-known/core` and searching the response for the resource with this resource type. If the server supports filtering, C can also directly search for a resource with this type:

```
GET /.well-known/core?rt=am-info
```

#### 12.2.1.2. AM Address in Resource Descriptions

If S uses validity option 2 or 3, the server requires the nonce that it specifies in the SAM information message to determine the validity of the ticket. C therefore should send a request to S to obtain a SAM information message containing the nonce. But if S uses validity option 1, it may provide the address of the responsible AM in the `.well-known/core` resource description. To achieve this, we need to create a link between the server's resources and the authorization manager that is responsible for granting access to them. The semantics of a link are described with the help of relation types (RFC 8288, p. 6). The relation type that we require for specifying the authorization manager of a resource does not yet exist; we therefore define the new relation type `am-authorize`, which is not yet registered.

The example below shows part of a response to a `.well-known/core` request; the first entry describes a temperature resource. The second entry is linked to the

first with the `anchor` parameter. The parameter `rel` specifies the relation type (see also RFC 8288, p. 11). The entry thus describes the URI of the authorization resource on SAM where the client can request access tickets for the temperature resource.

```
</sensors/temp>
;rt="temperature-c";if="sensor",
<coaps://sam.example.com/sensors/authorize>
;anchor="coap://temp451.example.com/sensors/temp"
;rel="am-authorize",
```

A constrained server will often only have a single authorization manager that is responsible for all resources. Instead of referring to a specific resource, the anchor of the `am-authorize` link may contain the base URI (RFC 3986, p. 28) of the server's resources. The example below shows the address of the SAM that is responsible for `temp451.example.com`.

```
</sensors/temp>
;rt="temperature-c";if="sensor",
<coaps://sam.example.com/sensors/authorize>
;anchor="coap://temp451.example.com/"
;rel="am-authorize",
```

C might also look up the address of the responsible SAM in a resource directory (draft-ietf-core-resource-directory-25) (see also section 3.5.2). In this case, S must be registered with the resource directory. The response to a resource directory lookup is similar to the `.well-known/core` responses described above, but if no `anchor` attribute was specified for a resource, the base URI that was registered for the resource is used as the anchor (draft-ietf-core-resource-directory-25, p. 35). Also, targets and anchors are fully resolved:

```
<coaps://temp451.example.com/sensors/temp>
;rt="temperature-c";if="sensor",
;anchor="coaps://temp451.example.com/",
<coaps://sam.example.com/sensors/authorize>
;anchor="coaps://temp451.example.com/"
;rel="am-authorize",
```

If the RD entry is forged, C may obtain a false address for SAM. In DCAF, CAM must authenticate SAM and validate its authorization (see section 12.1.2). C and CAM therefore cannot be tricked by a false SAM address to communicate with the wrong SAM. Nevertheless, forged RD entries may make it difficult for C to discover SAM's correct address. To avoid forged RD entries, the registration of resources with the RD should be protected.

Resource Directories may contain confidential information. To protect the overseeing principals' privacy, RD entries should only be presented to clients that are authorized to access them.

### 12.2.2. Access Request Message

To initiate the process of obtaining an access ticket, C sends an access request to CAM. C and CAM are expected to already have a security association at this point (see also section 12.1.2) and can communicate securely, e.g., using DTLS. C must protect the confidentiality and integrity of the access request message. To protect the confidentiality, C must use the keying material of an authorized CAM.

As described above, CAM may obtain SAM's address for C. In these cases, C may omit SAM's address from the message. C may define the resources it wants to access on S and the actions that it wants to perform on these resources. In DCAF, C should use an AIF object for this purpose. In AIF, the relative URI of the desired resource can be encoded, but the format does not use the authority part of a URI that identifies the server (draft-ietf-ace-aif-00, p. 4). The base URI (RFC 3986, pp. 28–29) therefore needs to be additionally specified in DCAF. C may include it in the access request message. C may also define additional attributes that further identify the server. If C communicates with multiple servers, it should specify the attributes in the access request message. The URI and the attributes enable CAM to find the server with which C wants to communicate. In some use cases it may be more useful if CAM specifies the server with which C is supposed to communicate. If C omits the attributes, CAM must define them in the ticket request message for SAM.

The access request contains the following information:

- the absolute URI (see also RFC 3986) that represents SAM's address,
- the server's attributes, including the authority component of the server's URI,
- the resources that C wants to access,
- the actions that C wants to perform on the resource,
- the server nonce, if specified by S in the SAM information message.

In the CWT format, SAM's address is identified by the label `sam`. The server is specified with the `audience` parameter. The resource and actions are represented by the `scope` parameter which may contain an AIF object. The server nonce has the label `cnonce`.

An example message is shown below. The scope contains an AIF object with the requested actions GET and PUT, encoded as 5. S uses validity option 2.

```
POST client-authorize
Content-Format: application/dcaf+cbor
{
  sam: "coaps://sam.example.com/authorize",
  audience: ["coaps://temp451.example.com"],
  scope: ["/s/tempC", 5],
  cnonce: h'50a93002fa9e71'
}
```

When C uses validity option 3, it starts a timeout when it sends the access request message. C must be able to determine which timeout applies to the communication (see also section 12.2.7).

A minimal access request message is empty. The client thereby leaves the discovery of the server and its authorization manager to CAM. An access request containing SAM's URI and the server's attributes has 5 bytes of fixed data. For the strings that are contained in the message, additional space is required. The content of the example message listed above has 90 bytes.

If the access request message is rejected with the CoAP response code 4.00 and the error description `nonce_required` (see also section 12.2.4), the client should send an unauthorized resource request to the server to obtain a SAM information message that contains a nonce.

### 12.2.3. Ticket Request Message

When CAM receives an access request message from C, it must check if SAM is authorized by COP to vouch for the server's attributes and issue keying material for the communication between C and S. If CAM does not yet have authorization information for SAM, it must now obtain it (see also section 12.1.2). How this is achieved is not part of DCAF.

CAM must also ensure that it is responsible for C: CAM must validate that the authorization rules defined by COP actually refer to the C with which it communicates, e.g., by validating that C is able to use certain credentials. Also, if C and S are supposed to use the RawPublicKey mode for secure communication, CAM must validate that the C that sent the access request actually is the holder of a certain RPK, e.g., by also using the RawPublicKey mode with C. CAM must check if the requested permissions in the access request message are covered by the authorization policies defined by COP for this C, SAM and S (see also section 12.1.3). If none of the requested actions on S are allowed, CAM declines the request.

If SAM is authorized and COP allows at least some of the actions, CAM can provide C with an access ticket. CoAP allows the caching of responses (see also section 3.5). If CAM already requested an access ticket for C from SAM earlier, CAM might have cached the response. If the cached response is still valid, CAM may send it to C.

If no cached response is available, CAM creates a new ticket request message to SAM. SAM's address is taken from the access request message. As payload, CAM uses the payload from the access request message including the request data, SAM's address, the server's attributes and any nonces. CAM adds the server's attributes if they were not specified by C in the access request. If COP defined the server's attributes in the authorization rules (see section 12.1.3), the attributes in the ticket request message must match these attributes. The ticket request message must contain the server's attributes.

If C did not specify resources and actions, CAM may add them to the ticket request. If COP allowed only some of the requested resources and actions, CAM

may change them to reduce the message size. For the enforcement of COP's authorization decisions this is not necessary, since CAM conveys COP's authorization rules to C in the ticket transfer message (see section 12.2.6).

CAM may additionally provide a description of the client's characteristics in the ticket request message. It may, e.g., convey the information that C is a climate control system (HVAC). The vocabulary of the description is not part of DCAF. If raw public keys are used for the communication between C and S, CAM specifies C's RPK in the ticket request. COP's authorization rules must refer to the C with this RPK. If CAM provides C's attributes or C's RPK in the ticket request, it must ascertain that they are still valid. It then must define a timestamp that states the creation time of the ticket request, and a lifetime for the provided information.

As described in section 12.1.7.4, CAM specifies sequence numbers for the client information. If CAM provides a revocation service, it should include the sequence number for the client information in the ticket request message.

The example below shows a ticket request. Validity option 3 is used on the server side, so the message contains a server nonce. The `sub` (subject) parameter (RFC 8392, p. 6) characterizes C as an HVAC device. CAM specifies a timestamp with the current time in the `CWT iat` (issued at) parameter (RFC 8392, p. 6). OAuth 2.0 defines the parameter `expires_in`, that contains the lifetime of an access token in seconds from its generation. Following this approach, we use this parameter to convey the lifetime of claims from the time of their generation. The `expires_in` parameter must not be confused with the `exi` parameter (see section 12.1.7). For the sequence number we use the parameter `cti` (RFC 8392, pp. 6–7).

```
POST coaps://sam.example.com/authorize
Content-Format: application/dcaf+cbor
{
  audience: "coaps://temp451.example.com",
  scope: ["/s/tempC", 5],
  sub: "HVAC",
  cnonce: h'50a93002fa9e71',
  iat: 1591608331,
  expires_in: 604800,
  cti: 12,
}
```

The CBOR representation of the message contents looks as follows (see also Appendix D):

```
{
  /audience/ 5 : "coaps://temp451.example.com",
  /scope/ 9 : ["/s/tempC", 5],
  /sub/ 2 : "HVAC",
  /cnonce/ 39 : h'50a93002fa9e71',
  /iat/ 6: 1591608331,
  /expires_in/ 2 : 604800,
  /cti/ 7 : 12
}
```

If raw public keys are used, the RPK is specified in the `req_cnf` parameter. This matches how clients specify their public key in the ACE framework (see draft-ietf-ace-oauth-authz-35, p. 23). The `req_cnf` parameter contains a `COSE_Key` structure with the key that is later used as proof-of-possession key (see example below).

```
POST coaps://sam.example.com/authorize
Content-Format: application/dcaf+cbor
{
  audience: "coaps://temp451.example.com",
  scope: ["/s/tempC", 5],
  cnonce: h'50a93002fa9e71',
  sub: "HVAC",
  iat: 1591608331,
  expires_in: 604800,
  cti: 12,
  req_cnf: {
    COSE_Key: {
      kty: EC2,
      crv: P-256,
      x: h'D055EE14084D6E0615599DB5839...',
      y: h'B418B64AFE8030DA1DDCF4F42E2...'
    }
  }
}
```

A minimal ticket request message containing only the audience has 3 bytes of fixed data, to which the value of the audience must be added. The example request without keying material above has 67 bytes, the example request containing *C*'s public key has 145 bytes.

The ticket request message must be securely transmitted to SAM; its confidentiality and integrity must be protected as described in section 12.1.4. CAM and SAM must mutually authenticate and authorize each other before CAM transmits the message. Details about this process can be found in section 12.1.3.

If the ticket request message is rejected with the CoAP response code 4.00 and the error description `nonce_required`, CAM replies to the access request with the same response code.

SAM must check if the ticket request message stems from an authorized CAM. If the ticket request contains *C*'s RPK or attributes, SAM must make sure that the request is fresh by checking the timestamp and lifetime that CAM specified. If CAM provided *C*'s attributes, SAM must ascertain that CAM is authorized to vouch for clients with these attributes (see also section 12.1.2).

SAM must store CAM's sequence number if it wants to be informed by CAM if information about the client is revoked.

#### **12.2.4. Access Ticket Generation**

Before SAM can generate the access ticket, it must have obtained authorization rules from SOP for *S*, CAM and *C* as described in section 12.1.2 and 12.1.3. SAM must validate that the keying material and the attributes that it has for *C* and *S* are still valid. Also, SAM must ascertain that it has the most recent authorization rules. SAM must specify the lifetime of the ticket so that it does not exceed the lifetime of the keying material and the authorization rules. SAM must only grant permissions that are covered by SOP's authorization rules for this *S*, CAM and *C* (see also section 12.1.3). The ticket request must be declined in the following cases:

- SOP has not provided any authorization information for CAM on S,
- SAM has authorization rules, but they do not cover the requested permissions,
- SOP defined fine-grained authorization rules but the ticket request message from CAM did not contain the required attributes for C.

In the following section, we will describe the DCAF access ticket. The ticket comprises the ticket face, which is meant for the server, and the client information, which is meant for the client. If SAM does not provide the server's keying material to the client, the client information may be omitted.

#### 12.2.4.1. Ticket Face

CAM specified the attributes of the desired server in the ticket request message. SAM must generate the ticket face for the server that has these attributes. The ticket face comprises the information that S requires to perform authenticated authorization concerning C. It therefore must contain:

- SOP's authorization information for S concerning C, or C's attributes,
- the keying material that is the verifier for C, or the key derivation information,
- a nonce or timestamp that enables S to validate that the authorization information and the keying material for C is fresh.
- the lifetime of the ticket,
- the sequence number of the ticket or a timestamp.

Additionally, the face may also contain:

- the intended recipient of the ticket (if it is not defined otherwise, e.g., by using a certain encryption key),
- the sequence number of the ticket that is replaced by this ticket.

SAM must include either authorization information concerning C or C's characteristics in the ticket face. If SOP provided detailed authorization rules in terms

of allowed actions and resources concerning this S, C and CAM, SAM specifies these rules in the ticket face. If CAM specified C's attributes in the ticket request message, SAM uses them to select the respective permissions. E.g., if CAM stated that C is an HVAC device, SAM only includes permissions in the ticket face that concern CAM's HVAC clients. To simplify the processing of the authorization rules for S, SAM should include only one set of permissions per resource.

If SAM does not have detailed authorization rules for C's access to S or if it is certain that S already has authorization rules for C, it may provide C's attributes in the ticket face instead of authorization information (see also section 12.1.5). In this case, DCAF is on the server side only used for key distribution.

The ticket face must contain information that enables the server to validate the freshness and validity of the ticket (see also section 12.1.7). SOP may have informed SAM which validity option S uses during the setup of the security association (see section 12.1.2). If SAM has no information about the server's validity option and it receives a ticket request that does not contain a server nonce, it assumes that S uses validity option 1. In this case, it adds a timestamp with the current time to the ticket face. If SAM receives a ticket face without a server nonce but knows that S uses validity option 2 or 3, SAM should reject the ticket request with the CoAP response code 4.00 (Bad Request). We define the error description `nonce_required` for this purpose. If the ticket request contains a server nonce (validity option 2 or 3), SAM reflects it.

SAM specifies the intended validity period of the ticket (`MAX_LIFETIME`) as the ticket lifetime. If SAM knows the server's accuracy, SAM may subtract the expected deviation from `MAX_LIFETIME` (see section 12.1.7.5).

The sequence number of the ticket face is specified by SAM. It is required to identify the most recent access ticket for the validity options 2 and 3 (see section 12.1.7.4). If the sequence number is not explicitly specified in the ticket face, it defaults to the value of the timestamp that states the creation time, i.e., the sequence number may be omitted for validity option 1. The ticket face must contain either a sequence number or a timestamp.

If a previous ticket for the communication between C and S exists which is not

yet expired, SAM may specify the sequence number or timestamp of the ticket that is replaced by the access ticket (see also section 12.2.11).

The ticket face contains either keying material or key derivation information to specify the holder of the ticket. Symmetric keys are confidential and should not be transmitted if this is not necessary. If a ticket face contains a symmetric key, the key itself or the whole ticket face must be encrypted. If C and S use the RawPublicKey mode to communicate, SAM must specify C's RPK in the ticket face. SAM must securely have received C's RPK from CAM.

To avoid the transmission of symmetric keying material, SAM and S can use a key derivation function to derive the symmetric key from the ticket face (see also section 12.2.13). To do so, SAM and S must share a secret, e.g., a symmetric key. SAM should specify in the ticket face which key derivation function is used, unless S and SAM already have a common understanding about it. SAM must only provide either a key or a key derivation function.

SAM must protect the integrity of the whole ticket face and enable S to validate that the ticket face stems from SAM. If the ticket contains sensitive information, e.g., the symmetric key or confidential authorization information, the confidentiality of the ticket face must also be protected, and SAM must ensure that it sends the ticket to the correct server. SAM does not directly communicate with S, and no transport layer security solution can be used to protect the ticket face.

If the key is derived from the information in the ticket face, no additional integrity protection is necessary: the correct key can only be calculated with the correct face and the key derivation key that S shares with SAM. In this case, S validates the integrity of the ticket face by using the symmetric key to communicate with C.

Without key derivation, an object security mechanism must be used. The preferred method in DCAF is to transport the ticket face in a COSE object. COSE defines how encryption, signing or MAC objects can be generated for a certain content (see also section 10.1.5). MAC or signing objects protect the integrity of the ticket face. If the ticket contains confidential data, e.g., a symmetric key, it must be encrypted using an encryption object. For the encryption, SAM must use the symmetric key it shares with S or S's RPK.

SAM must bind the ticket face to the intended recipient. Thereby, attacks are avoided where S is tricked into accepting a ticket face that SAM issued for a different S. If the ticket is encrypted, the integrity-protection must be applied afterwards, or an AEAD algorithm must be used. If a key derivation mechanism is used, the server may determine if it is the intended recipient by using the keying material it shares with SAM. If the ticket contains C's public key and is not encrypted, SAM must explicitly state the intended recipient in the ticket face. In this case, SAM and S must have a common understanding how S is identified. They may, e.g., have established this knowledge during the setup of the security association (see also section 12.1.2).

An example for a ticket face is given below. The requested resources and actions are encoded in the `scope` parameter, and the nonce defined by S in the `cnonce` parameter. For validity option 1, SAM specifies a timestamp with the current time in the `iat` (issued at) parameter. The sequence number is defined with the `cti` parameter. The `cnf` parameter (RFC 8747, pp. 5–6) contains information about the client's keying material, in this case a `kid` since the server uses key derivation to calculate the keying material for the communication with the client. We define the `dseq` parameter for the sequence number of the deprecated ticket. The client's attributes may be encoded in the `desc` parameter.

```
ticket-face:
{
  iat: 1591608328,
  scope: ["/s/tempC", 5],
  expires_in: 3600,
  cti: 42,
  cnf: {
    COSE_Key : {
      kty : 4,
      kid: h'e7509a8c032f3bc2a8df1df476f8ef03'
    }
  }
}
```

The CBOR notation of this ticket looks as follows:

```
{
  /ticket_face/ 1: {
    /iat/ 6: 1591608328,
    /scope/ 9: ["/s/tempC", 5],
    /expires_in/ 17: 3600,
    /cti/ 7 : 42,
    /cnf/ 8: {
      /COSE_Key/ 1: {
        /kid/ 2: h'e7509a8c032f3bc2a8df1df476f8ef03',
        /kty/ 1 : 4
      }
    }
  }
}
```

A minimal access ticket with a generation time, a lifetime, a scope, and a cnf claim with a kid and key type (`kty`) has 13 bytes of fixed data, to which the values of the generation time, lifetime, scope and kid must be added. The access ticket depicted above has 52 bytes.

The example below shows a ticket face as a COSE object that is encrypted with Advanced Encryption Standard (AES) in the Counter with CBC-MAC (CCM) block cipher mode. By specifying a key identifier in the `kid` parameter, SAM may hint to S which keying material was used to encrypt the message (RFC 8152, p. 13).

```
ticket_face: {
  cose_encrypt0: [
    protected: { alg: AES-CCM-16-64-128 }
    unprotected: { kid: "key1" }
    ciphertext: h'2e75eeae01b831e0b65c2976e06d90f4
      82135bec5efef3be3d31520b2fa8c6fb
      f572f817203bf7a0940bb6183697567c
      e291b03e9fca5e9cbdfa7e560322d4ed
      3a659f44a542e55331a1a9f43d7f',
  ]
}
```

To determine the size of an encrypted ticket face, 9 bytes for the cose wrapper must be added to the size of the ciphertext. The variable kid size and the size of the encrypted access ticket further increase the size of the ticket face. The ticket face depicted above has 95 bytes.

#### **12.2.4.2. Client Information**

SAM should provide C with the information that enables C to authenticate S. CAM may later add additional data to the client information to provide C with COP's authorization decisions (see section 12.2.6). SAM may omit the client information if raw public keys are used for the communication between S and C if SAM knows that C already has keying material for S.

If present, the client information must provide information about the server that is identified by the attributes that CAM specified in the ticket request message. In particular, the keying material that SAM provides in the client information must correspond to this server.

If the client information conveys keying material to C for the communication with S, it must contain:

- the server's keying material.
- a timestamp defined by SAM,

The client information may optionally also contain:

- the lifetime of the ticket,
- the sequence number of the ticket,
- the authorization information defined by SOP (informational),

SAM specifies the server's keying material in the client information. If symmetric keying material is used for the communication between C and S, SAM generates the symmetric key. If C and S use raw public keys, SAM specifies S's RPK. SAM must define a timestamp with the current time in the client information to inform CAM when the client information was issued.

SAM must set the lifetime that the client information is supposed to have. It should correspond to the lifetime specified in the ticket face. The CoAP Max-Age Option must not exceed the lifetime. Instead of explicitly stating the lifetime in the client information, SAM may use the Max-Age Option. Max-Age then needs to be protected accordingly (see also section 12.1.4).

SAM may specify its sequence number for the ticket in the client information to allow for revocation. The sequence number in the client information should equal the sequence number in the ticket face. If SAM does not explicitly state a sequence number, the timestamp must uniquely identify the ticket.

It may be useful for the client to know which permissions SOP defined for S to avoid unnecessary requests. SAM may therefore specify SOP's authorization rules in the client information. These rules are solely informational and may be modified or removed by CAM.

Like access information in the ACE framework, DCAF client information provides the server's public key in the `rs_cnf` parameter if asymmetric cryptography is used between C and S (draft-ietf-ace-oauth-params-13, p. 7). For symmetric communication, the parameter `cnf` conveys the symmetric key that C and S are supposed to share. SOP's permissions are specified with the `scope` parameter.

```
{
  rs_cnf: {
    COSE_Key: {
      kty: Symmetric,
      kid: h'e7509a8c032f3bc2a8df1df476f8ef03',
      k: h'48ae5a81b87241d81618f56cab0b65e
        c441202f81faabbe10075b20cb57fa939',
    }
  }
  cti: 89,
  iat: 1591608332,
}
```

A minimal client information structure specified by SAM has 11 bytes of fixed data to which the variable data for the timestamp and the server's keying material must be added. The client information depicted above has 53 bytes of data.

### 12.2.5. Ticket Grant Message

After SAM generated the ticket face and the client information, it must securely transfer them to CAM (see also section 12.1.4). SAM must ascertain that it sends the ticket grant message to the CAM that sent the ticket request.

An example for a ticket grant message is depicted below. The Max-Age Option is set to 86400 seconds, which indicates to CAM that the client information is valid for a day. CAM may cache the ticket during this time, but must ensure that its confidentiality is protected. The client information contains a symmetric key that is specified in the `cnf` parameter, the sequence number of the ticket in the `cti` parameter, and the timestamp defined by SAM in the `iat` parameter.

2.05 Content

Content-Format: application/dcaf+cbor

Max-Age: 86400

```
{
  ticket_face: {
    scope: [ "s/tempC", 5 ],
    cti: 42,
    cnonce: h'50a93002fa9e71',
    expires_in: 3600,
    cnf: {
      COSE_Key: {
        kty: Symmetric,
        kid: h'e7509a8c032f3bc2a8df1df476f8ef03',
      }
    }
  },
  cti: 89,
  iat: 1591608332,
  cnf: {
    COSE_Key: {
      kty: Symmetric,
      kid: h'e7509a8c032f3bc2a8df1df476f8ef03,
      k: h'48ae5a81b87241d81618f56cab0b65ec
        441202f81faabbe10075b20cb57fa939',
    }
  }
}
```

CAM must only accept ticket grant messages that are protected by a communication security solution, and only from the authorized SAM to which the access request message was sent. CAM must reject client information that does not contain the required parameters as specified in section 12.2.4.2.

### 12.2.6. Ticket Transfer Message

CAM may assist C in authenticating S and validating its authorization. If COP specified fine-grained authorization rules, CAM provides them to C. In this case, the client information may contain authentication information defined by SAM and authorization information defined by CAM. If no keying material and no authorization information for S needs to be provided to C, the client information may be omitted.

When CAM receives a ticket grant message that contains client information, it must check if the client information is still valid. With the help of SAM's timestamp, CAM determines the time that has passed since SAM issued the client information. CAM then uses the lifetime given in the client information or, if it is missing, Max-Age to determine the remaining lifetime of the ticket.

```
passed_time = (current time - SAM_TS)
SAM_R_Lifetime = lifetime - passed_time
```

CAM must not change the ticket face since it represents information from SAM for S. As the ticket face must be integrity-protected by SAM, S would notice changes in the ticket face and the access ticket would become invalid.

If COP specified fine-grained authorization rules in terms of allowed actions and resources for this C, SAM and S (see also section 12.1.3), CAM must add them to the client information. If SAM provided authorization information in the ticket grant message, CAM should determine the minimum of COP's and SOP's permissions and provide them in the client information. CAM should only include one set of permissions per server resource.

The ticket must provide information about the server with the attributes that C requested. If C did not specify attributes, CAM must specify client information that inform the client with which server it is supposed to communicate. In this client information, CAM may provide the server's attributes. If the client information contains a symmetric key, CAM should specify either the server's attributes or the server's permissions in the client information or both. Otherwise, the client can only use binary authorization with this server since it has no possibility to retrieve fine-grained permissions for this server.

The ticket transfer message must contain a lifetime which is the minimum of the authorization information lifetime and the authentication information lifetime. To determine the client information lifetime, CAM must calculate the minimum of the CAM lifetime and the remaining SAM lifetime. To set the lifetime in relation to the creation time defined by SAM, the time that has passed since the generation of the ticket is then added to the resulting lifetime. CAM may state the lifetime explicitly or encode it in the Max-Age Option, which then needs to be protected accordingly (see also section 12.1.4). Max-Age must not exceed the ticket lifetime.

```
C-info lifetime = min(SAM_R_Lifetime, CAM_Lifetime)
                  + passed_time
```

If CAM knows the client's expected deviation, CAM should subtract it from the lifetime as described in section 12.1.7.5. If the lifetime then is below or equal to zero, the client information is no longer valid. CAM must discard the message and may send another ticket request message to SAM.

If SAM did not specify any client information, CAM must specify its own timestamp in the client information that reflects the current time, unless CAM is certain that C uses validity option 2 or 3<sup>1</sup>.

CAM may specify the sequence number in the client information. It should match the sequence number that CAM send to SAM in the access ticket request.

If the client information contains keying material or authorization information concerning S, it must contain the following information:

---

<sup>1</sup>If the client uses validity option 2 or 3, it is not able to interpret CAM's timestamp.

- the server's keying material,
- for validity option 1: a timestamp defined by SAM or CAM.

The client information may additionally contain:

- the lifetime of the ticket (otherwise encoded in Max-Age),
- COP's authorization information concerning S,
- the server's attributes,
- the sequence number of the client information.

CAM may explicitly state the calculated lifetime in the client information, or instead use the Max-Age Option. Max-Age must not exceed the lifetime of the client information. CAM then sets the ticket face and the client information as the payload for the ticket transfer message.

CAM must protect the integrity and confidentiality of the ticket transfer message, e.g., using DTLS (see also section 12.1.4). To protect the confidentiality, the keying material must be used that identifies the C that is the holder of the client information. CAM must send the client information to the C that sent the access request message.

An example for a ticket transfer message in the DCAF format is depicted below. CAM added client authorization information and adapted the lifetime in Max-Age. For this example we assume that the lifetime defined by COP is longer than the lifetime specified by SAM. CWTs do not have a parameter for the client authorization information. We define the parameter `cscope` (client scope) for this purpose. The timestamp is specified in the `iat` parameter.

#### 2.05 Content

Content-Format: application/dcaf+cbor

Max-Age: 86100

```
{
  ticket_face: {
    scope: [ "s/tempC", 5 ],
    cti: 42,
    cnonce: h'50a93002fa9e71',
    expires_in: 3600,
    cnf: {
```

```
    COSE_Key: {
      kty: Symmetric,
      kid: h'e7509a8c032f3bc2a8df1df476f8ef03,
    }
  },
  cscope: [ "s/tempC, 1 ],
  iat: 1591608332,
  cnf: {
    COSE_Key: {
      kty: Symmetric,
      kid: h'e7509a8c032f3bc2a8df1df476f8ef03,
      k: h'48ae5a81b87241d81618f56cab0b65ec
        441202f81faabbe10075b20cb57fa939',
    }
  }
}
```

A minimal client information consisting only of the server's symmetric keying material is depicted below in CBOR notation.

```
/rs_cnf/ 41: {
  /COSE_Key/ 1: {
    /kty/ 1: /symmetric/ 4,
    /kid/ 2: h'e7509a8c032f3bc2a8df1df476f8ef03',
    /k/ -1: h'48ae5a81b87241d81618f56cab0b65ec'
  }
}
```

The size of the ticket transfer message is determined by the size of the ticket face and the client information. A minimal ticket transfer message consists only of the ticket face (if the client information can be omitted). Minimal client information provided by CAM consists only of the symmetric keying material, if the lifetime is specified in Max-Age, and then has 8 bytes of fixed data to which the size of the keying material must be added. A minimal ticket transfer message with client information then has 21 bytes.

### 12.2.7. Transmitting the Access Ticket

C must only accept ticket transfer messages that are integrity- and confidentiality-protected, and stem from authenticated and authorized CAMs. C relies on the underlying security solution to ensure that the message is the response to the previously sent access request message.

If the ticket transfer message contains client information, C must check if the client information is valid. The client information must contain the required parameters as described in section 12.2.6. Otherwise the message is discarded. If the client information does not contain authorization information and C is not able to determine the server's authorization, C must discard the ticket.

C must determine if the client information is still valid. We describe C's behavior for the various validity options in section 12.1.7. If the client information is no longer valid, C must discard the ticket. If the ticket with this sequence number was revoked, C must also discard the ticket. C may have requested a new access ticket for a server with certain attributes before the old one became invalid, e.g., to assure uninterrupted communication. How C behaves in this case is described in section 12.2.11).

C keeps the client information and transmits the ticket face to S. The transmission can be conducted in various ways. For the PreSharedKey mode, DCAF recommends that C transmits the ticket face in the DTLS handshake. For other approaches, e.g., if raw public keys or object security must be used, a different method is necessary. For these cases, S needs to provide a special resource to which C can post the access ticket.

C must delete the client information when it expires or when C receives a revocation message for this client information (see also section 12.4). If a DTLS connection exists for the client information, the connection must then be terminated.

#### 12.2.7.1. DTLS Handshake

In DCAF, the preferred way to transmit the ticket face from C to S is to use the DTLS handshake. If TLS and DTLS are used with pre-shared keys, the client can

specify a PSK identity in the ClientKeyExchange message of the handshake to indicate to the server which key is to be used (RFC 4279, p. 5). The ticket face either provides the key itself or the information how the key is derived. Therefore, the client may use the PSK identity to transmit the ticket face. C uses the key that SAM provided in the client information as the pre-shared key (PSK) to construct the premaster secret as described in RFC 4279 (RFC 4279, p. 5).

The PSK identity parameter is not confidentiality-protected; information transmitted in this parameter is transported in the clear. If confidential information is transmitted in the ticket face, it must be protected by other means (see also section 12.2.4.1).

S must validate the ticket face as we will describe in section 12.2.8. If the face is not valid, S does not finish the DTLS handshake and discards the ticket face. If C and S can finish the handshake, C and S have proven that they can use certain keying material. The attributes and authorization information in the ticket face and client information then refer to the C and S, respectively. After the handshake, C and S use DTLS to communicate securely (see also section 12.2.10). The DTLS connection must be terminated if the ticket expires, or if the ticket is revoked (see also section 12.4). The ticket must then be discarded.

Sending the ticket face in the handshake has the advantage that the server can discard access tickets immediately if the handshake fails. Also, DTLS provides protection against DoS attacks (RFC 6347, pp. 15–16).

In DTLS 1.3, the identity in the pre-shared key extension is used to transmit the access ticket. TLS 1.3 defines this extension for negotiating the identity of the pre-shared key in the handshake (RFC 8446, p. 55). The identity is part of the ClientHello message and enables the client to provide an identity to the server. The pre-shared key extension can also be used in DTLS 1.3 (draft-ietf-tls-dtls13-38, p. 26).

#### **12.2.7.2. Access Ticket Resource**

The server can provide a special resource for receiving access tickets. If this is the only means for C to transfer an access ticket, S is not able to validate C's autho-

rization before the resource access. Since every entity is able to submit tickets to the access ticket resource, this approach is particularly vulnerable to DoS attacks.

We define a new resource type for the access ticket resource and call it `dcaf-ticket`. `C` discovers the access ticket resource by sending a `GET` request to `.well-known/core` and searching the response for the resource with this resource type. As an alternative, `C` can search directly for a resource with this type if the server supports filtering:

```
GET /.well-known/core?rt=dcaf-ticket
```

After `C` discovered the resource, `C` sends a `POST` request to this resource to transmit the ticket face. The server must protect the access ticket resource: it must only allow `POST` requests to this resource. Other requests must be declined.

When `S` receives a ticket face on the access ticket resource, it must check if the ticket is new and still valid as we will explain in section 12.2.8. If the ticket is new and valid, `S` creates a new resource and sends a `2.01 Created` response to `C`, together with the path to the new resource. If the ticket replaces an old ticket, `S` may use the path of the old resource for the new resource. `C` may store the path of the new resource and use it to transmit updates to the access ticket, but `C` may also use the default access ticket resource.

If the ticket is transmitted in the DTLS handshake, `C` does not know where `S` stores the access tickets, and there is no reason for `C` to obtain this knowledge.

`S` should discard the tickets in the ticket resource as soon as they become invalid. This is especially important if `S` has only very limited storage space. Also, the tickets must be removed immediately if they are revoked by SAM (see also section 12.4). If a DTLS connection exists that is based on the access ticket, the connection must be terminated when the ticket is deleted. To mitigate the risk of DoS attacks, the server should also remove tickets if they have not been used for a certain period of time.

### 12.2.8. Validating the Ticket Face

If S already has a ticket with the same sequence number or, if it is missing, the same timestamp from the same SAM, S discards the new ticket and uses the existing ticket instead. Thus, unnecessary effort for validating the ticket can be avoided.

S then should use the ticket's sequence number to determine if SAM revoked the new ticket (see also section 12.4). If the ticket was revoked, S must discard the ticket.

S must validate that the ticket face contains all information that S requires for the authenticated authorization. The parameters that the ticket face must contain are described in section 12.2.4.1. If the ticket is not confidentiality-protected, it must contain a destination<sup>2</sup>.

Key IDs cannot be expected to be unique (RFC 8152, p. 13) and should therefore not be used to retrieve keying material for the holder of the ticket. The server therefore must discard tickets that only contain a `kid`, unless a key derivation mechanism is used (see below). If a symmetric key was transported in a ticket face that was not encrypted, the keying material is compromised. S must discard the ticket and not use the keying material to communicate.

S must only accept ticket faces that are integrity-protected and that stem from an authorized SAM. S can validate the integrity and authenticity of the ticket by one of the following means:

- S uses the key derivation function that is specified in the ticket face to generate the symmetric key for the communication with C.
- S checks the signature that SAM provided with the ticket face.
- S checks the MAC that SAM provided with the ticket face.

If SAM specified key derivation information, e.g., a key ID, in the `COSE_Key` structure in the ticket face, S uses key derivation to validate the integrity of the ticket face and calculate the symmetric key as described in section 12.2.13. S must

---

<sup>2</sup>If the ticket is encrypted, the intended receiver is thereby implicitly defined.

validate that it shares the key that is used for the key derivation with an authorized SAM. If the ticket contains a COSE signature object instead of a key derivation function, S must check the signature and validate that it was made with a key that identifies the signer to be an authorized SAM. If the ticket contains a COSE MAC object, S must check the MAC and validate that it was made with a symmetric key that S shares with an authorized SAM.

After the integrity and authenticity of the ticket is validated, S must validate that the ticket face is fresh and determine the remaining lifetime of the ticket. How this is accomplished depends on the server's validity option (see section 12.1.7).

S must ascertain that it is the intended recipient of the ticket. If the ticket face is not encrypted, and no key derivation is used, S must ascertain that the intended recipient is explicitly defined in the ticket face and set to S. If S is not the intended recipient or if S cannot determine that it is the intended recipient, it must discard the ticket.

The ticket face must contain either authorization information or C's attributes. Otherwise, S must discard the ticket. If the ticket does not contain authorization rules and S is not able to determine C's authorization with the help of the attributes given in the ticket, S may wait a short time for the corresponding authorization information to arrive but must then discard the ticket.

If the ticket is valid and meant for S, S checks if SAM specified the sequence number of a ticket that is replaced by this ticket. If SAM issued the ticket that is to be replaced, S removes this ticket.

### 12.2.9. Security Association Setup between C and S

To establish the security association between C and S, the endpoints must use their respective keying material. On the client side, if C received client information, it uses the keying material provided therein as a verifier. If RPKs are used, C may have obtained the server's RPK from COP outside the DCAF protocol flow. If C has no verifier for S, it cannot establish the security association. On the server

side, S must validate that C uses the key that was provided either directly in the ticket face or was derived from it using key derivation.

If C and S use DTLS, they authenticate each other by finishing the DTLS handshake. For an object security solution, C and S authenticate each other by encrypting messages for the sender and validating that received messages actually stem from the communication partner by checking the signature or MAC of the message.

### **12.2.10. Authorized Communication**

Before sending each request, C must validate that the requested action and resource is covered by the authorization information that is directly or indirectly specified by the client information, and that the client information is still valid. Before processing the request and sending the response, S must validate that the requested action is allowed on the requested resource according to the authorization information that is specified in or relates to the ticket face, and ensure that the ticket face is not expired.

Constrained nodes may only have very limited storage space. They should therefore only keep access tickets they currently require. C should remove access tickets if the respective communication is finished. If a transport security solution is used between C and S, it may be useful for S to remove the access ticket a short time after the connection was terminated. For an object security solution, access tickets should be deleted if they have not been used for a certain period of time. Clients and servers that use validity option 3 may not be able to measure time while sleeping. In this case, tickets should be deleted before the device goes to sleep.

If access to resources is restricted to authorized peers, clients must send a signature or MAC with their requests that must be validated by the servers. If reading access to the resource is only allowed for authorized entities, both the client and the server also must ensure that at least messages concerning this resource are encrypted, e.g., servers must encrypt responses to GET requests, clients must encrypt PUT and POST requests.

If a client must only receive information from authorized servers, it must check the integrity of responses that contain a resource representation, and make sure that they stem from an authorized S. If the application requires C to ascertain that a certain server has received and acted upon a requests, C must also check the integrity and authenticity of responses.

In some cases it may be necessary that the server can determine that it is the intended destination of a request. Client and server then need to use a mechanism that provides data destination verifiability (see also section 12.1.4).

### 12.2.11. Updating the Access Ticket

If the access ticket expires, C can no longer access resources on S. To avoid communication gaps and potential overhead for the establishment of a new security association, C may request a new ticket before the old ticket is expired.

For validity options 2 and 3, S must provide information in the SAM information message that it can later use to check the validity of the ticket face. C should ask S for new validity information unless C knows that S uses validity option 1. To do so, C sends a request to the AM-info resource, to to which S will respond with a SAM information message (see also section 12.2.1.1). C then uses the usual protocol flow to request a ticket from SAM.

If SAM issues a new ticket for a connection for which it already previously provided a ticket, SAM should specify the sequence number of the old ticket in the ticket face. If C receives a new access ticket, it tries to update the security association with S as described below. If the update was successful, the old client information is replaced.

The update procedures for DTLS with or without key reuse are described below. If an object security solution is used between C and S, the usual protocol flow is used with one exception: if a security association between C and S still exists, C should use it to protect the ticket during the transfer to the access ticket resource on S. S handles the ticket as described in section 12.2.7.2.

**12.2.11.1. DTLS without Key Reuse**

In DTLS, the secure channel is bound to the used keying material. C and S can only authenticate each other if the correct keys are used in the connection. If SAM changes the verifier in the access ticket or the client information, the DTLS channel must be replaced. In this case, the old DTLS connection must be terminated and a new DTLS session must be established.

If C and S use DTLS in the RawPublicKey mode, C or S may get a new RPK at some point. The authorization managers must validate that their constrained devices actually currently have a certain RPK (see also section 12.1.2). How this is accomplished is out of scope for DCAF. If the PreSharedKey mode is used between C and S, SAM may decide to issue a new verifier for the new ticket.

**12.2.11.2. DTLS with Key Reuse**

Symmetric and asymmetric keying material must be changed regularly, but may have a longer lifetime than access tickets. Overseeing principals must assess the risk of attacks and the potential damage for their data and devices, and then decide how long the keying material may be valid.

If the keying material for C and S does not change, the constrained devices may continue to use the existing DTLS channel. In the RawPublicKey mode, the usual protocol flow is used, but C should use the existing DTLS channel to transfer the ticket to the access ticket resource.

If symmetric keys are used in the ticket, SAM might decide to reuse the symmetric key. SAM should only reuse symmetric keying material if the new ticket will likely reach C and S before their DTLS session is terminated. If SAM reuses the keying material, it should not specify the key itself in the ticket face and the client information, but only its identifier. This has the advantage that the symmetric key does not have to be transmitted again. C and S can only use the ticket if the old ticket still exists. Keys can therefore by design not be reused long after the old ticket expired. The disadvantage is that C must request a new ticket if C and S do not receive the updated ticket in time.

As always, SAM must integrity-protect the ticket face. If the ticket face is not encrypted, it must contain the intended destination in the `aud` parameter. An example for an updating access ticket is given below:

```
{
  ticket_face: {
    aud: "server.example.com",
    scope: [ "s/tempC", 5 ],
    cti: 43,
    dseq: 42,
    cnonce: h'51a7d012g5zk53',
    expires_in: 3600,
    cnf: {
      kid: h'e7509a8c032f3bc2a8df1df476f8ef03
    }
  },
  cti: 89,
  iat: 1591608632,
  cnf: {
    kty: Symmetric,
    kid: h'e7509a8c032f3bc2a8df1df476f8ef03
  }
}
```

C requests update for a certain server from CAM. If the response from CAM contains client information with only a Key ID, C must check if the old client information for this server still exists. If this is the case, C compares the `kid` value of the `cnf` or `rs_cnf` structure in the old client information with the `kid` in the ticket transfer message. The server's attributes that C specifies in the update request together with the `kid` identify the keying material of the server. Therefore, C discards the new ticket if the kids do not match. Otherwise, C uses the access ticket resource to transmit the ticket face to S (see section 12.2.7.2) and discards the old ticket. C should protect the transfer using the security association between C and S, e.g., the DTLS channel.

If S receives a ticket face with only a `kid` in the `COSE_Key` parameter of `cnf`, it must validate the ticket as described in section 12.2.8. S keeps the new ticket only if the ticket that is to be replaced still exists, i.e., if S has a ticket with a sequence number that matches the deprecated ticket sequence number in the new ticket.

The sequence number is unique for tickets of this SAM for this S (see section 12.1.7.4) and is therefore suitable to identify the access ticket. S then compares the kid value of the old ticket's `cnf` parameter with the kid value given in the new ticket. If they do not match, the new ticket is discarded. Otherwise, the old ticket is discarded.

### 12.2.12. Resource Registration

CoAP allows clients to request the creation of new resources on servers (RFC 7252, p. 47). If the permissions for newly generated resources are not managed correctly, access to resources may falsely be granted or denied, i.e., the integrity, confidentiality or availability of the resources may be breached.

The newly created resources may inherit ownership and authorization rules of their parent resources, or default authorization rules may be configured for these resources. With the AIF extended model SAM may specify the permissions for newly generated resources (Bormann 2020, p. 5).

The approaches above require the server to manage the authorization rules for resources; since SAM does not know which resources are created, it cannot explicitly provide (or revoke) permissions for them. The permissions that are managed by the server must expire at some point, e.g., together with the client's permission for the parent resource, but then the server must store the relationships between resources. Also, it is not clear if or how permissions can afterwards be regained. Managing these permissions by itself may become too difficult for a constrained server. The complexity may be reduced if only a small number of newly generated resources are allowed and if resources are only short-lived.

Where these approaches are not sufficient, SAM must be informed about newly created resources. In this case, a mechanism is required for securely registering new resources with the respective AM. S may, e.g., contact its SAM after a new resource was generated and provide it with the resource's characteristics. The details of the resource registration process are out of scope for DCAF and for this thesis, and should be addressed by future work.

### 12.2.13. Key Derivation

SAM and S can use a key derivation function to generate the symmetric key that is used between C and S. Using key derivation to determine the symmetric key has the advantage that the confidential symmetric key does not have to be transmitted to S. Also, no signatures or MACs are required, which makes the size of the ticket face even smaller.

To use key derivation, SAM and S must have a common secret, i.e., a shared symmetric key. This key is then used to derive the session key for the communication between C and S from the ticket face. If SAM and S have a default key derivation algorithm, the ticket face may only provide the `kid` in the `COSE_Key` structure. Without a default algorithm, SAM must specify the key derivation algorithm in the ticket face. SAM and S may use an HMAC-based extract-and-expand key derivation function (HKDF, RFC 5869), e.g., HKDF SHA-256 (RFC 8152, p. 53). The keying material shared between S and SAM and the ticket face are used as input for the key derivation function (see also draft-ietf-ace-dtls-authorize-13, pp. 13–14).

SAM may provide a hint which key S should use in the key derivation function. In this case, SAM specifies the `kid` of the shared key in the `cnf` structure (see below). When S receives a ticket face, S must validate that the specified derivation key belongs to an authorized SAM (see also section 12.2.8).

```
cnf: {
  kid : h'79678322e91cd5b5568783e9dc0ab85f',
  COSE_Key : {
    kty : 4,
    kid : h'e7509a8c032f3bc2a8df1df476f8ef03'
  }
}
```

## 12.3. Alternative Architectures

The architecture that derives from the task delegation architectural style depends on the characteristics of the application scenario, such as the architecture level of

the actors and the security domains of the overseeing principals (see also section 9.3.2). The four-corner model with two authorization managers is the most general and can be used in every scenario. It needs to be used in scenarios with two autonomous devices of distinct security domains that require the assistance of an authorization manager.

In specific scenarios, a single authorization manager may be used: if the devices belong to the same security domain, or if one of the devices does not require an authorization manager, e.g., because it is directly controlled by an overseeing principal that makes the authorization decisions.

We distinguish three architecture variants:

- four-corner architecture,
- combined CAM and SAM,
- combined C and CAM,
- combined S and SAM.

DCAF can be used in all of these scenarios if the combined endpoints act accordingly. For the other actors nothing changes: they use the normal protocol flow. Details about the protocol flow for these scenarios can be found in Appendix C.

## 12.4. Revocation

In DCAF, SAM and CAM may use revocation to call back access tickets that are no longer valid, e.g., because C or S was compromised. The more strict a revocation mechanism is, the more strain it puts on the constrained devices. Overseeing principals must consider the advantages and disadvantages of a revocation mechanism and decide if and how it is useful for their application scenario. Using a revocation mechanism is therefore optional in DCAF. In the following, we will describe a draft for a revocation approach that uses CoAP observe (RFC 7641).

To use observe for revocation, the AM must act as an observe server. To get regular revocation updates, C and S must register with their AM's registration

resource. They may obtain the path of the revocation resource during the setup of the security association. As an alternative, the name of the revocation resource may be obtained using the AM's well-known resource.

To make a server-side revocation known to C, CAM may register with SAM's revocation resource and relay the information to C by revoking the respective client information for C. Similarly, SAM may register with CAM's revocation resource and react to revoked client information by also revoking the respective access ticket. SAM can thus handle situations where a client no longer belongs to CAM, e.g., because it was compromised or decommissioned (see also section 14.4.5).

After the observe registration, the AM must regularly send the current revocation state to the observing endpoints. To do so, a sliding window mechanism may be used (cf. Hartwich 2015, p. 53). The revocation message contains the lowest ticket number as the lower bounds, and the size of the window. If possible, the window size should correspond to the maximum number of tickets that are valid at the same time. The tickets are represented by a bit vector where each bit indicates if a ticket was revoked (1) or not (0). Thereby, the whole revocation state can be transmitted in an observe notification.

The observe server must regularly provide updates to its observers. Otherwise an attacker might intercept and withhold revocation messages and thereby render the mechanism ineffective. The overseeing principals should define a maximum time span between observe messages. If an observer did not receive an update during this time span, it should request an update from the observe server. If the observer gets no response, the connection to the observe server should be considered as disturbed. How an observer should behave in such a case depends on the overseeing principal's security objectives. If it is most important to protect the data, the endpoints should mark claims from this server as *not reliable* until they hear from the server again. The endpoints should not communicate with peers whose permissions or attribute claims are marked that way. If the availability of the service that the endpoint provides is more important, it should continue to communicate. This may result in the endpoint communicating with peers for which the permissions were revoked.

The observe notifications must be integrity-protected and the subscriber of revocation notifications must ascertain that received notifications stem from an au-

thorized AM. The underlying security solution should also provide replay protection. The AM may also confidentiality-protect the revocation messages if the overseeing principals decide that the knowledge of revoked tickets is sensitive information.

If an AM revokes a ticket, C and S must keep the revocation information as long as the revoked ticket is valid. The respective ticket face and client information must be removed. DTLS connections that exist for the ticket should be terminated.

## 12.5. Group Communication

DCAF may be used to distribute the keying material and authorization information that is required for group communication. As usual, the constrained devices must first have a security association with their respective AM (see also section 12.1.2) before they can use DCAF.

DCAF allows the use of attributes to identify multiple entities with the same characteristics. SAM is thus able to issue tickets to all clients with certain attributes, e.g., to all clients that are light switches and are located in the living room. Also, SAM can define the destination of the ticket so that all servers with certain attributes are included, e.g., all servers that are light bulbs and are located in the living room. Similarly, CAM can inform all clients with certain attributes about COP's authorization rules, and those rules may refer to all servers with specific attributes. As usual, the constrained nodes must be able to validate that authorization information and keying material stem from an authorized AM, i.e., they must validate the integrity and authenticity of the ticket face and the client information. The keying material that identifies the AM should be an asymmetric key. If this is not possible, distinct keying material needs to be used for every constrained device.

The access ticket must reach all constrained devices that require them. How this is achieved is not part of this work. The AM may, e.g., use a multicast protocol to communicate with its constrained nodes. In this case, an object security solution is used to protect the DCAF communication. Constrained devices should not

communicate with other devices if they did not yet receive authorization information from their AM. To allow for group communication, the keying material that is distributed with the ticket face and client information should be a symmetric key that is shared between all constrained devices in the group and their AMs.

The distributed keying material is then used to secure the group communication. **If an access ticket (client information or ticket face) refers to certain keying material, all nodes that use this keying material can perform all actions that are covered by this ticket.** The group communication keys must therefore only be used for tasks that every member of the group is allowed to perform. E.g., if every group member is allowed to receive a certain resource representation, the group key may be used to protect the confidentiality of the message. As explained in section 5.4.2, verifiers must exclusively identify entities for which the claim statement is true. If certain actions must only be performed by a certain device, e.g., only an authorized light switch is allowed to command light bulbs to turn on, the keying material must be suitable to identify these authorized entities. The constrained devices must check if the messages are protected accordingly. A combination of mechanisms may be useful in many cases: group keys are used to enforce permissions for the whole group, and individual keys ascertain the authorization for actions that only certain devices are allowed to perform.

For constrained devices that use group communication it is especially important that the keying material is changed often and that they use a revocation mechanism (see section 12.4).

## 12.6. Server-Initiated Ticket Request

There may be scenarios where the client is not able to directly contact its CAM and only the server can reach its SAM. For these scenarios, we define the Server-Initiated Ticket Request (SITR) (draft-gerdes-ace-dcaf-sitr). It enables the server to obtain the access ticket for the client.

Servers are the passive part in a communication relationship since they react to incoming requests. They therefore are therefore more vulnerable to DoS attacks

(see also section 12.7.1). If the server is responsible for obtaining the authorization rules, it must send a request to SAM for each client request. A malicious client can exploit this behavior by requesting arbitrary resources without a ticket. Since sending and managing requests is time and energy-consuming, the server will become less responsive and might no longer be able to provide its service. Solutions where the server is responsible for obtaining the authorization rules should be avoided if possible. Overseeing principals should consider to exchange the roles of client and server instead.

The Server-Initiated Ticket Request (SITR) protocol flow (see also draft-gerdes-ace-dcaf-sitr) is depicted in figure 12.7. SITR reuses DCAF concepts. We therefore choose to forgo a detailed description of SITR in this work.

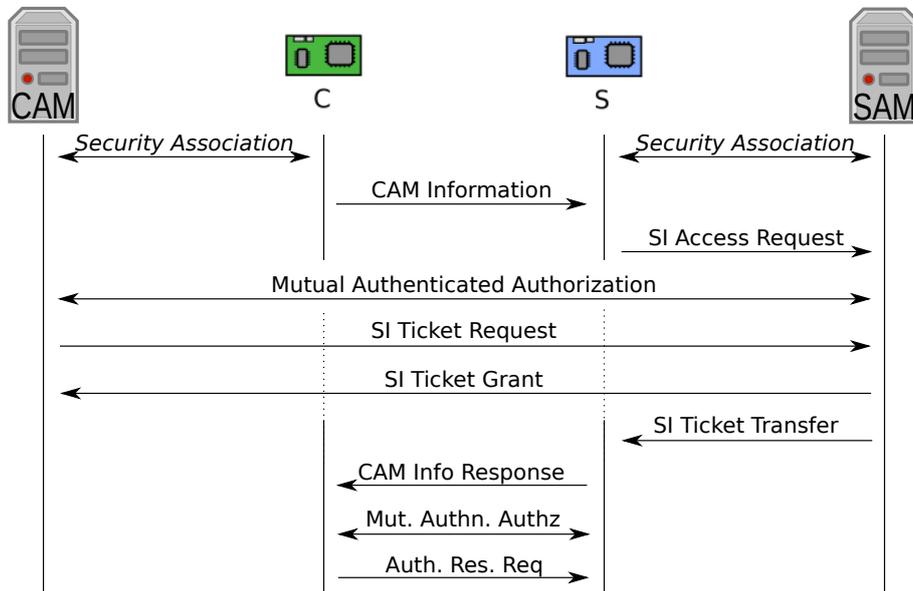


Figure 12.7.: Server-Initiated Ticket Request Protocol Flow

## 12.7. RESTful Design

As an authorization framework, DCAF's objective is the secure and authorized transmission of data. To achieve this, DCAF considers the limitations and special characteristics of constrained devices and of the environments where they are

used. To be of use for a Web of Things, DCAF integrates REST principles where possible.

### **12.7.1. Statelessness**

Constrained devices are especially prone to DoS attacks (see also section 4.2.4). In RESTful environments, servers need more protection than clients. Since clients initiate the conversation they have control over the number of connections they attempt to open at a given time. RESTful servers are passively waiting for requests from clients. They have no control over the time of the request or the number of clients that are trying to contact them at once. Thus, servers are more vulnerable to DoS attacks.

DCAF is designed to limit the effort for clients and servers as much as possible. Difficult security-related tasks are delegated to their Authorization Managers (see also section 9.3.1). Initially, each constrained device only needs to be configured with the credentials of its own authorization manager. Authentication and authorization information concerning other communication partners is obtained if needed. Thus, the required storage space on the devices is limited.

The server itself only performs the tasks that each device that securely participates in the communication must perform. The procurement of the ticket is left to the client. Only after being presented with a valid ticket, the server allows the client to establish a session and start the communication. Leaving the effort of obtaining the ticket to the client is necessary to protect against DoS attacks and to support the REST stateless server constraint (see also section 3.4).

The DCAF ticket face is self-descriptive: it contains all information that the server requires to securely communicate with C. The burden on the server is reduced since it does not need to obtain additional data.

### **12.7.2. Loosely Coupled Components**

DCAF implements the task delegation architectural style (see chapter 9). Constrained devices have a close relationship with their respective less-constrained

authorization manager. These devices are more closely coupled by nature. As they belong to the same security domain, they can easily be evolved together.

The need for the client side to understand the server side is reduced to a minimum. The respective interfaces are clearly defined in DCAF. Thus, components on the client side are only loosely coupled to components on the server side. Unlike the ACE framework, DCAF avoids the close coupling of endpoints from distinct security domains.

That each constrained device is coupled with its own less-constrained AM reduces the burden on constrained devices. The AM assists with difficult tasks and communicates with AMs from other security domains. By including two AMs in DCAF, the provisioning of the required authentication and authorization information to the constrained devices can be combined in a single protocol flow. Thus, DCAF is especially useful for cross-domain communication (see also Heuer et al. 2015, p. 40)

## 12.8. Implementation

We provide an open source implementation<sup>3</sup> of our DCAF protocol as a proof of concept. The implementation was conducted in cooperation with Olaf Bergmann. Our primary contribution is the implementation of the protocol logic.

Our DCAF implementation uses the CoAP library `libcoap`<sup>4</sup>. As the CBOR implementation `cn_cbor`<sup>5</sup> and, more recently, additionally Olaf Bergmann's CBOR implementation `anybor` that was optimized for DCAF are utilized. The main DCAF actors, i.e., the constrained devices and their authorization managers, were implemented. The implementation of the constrained devices is written in C, the code for the authorization managers has been developed in C++.

---

<sup>3</sup><https://dcaf.science>

<sup>4</sup><https://libcoap.net>

<sup>5</sup><https://github.com/cabo/cn-cbor>

To facilitate the use of DCAF on constrained devices, our DCAF implementation has been packaged for RIOT<sup>6</sup> by Olaf Bergmann<sup>7</sup>. RIOT is an open source operating system for embedded devices.

An older DCAF version was implemented by Tobias Hartwich for the Contiki operating system<sup>8</sup>. Also, a DCAF extension<sup>9</sup> was developed by Sara Stadler that includes attribute based credentials to allow for anonymous CAMs.

We will examine the code size and RAM requirements of our implementation in section 13.2.2.

## 12.9. Conclusion

In this chapter, we have presented DCAF, a framework to establish authenticated and authorized communication between constrained devices. Each constrained device has its own authorization manager that assists in performing the necessary authentication and authorization tasks. DCAF thus implements the four-corner architecture and fully supports autonomous clients. Where this is not required, e.g., because client and server share a single authorization manager, or because the client is directly controlled by its overseeing principal, the DCAF protocol flow can be adjusted accordingly. DCAF implements the task delegation architectural style and therefore is very flexible. It can be adapted to the needs of the respective application scenario.

DCAF aims at avoiding gaps in the authenticated authorization process. It clearly specifies the pieces of data that are out of scope for DCAF but must be exchanged prior to or during the communication for a secure protocol flow. For the authorization information and the related keying material, DCAF provides the necessary semantic information and explicitly defines the required parameters that enable both client and server to validate and process the data.

---

<sup>6</sup><https://riot-os.org>

<sup>7</sup><https://github.com/obgm/RIOT/tree/dcaf>

<sup>8</sup><https://github.com/toha/DCAF>

<sup>9</sup><https://gitlab.informatik.uni-bremen.de/DCAF/dcaf/tree/abc>

DCAF does not require unique identifiers for their endpoints, which facilitates the protection of the communication partners' privacy. The transmission of unprotected information is avoided where possible. A special resource is reserved for the initial request, so that the client does not need to disclose confidential request data.

Instead of identifiers, DCAF enables overseeing principals to define attributes for their constrained devices. The authorization managers vouch for the attributes of their constrained devices. The attributes can be chosen to fit to the current application scenario. Attribute-based authenticated authorization is very flexible since endpoints can communicate with any entity that has the required attributes. The solution is thus able to react quickly to situations where a desired communication partner suddenly becomes unavailable. A client may discover suitable servers, e.g., with the help of resource directories. Also, CAM may inform C with which server C is supposed to speak by providing it with the respective access ticket.

DCAF's support for constrained devices goes beyond reducing message sizes. It introduces three options for determining the validity of ticket face and client information. Even constrained devices with only a very limited ability to measure time are thereby enabled to communicate securely. An additional time synchronization mechanism is not required.

In the IoT, autonomous clients must enforce authorization rules without the assistance of their overseeing principals. In DCAF, the authorization managers can provide authorization rules not only to servers but also to clients. Also, each constrained device is enabled to check if authentication and authorization information is still valid. The provisioning with authentication and authorization information for both the client and the server is combined in a single protocol flow. No additional messages are required, which reduces the burden on the constrained devices.

In the Internet, clients and servers do not necessarily know each other before they communicate. With DCAF, constrained devices are not required to register with an AM from a different security domain. Our framework is therefore particularly flexible since it allows for spontaneous setup of a secure constrained

to constrained communication across organization boundaries. Making information available regardless of organizational boundaries is, according to Fielding and Taylor, the main purpose of the Internet (Fielding and Taylor 2002, p. 119). Because of its unique features, DCAF therefore is a particularly good fit for the Internet of Things.

## 13. DCAF Evaluation

In this chapter we analyze if DCAF enables the participating endpoints to communicate securely, and evaluate if our framework is suitable for constrained environments. We apply ANAZEM (see chapter 5) to the normal DCAF protocol flow to analyze if DCAF complies with the authorization and delegation fundamentals (see section 13.1). Quantitative observations are made in section 13.2, where we examine the quantity of network communication for the normal DCAF protocol flow, and determine the memory footprint of the DCAF implementation. In section 13.3, we analyze if DCAF meets the requirements that we defined in the background part, in particular the hardware requirements of constrained devices (see chapter 3), and the requirements that we derived from the use cases (see chapter 4). Section 13.4 summarizes the findings.

### 13.1. Authorization Fundamentals Evaluation

The authorization fundamentals (see section 5.2.2) must be applied to every message by each communication partner. We will now analyze if DCAF enables the participating actors to comply with the authorization fundamentals. In particular, we will check if DCAF is suitable to securely provide S and C with the necessary information to perform authenticated authorization. We will focus on DCAF's basic protocol flow in the evaluation and analyze each step of it.

The DCAF protocol description presented in chapter 12 is different from the first draft: we used the findings of our analysis to adapt DCAF until it described all necessary tasks. Our analysis has two main objectives: first, to show that DCAF

securely provides constrained clients and servers with all necessary data to enable them to participate in the protection of their overseeing principals' data; and second to show that it is possible to write a protocol that describes how every necessary authorization and delegation task must be performed. By listing the tasks that need to be performed outside of the protocol flow and identifying the requirements on underlying security solutions, the risk of gaps and vulnerabilities is mitigated and a secure overall solution can be reached.

The various actors that participate in the DCAF protocol flow often need to act in different roles in the terms of our model; e.g., CAM receives information from COP and then acts as a delegator. But CAM also provides information to C and then has the role of the claim issuer. The role determines which delegation and authorization tasks the actors must perform. Actors may also need to perform tasks multiple times for different actors; e.g., SAM may receive information from SOP and CAM and must perform the necessary tasks for each of these claim issuers. Endpoints may even have to perform the same task for the same endpoint multiple times to enforce the decisions of each of their overseeing principals; e.g., CAM must perform the authorization tasks concerning SAM to enforce COP's decisions, and then again to also consider those from SOP. The resulting actions that the endpoint must perform may be the same, e.g., encrypting the ticket request message; but it is still necessary to check if additional actions are required.

We assume that the underlying security protocols such as DTLS or object security mechanisms are not compromised and are able to perform the tasks that they are used for in compliance with the authorization and delegation fundamentals (**Unbroken Cryptography Assumption**).

We will only analyze the protocol flow itself; threats outside the protocol flow, e.g., physical attacks on the device, are not considered. Likewise, we will only analyze the integrity and confidentiality of data that is transported in the protocol flow. We assume that claims and permissions that were received outside the protocol flow—such as time synchronization information—were obtained and validated in compliance with ANAZEM (**Model-Compliant Claims Assumption**). Nevertheless, the protocol must describe where such claims are used, which requirements must be met (if and how claims must be protected) and how claim statements are used in the protocol flow.

We will assume that sensitive data was not disclosed or modified outside of the protocol flow, in particular confidential keying material such as the symmetric keys or asymmetric private keys. This assumption does not rule out that keying material may be compromised in the future. (**Non-Disclosure Assumption**).

In DCAF, C and S delegate authentication and authorization tasks to their respective AM. CAM provides C's characteristics and maybe also its keying material to SAM as input for the authenticated authorization. CAM relies on SAM concerning the server's characteristics and keying material. We will now analyze the messages of the basic DCAF protocol flow with ANAZEM.

### **13.1.1. Unauthorized Resource Request Exchange**

As described in section 12.2.1, one way for C to obtain SAM's address is to send an unauthorized resource request message to S that is answered by S with the SAM information message. The communicating parties are C and S. The messages are not protected since C and S do not yet have the means to do so. Accordingly, C and S are not able to validate each other's authorization. To satisfy the authorization fundamentals, they must use unrestricted authorization. DCAF specifies that no sensitive data must be transmitted in the unauthorized resource request message (see section 12.2.1).

Which resource C requests may be confidential information. DCAF therefore provides the AM-info resource. C can use it to obtain a SAM information message without revealing which resource it actually wants to access.

### **13.1.2. Access Request Evaluation**

C sends an access request to CAM to obtain the access ticket. It may comprise the SAM address, the requested resource and the actions that C wants to perform on the resource. It may also contain validity information and the chosen content format.

### 13.1.2.1. C's Side

The application may require the confidentiality of request data. In this case, C must ensure that the message is sent to a specific authorized CAM and perform the authorization tasks.

In DCAF, C must protect the confidentiality of the access request message (see section 12.2.2). As described in section 12.1.2, C was informed by COP which entities are authorized to be its CAM, and obtained the respective keying material for them. DCAF does not specify how the initial key provisioning is accomplished, but demands that the authorization rules and corresponding keying material were securely provisioned. DCAF also demands that security associations between the constrained devices and their authorization managers must be updated regularly. This covers the attribute validation (task **An1**) the configuration of the authorization information (task **A1**) and the task of obtaining them (task **A2**).

How C validates and evaluates the obtained permissions for CAM is with the exception of task **A6** out of scope in DCAF, but C is expected to have securely obtained the knowledge from COP which entities are authorized CAMs, which includes the tasks **A3 – A5**, task **A7** and **A8**. To protect the confidentiality of the message, C must use the keying material that identifies an authorized CAM. C thereby performs task **A6**.

### 13.1.2.2. CAM's Side

The data of the access request contains input that may be used by CAM (and also SAM) to generate the parts of the access ticket. The ticket face and client information are claims, and the participating parties must comply with the delegation fundamentals. The **claim issuer participation fundamental** requires claim issuers to check the authorization of entities that provide input to claims. For the information in the access request message, unrestricted authorization does not suffice. An attacker that modifies requested resources or validity information cannot trick CAM or SAM into falsely granting access. But the availability of

the server's resources may be affected: SAM might generate an access ticket for resources in which C is not interested. Also, the ticket face may contain wrong validity information which would result in S discarding the ticket. Therefore, CAM should actively validate C's authorization and thus perform task **D0**, which means that it must perform the authorization tasks for C.

In DCAF, authorization information and keying material concerning C is expected to be securely provided to CAM and must be kept up to date (see section 12.1.2). This covers the tasks **A1**, **A2**. The details of this process are out of scope for DCAF.

The validation of the authorization information is also out of scope, but CAM must have securely obtained from COP for which clients it is responsible (see section 12.1.2), which includes the validation and evaluation of the authorization information. CAM thereby performs the tasks **A3** to **A5**, task **A7** and task **A8**.

If COP informed CAM that it is in charge of C, C is authorized to send an access request to CAM. In DCAF, CAM must check if it is responsible for the client that sent the access request message (see section 12.2.3). Also, the integrity and authenticity of the access request message must be protected. CAM performs task **A6** using the obtained keying material that identifies C as an authorized client. In DCAF, C is allowed to send the access request if it proves to CAM that it can use certain credentials (see section 12.2.3). DCAF therefore describes how CAM performs task **A6** for C.

### **13.1.3. Ticket Request Evaluation**

CAM sends the ticket request message to SAM to request a ticket in behalf of C. The ticket request message contains the request data, and may also include C's characteristics and C's RPK. CAM assists SAM with task **An1**, either directly by providing C's characteristics, or implicitly by relaying the ticket to the correct C. CAM thus is a claim issuer for SAM. If CAM receives keying material for the server from SAM, SAM also is a claim issuer for CAM. CAM and SAM must perform the respective delegation tasks.

### 13.1.3.1. CAM's Side

CAM validates *C*'s attributes for SAM (task **An1**). As a claim issuer for SAM, CAM must perform the delegation tasks **D0** and **D1**. As a claim issuer for *C*, CAM must validate SAM's authorization to receive the ticket request message to comply with the **claim issuer participation fundamental**.

In DCAF, CAM is expected to securely obtain *C*'s attributes and keying material (see section 12.1.2), and must ascertain that they are still valid before including them in the ticket request message (see section 12.2.3). If CAM does not explicitly state *C*'s attributes, it must ascertain that the ticket is provided to the correct *C*. In DCAF, CAM must protect the confidentiality of the ticket transfer message and ascertain that it is securely transmitted to the *C* that sent the access request message (see section 12.2.6). CAM thereby performs task **An1** for SAM.

CAM only explicitly provides a claim to SAM in the ticket request message if it states *C*'s attributes. DCAF defines how CAM states *C*'s characteristics for task **D1a** (see section 12.2.3). To specify the destination of the claim, CAM relies on the underlying security mechanism. In DCAF, the communication between CAM and SAM must be encrypted before the integrity-protection is applied, or an AEAD algorithm must be used (see section 12.1.4). Thereby, the destination of the claim is set (task **D1b**). CAM must relay the access ticket to the *C* that sent the access request message (see section 12.2.3). Additionally, CAM may specify *C*'s public key in the ticket request message. CAM thereby binds the holder to the claim and performs task **D1c**. The security solution is used by CAM to endorse the claim: it protects the integrity and authenticity of the ticket request (see section 12.1.4). CAM therefore performs task **D1** for SAM by using DCAF.

For task **D0**, CAM must check if—according to the server's SOP—this SAM is authorized to receive *C*'s attributes and keying material. Whether the data of the ticket request message is confidential depends on the application. If the data is not confidential, every SAM is authorized to receive it. Otherwise, CAM must ascertain that it send the ticket request message to a SAM that is authorized by SOP. DCAF demands that for these cases the responsible SAM and the resource on SAM where the access ticket is requested must securely be obtained from a

claim issuer that is authorized by SOP (see section 12.2.1). The details of this process are out of scope. If the **Model-Compliant Claims Assumption** holds, the authorization tasks **A1** to **A5**, **A7** and **A8** were performed correctly. CAM must ascertain that it communicates with a SAM that is authorized by SOP before sending the ticket request message (see section 12.1.3). CAM thereby performs task **A6** for SAM.

As a claim issuer for C, CAM must ascertain that COP also authorized SAM to get the ticket request message. CAM therefore must perform the authorization tasks for SAM with COP as the overseeing principal. COP must securely provide its authorization rules about SAM to CAM for the tasks **A1** and **A2**. In DCAF, the provisioning of authorization rules to CAM is out of scope, but the authorization rules and corresponding keying material for SAM are expected to securely have been provisioned by COP to CAM and to be updated regularly (see section 12.1.3). COP and CAM thereby perform the tasks **A1**, **A2** and **An1** concerning SAM. DCAF also specifies how authorization rules must be validated and evaluated, which covers the tasks **A3** – **A5**, **A7** and **A8**.

To perform task **A6**, CAM must encrypt the message. DCAF specifies that the confidentiality of the ticket request message must be protected (see section 12.2.3). To achieve this, CAM relies on the underlying security protocol, which must offer confidentiality-protection (see section 12.1.4). CAM thus performs task **A6** for SAM.

#### **13.1.3.2. SAM's Side**

Before SAM processes a received ticket request message, it must validate CAM's authorization to provide the information in the message. CAM is authorized if SOP allows SAM to provide access tickets to CAM and CAM's clients. If the ticket request contains C's attributes, SAM must also check if SOP authorized CAM to provide them before the attributes are processed. The details about this process are out of scope, but DCAF states that SOP must have provided authorization rules and associated keying material concerning CAM to SAM, and that these pieces of information must be updated regularly (see section 12.1.2). This

covers the tasks **A1**, **A2** and **An1**. DCAF demands that SAM must securely be provided with authorization rules for CAM, which includes that SAM validates and evaluates them (tasks **A3** – **A5**, task **A7** and task **A8**).

To perform task **A6**, SAM must rely on the underlying security protocol. In DCAF, the integrity and authenticity of the ticket request message must be protected (see section 12.2.3). Also, SAM must check that the ticket request message stems from an authorized CAM. SAM thus performs task **A6** for CAM.

If the ticket request message contains a claim about *C*, SAM must perform the delegation tasks as the delegator. For task **D2** (obtaining the claim), SAM must validate if CAM's claim about *C* is still valid. In DCAF, CAM must provide a timestamp with the ticket request that defines the creation time of the request and a lifetime, if the ticket request contains *C*'s attributes (see section 12.2.3). SAM must check them when it receives the message. SAM thereby performs task **D2**. Additionally, CAM may offer a revocation service that may be used by SAM which may contribute to satisfying the **claim completeness fundamental**.

To validate if all required information is bound to the claim (task **D3**), SAM must rely on the underlying security mechanism. In DCAF, the integrity and authenticity of the ticket request message must be protected. SAM thereby validates that the claim statement and destination are bound together. If CAM does not explicitly specify the holder in the claim, e.g., if pre-shared keys are used between *C* and *S*, SAM must rely on CAM to send the access ticket to *C* as the holder of the claim. DCAF therefore specifies how SAM must perform task **D3**.

SAM must have validated that CAM is authorized to provide information about *C* (task **D4**). How SAM performs the authorization tasks for CAM is described above. SAM relies on the underlying security solution to ensure that it is the destination of the claim: since the ticket request message is confidentiality-protected, SAM is only able to read it if it is the intended destination (task **D5**).

SAM uses CAM's claim about *C* to create the access ticket for *C*. By using the keying material specified by CAM or trusting CAM to relay the ticket to *C*, SAM ensures that only the holder of the claim can use the access ticket (task **D6**): *S* then only communicates with *C* if *S* can validate *C*'s authorization using the keying

material that SAM specifies in the ticket face (see section 13.1.9). CAM is the only claim issuer that states attribute claims concerning C for SAM in DCAF (task **D8**).

To comply with the **claim issuer participation fundamental**, SAM must also check if CAM is authorized by COP to provide input to the access ticket (task **D0**). CAM is authorized to provide input for access tickets which it later receives itself. In DCAF, SAM validates CAM's authorization by ascertaining that the same CAM that sent the access ticket request receives the access ticket (see section 12.2.5).

### **13.1.4. Ticket Face Generation Evaluation**

S delegates authorization tasks to SAM: SAM determines for S if C is authorized by SOP. SAM receives the authorization rules as a claim from SOP. SOP and SAM thus must perform the delegation tasks. SAM must also perform the delegation tasks **D0** and **D1** in the role of the claim issuer for S.

#### **13.1.4.1. SAM Validates the Server's Authorization (Task D0)**

For task **D0**, SAM must perform the authorization tasks for S to validate that—according to SOP—S is authorized to delegate tasks to SAM and to receive the access ticket. In DCAF, SOP is expected to provide SAM with the authorization information and keying material for each of SAM's servers (tasks **A1** and **A2**), and SAM must have validated this information (tasks **A3** to **A5**, **A7** and **A8**), see section 12.1.2).

To perform task **A6** for task **D0**, SAM must provide the access ticket only to the authorized S. DCAF demands that the ticket face must be confidentiality-protected if it contains confidential data, and that SAM must send it to the correct server (see section 12.2.4.1). SAM thus performs the authorization tasks that are necessary for task **D0**.

#### 13.1.4.2. SAM performs Tasks in Behalf of S

SAM must perform task **An1** concerning C in behalf of S. In DCAF, SAM has a security association with CAM but not with C. SAM therefore relies on CAM for the validation of C's attributes. SOP authorizes CAM to relay access tickets to clients (see section 12.1.2), and thereby CAM's clients to access S as the access ticket permits. SOP may also authorize CAM to provide C's characteristics to SAM to enable more fine-grained authorization for C. How SAM validates CAM's authorization to provide C's attributes and how SAM performs the delegation tasks for claims from CAM is described in section 13.1.3.2.

SAM receives the authorization rules concerning the access to S as a claim from SOP. This claim must contain all necessary information to enable CAM to specify an authorization claim for S that reflects SOP's decisions. In DCAF, the overseeing principals are expected to have provided their AMs with the authorization information entries for their constrained devices that contain the permission, the destination, i.e., the constrained device that must enforce the permission (in this case S), the AM that is the holder of the permission (CAM), the validity time, and, for fine-grained authorization, the attributes of the C that is authorized to access, which then also is a holder of the permission (see section 12.1.3). SOP must bind these pieces of information to each other and endorse them, thereby performing task **D1**. SOP must provide the authorization information entries to SAM and must keep them up to date. SOP and SAM thereby perform task **D2**. The entries must securely be obtained which means that SAM must validate and evaluate the claims and thus performs tasks **D3** to **D5** and **D8**.

For task **D6**, SAM must ascertain that the access ticket is transmitted to the CAM that is the holder of the claim. In DCAF, the confidentiality of the ticket grant message must be protected (see section 12.2.5), and SAM must ascertain that CAM is authorized by SOP (see section 12.1.2). If COP specified fine-grained authorization rules, SAM must check if the client has the required attributes (see section 12.2.4). Thus, SAM performs task **D6**.

### 13.1.4.3. SAM Specifies the Access Ticket for S (task D1)

SAM must convey the information from SOP to S. In DCAF, SAM specifies SOP's authorization rules for S, CAM and C in the ticket face if SOP specified detailed authorization rules (see section 12.2.4.1). If SOP did not provide detailed rules or SAM is certain that S already knows the rules, SAM may only provide C's keying material to S. SAM thereby defines the claim statement (task **D1a**).

Also, SAM must specify which server is the intended recipient of the ticket face for task **D1b**. DCAF ensures that this task is performed by demanding that SAM specifies the intended recipient of the ticket face either implicitly by encrypting the face with the server's keying material, or explicitly by providing an identifier for S in the ticket face (see section 12.2.4.1).

SAM must bind the ticket face to the C that SOP specified in the authorization rules to perform task **D1c**. SAM relies on CAM for task **An1** concerning C: CAM provides the client information to the correct C in the ticket transfer message. Also, CAM may relay C's RPK to SAM in the ticket request message. SAM performs task **D1c** by specifying C's RPK, a symmetric key or a key derivation function with which S can derive the symmetric key in the access ticket. To comply with the **holder relation fundamental**, the verifier must securely be distributed to the participating parties (see section 5.4.2.1). DCAF demands that the confidentiality of symmetric keying material is always protected (see sections 12.1.4, 12.2.4.1, 12.2.5, and 12.2.6).

SAM must bind the statement, holder, destination, and freshness information together to finish task **D1**. In DCAF, SAM must ensure that S is able to validate the integrity and authenticity of the ticket face. To do so, SAM must provide a signature or MAC of the ticket face, or SAM derives the symmetric key from the ticket face using the keying material that it shares with S (see section 12.2.4.1). If SAM implicitly defines the destination by encrypting the ticket face, DCAF demands that the ticket is first encrypted and then integrity-protected or that an AEAD algorithm is used (see section 12.2.4.1). DCAF therefore defines how SAM must perform task **D1** for the authorization tasks that it performs for S. If the **Unbroken Cryptography Assumption** and the **Non-Disclosure Assumption** hold, the task is performed in compliance with the authorization fundamentals.

To reduce the burden on S, SAM may provide simplified authorization rules to S and thus assist S in performing task **A8**. DCAF specifies that SAM should only provide a single permission per resource (see section 12.2.4.1).

### **13.1.5. Client Information Generation Evaluation**

CAM and C may rely on SAM to provide the server's keying material. SAM then must perform task **An1** for them, as well as the delegation tasks **D0** and **D1**.

#### **13.1.5.1. SAM Validates the Server's Attributes (Task An1)**

CAM specifies the attributes of the server with which C wants to communicate in the ticket request message. To help CAM and C with task **An1**, SAM must use these attributes to identify S. In DCAF, SAM is expected to securely have obtained the attributes and keying material of the servers for which it is responsible, and the attributes must be kept up to date (see section 12.1.2). In DCAF, SAM must specify keying material in the client information that corresponds to the server that CAM specified in the ticket request message (see section 12.2.4.2) and thereby helps CAM and C with task **An1** concerning S.

#### **13.1.5.2. SAM Performs the Delegation Tasks for CAM**

For task **D0**, SAM must validate that the ticket request message that delivers input for task **An1** stems from an authorized CAM. Section 13.1.3.2 analyzes how SAM validates CAM's authorization in DCAF and thereby also performs the tasks **A2** to **A5**, **A7**, and **A8** for task **D0**.

The client information must be transferred to the authorized CAM for task **A6**. In DCAF, the ticket grant message must be confidentiality-protected (see section 12.2.5). SAM relies on the underlying security solution for performing task **A6** for task **D0**. SAM thus performs task **D0** for CAM in DCAF.

To perform task **D1**, SAM must first set the claim statement (task **D1a**), in this case the server's attributes. In DCAF, CAM must specify the server's attributes in the ticket request message (see section 12.2.3), and the ticket grant message must be bound to this ticket request message. Otherwise, CAM would not be able to distinguish access tickets for two distinct servers, and may provide the wrong access ticket to C. In DCAF, the underlying security solution must enable the communication partners to determine if a message belongs to the current communication context (see section 12.1.4). Therefore, DCAF describes how SAM must perform task **D1a** for CAM and C.

SAM must set the CAM that is the intended destination of the claim for task **D1b**. In DCAF, SAM must protect the confidentiality of the client information (see section 12.2.5). It thereby specifies CAM as the intended destination and performs task **D1b**.

For task **D1c**, SAM must specify the server as the holder of the client information. In DCAF, SAM must ensure that it provides the verifier of the server with which C wants to communicate in the client information (see section 12.2.4.2). SAM either generates a symmetric key that must securely be provided to C and S, or uses the server's public key as the verifier for S. DCAF demands that the confidentiality of the symmetric key is always protected (see sections 12.1.4, 12.2.4.1, 12.2.5, and 12.2.6), which covers task **D1c**.

SAM must endorse the statement, destination, holder, and freshness information of the client information to finish task **D1**. In DCAF, SAM must protect the integrity and authenticity of the client information (see section 12.2.5). The statement is bound to the ticket as described above. The confidentiality-protection must have been applied before, or an AEAD algorithm must be used, which provides data destination verifiability. DCAF therefore describes every necessary detail of how SAM performs task **D1** for CAM and C.

### **13.1.6. Ticket Grant Message Evaluation**

The ticket grant message contains the ticket face and the client information. We evaluate how SAM performs the necessary tasks for these parts of the message in sections 13.1.4 and 13.1.5. In this section, we focus on CAM's side.

If SAM provides client information to CAM, CAM receives an attribute claim concerning S with the ticket grant message. We describe how SAM performs the delegation tasks **D0** and **D1** for CAM in section 13.1.5.2. CAM must perform the remaining delegation tasks for the delegation to SAM.

For task **D2**, CAM must check if the claim in the ticket grant message is up to date. In DCAF, SAM must provide a timestamp in the client information that specifies when the claim was created (see section 12.2.4.2). The ticket grant message must also contain the lifetime of the ticket. CAM and SAM are expected to be time-synchronized (see section 12.1.7). The communication security solution must protect the integrity of the client information (see section 12.2.4.2). CAM is therefore able to determine how old the client information is and how long it is supposed to be valid. The creation time also allows CAM to determine which client information is the most recent. DCAF therefore defines how CAM must perform task **D2**. The revocation mechanism described in section 12.4 may further contribute to satisfying the **rule completeness fundamental**, but the evaluation of this mechanism is not part of this analysis.

For task **D3**, CAM must validate that the claim information is bound together and stems from the same SAM. In DCAF, the client information must provide information about the server which CAM specified in the ticket request message (see section 12.2.4.2). The security solution must enable the communication partners to determine if a message belongs to the current communication context (see section 12.1.4). The claim statement is therefore bound to the claim. The information that CAM is the intended destination of the claim is provided by encrypting the message with the security solution (see also below). The holder is represented by the keying material that is specified in the client information and protected by the communication security solution. If the **Unbroken Cryptography Assumption** holds, CAM performs task **D3** by relying on the security solution.

CAM must check if SAM is authorized to provide the client information concerning S to perform task **D4**. How CAM performs the authorization tasks **A2** to **A5**, **A7** and **A8** for SAM is described in section 13.1.3.1. SAM must protect the integrity of the ticket grant message (see section 12.1.4). CAM relies on the underlying communication security solution to perform task **A6**.

In DCAF, the confidentiality of the messages that are exchanged between CAM and SAM must be protected, and the underlying security solution must provide data destination verifiability for these messages (see section 12.1.4). CAM therefore ascertains that it is the intended destination of the ticket grant message using the underlying communication security solution, and thus performs task **D5**.

CAM does not use the client information itself but relays it to C. How C performs task **D6** is described in section 13.1.9. In DCAF, SAM is the only claim issuer for CAM concerning S. CAM explicitly sends a request to SAM and can thereby determine which client information is the most recent (task **D8**).

### **13.1.7. Ticket Transfer Message Evaluation**

C has no security association with SAM. It relies on CAM to perform authorization tasks and attribute validation for S. CAM may assist C in validating the server's attributes (task **An1**). Also, CAM may provide C with authorization information from COP. In these cases, CAM is a claim issuer for C and must perform the delegation tasks **D0** and **D1** for C. If the claim is an authorization claim, CAM must also perform task **A1**. When CAM receives claims from COP and SAM, CAM must perform the respective delegation tasks to securely obtain these claims.

#### **13.1.7.1. CAM Performs Tasks for C**

If CAM receives keying material from SAM, it relies on SAM to validate the server's attributes (task **An1**). CAM then receives the necessary information with the ticket grant message. Section 13.1.6 describes how CAM performs the delegation tasks for SAM and validates SAM's authorization to provide this information.

CAM may also assist C with authorization tasks. If this is the case, CAM must obtain a claim from COP with the respective authorization rules for C. Accordingly, COP and CAM must perform the delegation tasks. To perform task **D1**,

COP issues a claim for CAM containing the rules. In DCAF, CAM may obtain authorization rules concerning C, SAM and S from COP (see sections 12.1.2 and 12.1.3). The respective authorization rules and keying material must be kept up to date in DCAF. COP and CAM are therefore also expected to perform task **D2**. CAM must securely have obtained the authorization rules from a claim issuer that is authorized by COP (see section 12.1.2), which encompasses the tasks **D3** to **D5** and **D8**. SAM and maybe also S are the holders of the claim from COP. CAM validates that it actually is interacting with the holder of the claim by ascertaining that the ticket grant message was sent by this SAM. In DCAF, CAM must check if the ticket grant message stems from a SAM that is authorized by COP (see section 12.2.5), and thereby performs task **D6**.

#### **13.1.7.2. CAM Performs Task D0 for C**

C must be authorized by COP to provide input to CAM and to afterwards receive the resulting claim. How CAM validates C's authorization to send the access request is evaluated in section 13.1.2. C is authorized to receive the ticket transfer message if it sent the access request message. In DCAF, CAM must confidentiality-protect the ticket transfer message (see section 12.2.6) and must send it to the C that sent the ticket request message. DCAF therefore specifies how CAM must perform task **D0**.

#### **13.1.7.3. CAM Specifies the Client Information for C**

CAM transforms the claim that it received from COP into an authorization claim for C. The claim must reflect the OVP's decisions. In DCAF, the OVPs must have provided their AMs with authorization information entries which must contain the permissions, the constrained device's attributes and the permissions' lifetime and intended destination (see section 12.1.3) and therefore provide all information for an authorization claim. If COP specified fine-grained authorization rules for C, SAM and S, CAM must provide them in the client information (see section 12.2.6). CAM specifies the respective authorization claim for C in DCAF as

follows: the authorization rules must refer to this C, SAM and S. The server's attributes that represent the claim statement are specified by C in the access request message (see section 12.2.2) and the ticket transfer message is bound to it by the underlying security solution (see section 12.1.4). If C did not specify the server's attributes, CAM must explicitly define them in the ticket transfer message (see section 12.2.6). The client information must contain either the attributes or the authorization rules that COP specified for this C, SAM and S or both. DCAF therefore specifies how CAM must perform task **D1a** and, if the claim is an authorization claim, task **A1a**.

In DCAF, CAM must protect the confidentiality of the ticket transfer message (see section 12.2.6), and thus binds the intended destination to the claim (task **D1b** and, for an authorization claim, task **A1b**).

The claim must contain the server's keying material that represents the holder (task **D1c** and, for an authorization claim, task **A1c**). CAM must also provide freshness information and protect the integrity and confidentiality of the ticket transfer message (see section 12.2.6). DCAF therefore describes how CAM must perform task **D1** and, for authorization claims, task **A1**.

#### **13.1.7.4. C Validates the Ticket Transfer Message**

C must validate if CAM is authorized to provide the information in the ticket transfer message: it must perform the delegation tasks **D2** to **D8**. The provisioning of authorization information is a task delegation. Authorization tasks correspond to the delegation tasks with the same number, but authorization has the additional task **A7**, which C needs to perform if the claim is an authorization claim.

For tasks **D2** and **A2**, C must check if the client information in the ticket transfer message is up to date. In DCAF, C can choose from various validity options (see section 12.1.7). For validity option 1, C must rely on the timestamp and the lifetime specified by CAM to determine the remaining lifetime of the client information. CAM and C must be time-synchronized. If the **Model-Compliant Claims Assumption** holds, time information claims are securely provided by an

authorized entity. CAM must define a timestamp and a lifetime in the ticket transfer message (see section 12.2.6). DCAF specifies how C must use them to calculate the remaining lifetime for validity option 1 (see section 12.1.7). Since the ticket transfer message must be integrity-protected (see section 12.2.6), an attacker is not able to modify the timestamp and lifetime. C must remove the client information when it expires (see section 12.2.7).

For validity option 2, C must use the stored client timestamp instead of CAM's timestamp to calculate the remaining lifetime. C relies on the underlying security solution to determine that the ticket transfer message is the response to a certain access request message (see section 12.1.4). The ticket transfer message is thereby bound to the stored timestamp, and the timestamp can be used to validate this message. In DCAF, C determines if the client information is still valid using the lifetime that CAM must specify in the ticket transfer message (see section 12.2.6). The lifetime must be integrity-protected (see section 12.1.4) and therefore cannot be modified by an attacker. This option does not consider the time that passes between C sending the access request and CAM specifying the ticket transfer message as response. The remaining lifetime is therefore shorter than intended by CAM. But an attacker is not able to trick C into accepting expired client information. Validity option 2 therefore sufficiently satisfies the **claim completeness fundamental**.

If the client uses validity option 3, C checks the freshness of the client information with the help of a timeout. In DCAF, C must use the lifetime together with the timeout to determine the remaining lifetime of the client information. As in validity option 2, C is able to rely on the underlying security solution to determine that the ticket transfer message is the response to a certain access request message and retrieve the corresponding timeout. The integrity-protection of the ticket transfer message again helps against manipulations of the lifetime. Therefore, validity option 3 is also suitable to sufficiently satisfy the **claim completeness fundamental**. DCAF thereby provides validity options that even clients with a very limited ability to measure time can use for performing task **D2** and task **A2**.

DCAF's revocation mechanism may further contribute to satisfy the **claim completeness fundamental**, but the evaluation of the revocation mechanism is not in scope of this work.

For tasks **D3** and **A3**, *C* must validate that the required pieces of claim information are bound together and stem from the same *CAM*. The claim statement and intended destination are bound to the client information by the security solution. DCAF demands that the server's verifier must be specified in the client information (see section 12.2.4.2). *C* therefore performs task **D3** and **A3** by checking the integrity of the ticket. In DCAF, *C* must not accept ticket transfer messages that are not integrity-protected (see section 12.2.7).

Tasks **D4** and **A4** require that *C* validates that *CAM* is authorized to provide the client information. We already evaluated how *C* performs the authorization tasks for *CAM* in section 13.1.2. DCAF demands that *C* must only accept ticket transfer messages from an authenticated and authorized *CAM* (see section 12.2.7). DCAF therefore specifies how *C* must perform tasks **D4** and **A4**.

To perform tasks **D5** and **A5**, *C* must validate if it is the intended destination of the claim. DCAF specifies that *C* must check if the ticket transfer message is encrypted. *SAM* must encrypt the message before the integrity-protection is applied afterwards or an AEAD algorithm must be used. *C* performs tasks **D5** and **A5** by only accepting encrypted messages (see also section 12.2.7).

*C* performs task **D6** together with task **A6** for *S*. The tasks will be analyzed in section 13.1.9.

With AIF, issuers specify which actions may be performed on which resource. The resource is a context information. In DCAF, *C* must validate if the authorization information from *COP* cover a request before sending it. *C* thereby performs task **A7**.

In DCAF, *CAM* is the only claim issuer for *C*. *C* requests only a single ticket from *CAM* at a time, and can determine if a ticket transfer message is the response to a certain ticket request. *C* can therefore assume that the client information it received last is the most recent. *C* therefore performs tasks **D8** and **A8**.

If *CAM* does not specify authorization information for *S*, *C* must obtain it from somewhere else. The client information is an attribute claim in this case: it conveys keying material for the server with the requested attributes. The attributes

enable C to retrieve the corresponding authorization information, if COP specified any for a server with these attributes. The client information therefore helps C with the tasks **An1** and **A6** concerning S. C must discard tickets that do not contain authorization information if it cannot determine the server's authorization (see section 12.2.7). If the authorization information is not conveyed in the client information, C must have obtained it outside the DCAF protocol flow. If the **Model-Compliant Claims Assumption** holds, C obtained and validated the authorization information in compliance with ANAZEM. C therefore enforces the overseeing principals' decisions in these cases.

### 13.1.8. Ticket Transmission Evaluation

To establish a security association with S, C must first transfer the ticket face. Since SAM protected the ticket face, no additional actions are necessary to validate the server's authorization to receive it. When S receives the ticket face, it does not have to check C's authorization to provide it because C only relays the ticket.

The ticket face contains a claim from SAM; S must perform the delegation tasks **D2** to **D8**. If SAM includes authorization information in the ticket face, it is an authorization claim. S then must perform the authorization tasks. As described above, the authorization tasks correspond to the delegation tasks with the same number. Authorization has the additional task **A7**.

S must check if the received claim is up to date. In DCAF, the ticket face must contain the lifetime and freshness information (see section 12.2.4.1). The content of the freshness information depends on the validity option used by the server. For validity option 1, SAM generates a timestamp and a lifetime. S and SAM must be time-synchronized. If **Model-Compliant Claims Assumption** holds, S and SAM have the correct time. Since the ticket face must be integrity-protected, attackers are not able to modify timestamp and lifetime. S can therefore use validity option 1 to perform tasks **D2** and **A2**.

For validity option 2, the server generates a timestamp and a nonce and stores them. The nonce is sent with the SAM information message and must be returned

to S with the ticket face. If the nonce is not found, the ticket is discarded (see section 12.1.7). An attacker cannot trick SAM into issuing access tickets for the future since S stores the timestamp with the nonce and compares the received nonce with the stored nonce. The lifetime in the ticket face is integrity-protected and cannot be modified by an attacker. Using the timestamp and the lifetime, S is able to determine the remaining lifetime of the ticket face. Option 2 is therefore also suitable for task **D2** and **A2**.

If S uses validity option 3, a nonce and timeout are used to determine the remaining lifetime of the ticket. S must store the nonce until the timeout is reached and send the nonce in the SAM information message. S can only retrieve the correct timeout with the corresponding nonce. The remaining lifetime is calculated with the help of the timeout. Attackers cannot modify the timeout via the protocol flow. As stated above, attackers are also not able to alter the lifetime in the ticket face. Option 3 is therefore suitable for task **D2** and task **A2**.

Using a revocation mechanism may additionally help to satisfy the **claim completeness fundamental**. Analyzing the revocation mechanism in section 12.4 is not in scope of this work.

To perform tasks **D3** and **A3**, S must check if the required claim information is bound together and stems from the same entity. Section 13.1.4.3 describes how the information in the ticket is bound to each other. In DCAF, S must check the integrity and authenticity of the ticket face as described in section 12.2.8, and check if the ticket face contains all necessary information. If the ticket does not contain a destination, S must ascertain that its confidentiality is protected. S therefore performs task **D3** and **A3**.

For task **D4**, S must determine SAM's authorization. In DCAF, SOP must have specified for S which entities are authorized SAMs (section 12.1.2) and provided the necessary keying material. Authorization rules must be updated regularly. SOP and S thereby perform task **A1** and task **A2**. S must securely obtain authorization rules and keying material for SAM, which encompasses that S validates and evaluates the received claims (tasks **A3** to **A5**, **A7** and **D8**).

S must validate if the ticket face was actually issued by the authorized SAM. In DCAF, S must check that the signature or MAC of the ticket face was generated

by an authorized SAM or derive the symmetric key for the communication with C using the key that S shares with SAM (see section 12.2.8). S thereby performs task **A6** for task **D4** and **A4**.

For task **D5** and **A5**, S must check that it is the intended destination of the claim. In DCAF, S must check if the ticket face defines it as the intended recipient if the ticket face is not encrypted (see section 12.2.8). S and SAM are expected to have a common knowledge how S is identified (see section 12.1.2). DCAF therefore specifies how S must perform task **D5** and **A5**.

We will analyze how S performs task **D6** and **A6** in section 13.1.9.

AIF enables SAM to specify the resources to which the authorization rules apply and thereby defines a context for the authorization rules. In DCAF, S must check if a request is allowed according to the authorization information before it sends the response (see section 12.2.10). S thereby performs task **A7**.

In DCAF, clients can communicate with servers after obtaining access tickets from a single SAM. The tickets contain sequence numbers or timestamps that enable S to determine which ticket is the most recent (see section 12.2.4.1), and S must only keep the most recent ticket (see section 12.2.8). Thus, only a single access ticket is valid for a communication at a time. S thereby performs task **D8**.

If the ticket face does not contain authorization information, S must have obtained them in a different way. A ticket face without authorization information is an attribute claim: it must contain the client's keying material and attributes (see section 12.2.4.1). The attributes help S to retrieve the corresponding authorization information if COP specified any for a C with these attributes. In this case, the access ticket assists S in performing task **An1** and task **A6** concerning C. The authorization information must have been obtained outside the DCAF protocol flow. If **Model-Compliant Claims Assumption** holds, S obtained the authorization information in compliance with ANAZEM. If the ticket face does not contain authorization information and S cannot determine C's authorization, S must discard the ticket face (see section 12.2.8). S therefore also enforces the overseeing principals' decisions in these cases.

### 13.1.9. Trust Establishment Evaluation

C and S must validate each other's authorization before they can securely exchange application data. C uses the client information provided by CAM to validate the server's authorization while S applies the ticket face issued by SAM. How C and S perform the authorization tasks **A2** to **A5**, **A7** and **A8** is described in sections 13.1.7.4 and 13.1.8, respectively. We will now describe how C and S perform task **A6** and task **D6**.

C must validate if the S with which it communicates is the holder of the client information. In DCAF, C must ascertain that S is able to use the keying material that is specified in the client information. To do so, C must encrypt messages to S and check the integrity and authenticity of messages from S (see section 12.2.10), and, if DTLS is used, by finishing the DTLS handshake (see section 12.2.9). DCAF therefore defines how C performs task **A6** and **D6**.

On the server side, S must determine if the ticket face refers to the client it communicates with. In DCAF, S must validate that C can use keying material corresponding to the verifier in the ticket face: S must encrypt messages to C and check the integrity and authenticity of messages sent by C (see section 12.2.10). S thereby performs the tasks **A6** and **D6**.

### 13.1.10. Authorization Fundamentals Summary

For the evaluation summary, we distinguish between tasks that DCAF specifies directly and tasks that are expected but not specified. DCAF describes all security associations that are a prerequisite for the framework, and specifies the required pieces of information that must be obtained outside of the protocol flow, such as certain keying material, authorization rules and other information about the actors. DCAF specifies how the information must securely be obtained, i.e., how the respective endpoint must validate and evaluate the information and ascertain that it stems from an authorized claimant, but the details about how the data is acquired is out of scope. If tasks are performed outside of the protocol flow, they not necessarily need to be specified by the protocol. In DCAF, this applies

to the tasks **A1** to **A5**, **A7** and **A8** for authorization claims, and the tasks **D1** to **D5** and task **D8** for attribute claims. But, even if certain information is expected to already have been obtained outside of the protocol flow, the protocol must explicitly specify how endpoints check if these pieces of information apply to the current communication to comply with the **reference monitor fundamental**. DCAF describes how C and CAM, CAM and SAM, and S and SAM perform the tasks **A6** and **D6** for each other, respectively, and how the communication between these actors must be protected as a result.

The main focus of DCAF is to describe how the authentication and authorization information is securely provided by the authorization manager to the constrained device. DCAF describes in detail how SAM must prepare and protect the attribute claim about S for CAM and the authorization claim concerning C for S, and how CAM must generate and protect the attribute claim about C for SAM and the authorization claim concerning S for C. All required authorization and delegation tasks are covered by DCAF.

On the server side, DCAF describes how SAM obtains and validates claims about C from CAM. Also, DCAF specifies how S performs the delegation and authorization tasks for obtaining and validating the ticket face.

On the client side, DCAF describes how CAM obtains and validates the attribute claim from SAM containing the server's keying material. Finally, DCAF explains how C performs the delegation and authorization tasks for obtaining and validating the client information.

Summarizing, the analysis shows that DCAF specifies all authorization and delegation tasks that are necessary to securely provide S and C with the authenticated authorization information that they require to authenticate each other and validate each other's authorization. The devices are thereby enabled to participate in the protection of their overseeing principals' security objectives.

The analysis shows that our model enables protocol designers to specify how the necessary tasks are performed inside and outside the protocol. By encouraging protocol designers to consider how each task is performed in their protocol, the risk of accidentally omitting important details is reduced. Interfaces with other

solutions can be specified more clearly: the requirements on additional protocols such as the underlying security protocol can be explicitly defined, which enables implementers to combine solutions as necessary. ANAZEM can thus reduce the risk of gaps and vulnerabilities in the overall security solution.

Conducting a protocol analysis with ANAZEM requires an understanding of the roles that the actors perform in the protocol flow, e.g., who is a claim issuer for whom, and which tasks therefore need to be performed. Also, the analyst must determine if a certain measure sufficiently satisfies the fundamentals. Simplifying this process to make the model easier to apply is a topic for future work. The model is a useful complement to formal methods because it assists analysts with identifying the security properties of the protocol that can then be used for formal verification. ANAZEM may even be assist with improving formal verification tools if security properties proposed by the model are missing in the tools. Examining and—if necessary—enhancing the tools is an interesting topic for future work.

## 13.2. Quantitative Evaluation

In the previous sections we performed a qualitative evaluation of the DCAF protocol. In the following, we examine quantitative aspects by comparing DCAF with the ACE framework regarding message sizes and implementation characteristics.

Section 13.2.1 analyzes DCAF and ACE framework messages and compares their sizes. In section 13.2.2, we examine code and data sizes of DCAF and the ACE DTLS profile.

### 13.2.1. Quantity of Network Communication

Sending and transmitting messages is expensive for constrained devices (see section 3.1.3). The authorization solution should therefore minimize the number of messages and reduce the message sizes where possible. In the following section

we will analyze DCAF messages and compare them to the ACE framework and the DTLS profile.

We will first compare the data that is exchanged between C and SAM in the ACE framework with the DCAF protocol flow for a combined client and CAM (see also section C.2). Afterwards, we will analyze the messages that C has to send in DCAF and the ACE framework. We do not include the unauthorized access request message and the SAM information message in the analysis since they are optional. We only consider the size of the protocol messages without the underlying security solution.

#### 13.2.1.1. Ticket and Token Request Message

The token request message in the ACE framework must contain the `cnonce` parameter if it was provided by the server (draft-ietf-ace-oauth-authz-35, p. 22). Additionally, the client may specify the grant type, the audience, i.e., the intended server, the scope and the client's keying material in the token request message (draft-ietf-ace-oauth-params-13, p. 3). These fields are optional. C may also include an empty `ace_profile` parameter (draft-ietf-ace-oauth-authz-35, p. 23).

The DCAF ticket request message contains the requested resources as the scope, and the nonces if the server provided any in the SAM information message (see section 12.2.3). C/CAM may also state C's attributes or keying material in the ticket request message. It then must specify a timestamp with the creation time of the message and a lifetime for the included information, and may include a sequence number for revocation. The ticket request message must contain the server's attributes as the audience. Table 13.1 shows the respective parameters of the requests. The CBOR labels in square brackets are not (yet) registered. We do not specify the CBOR labels of the keying material since they depend on the algorithm that is used and would make the table unnecessarily complex.

As we can see, most of the parameters in both messages are optional. A minimal token request message is empty. A minimal ticket request has 3 bytes of fixed data to which the variable data for the server's attributes (the audience) must be added.

DCAF	ACE Framework	CBOR Label
Required		
audience	–	[5]
Optional		
server nonce (cnonce)	server nonce	[39]
scope	scope	[9]
C's keying material	C's keying material	
client desc. (sub)	–	2
sequence number (cti)	–	7
timestamp (iat)	–	6
lifetime (expires_in)	–	[2]
–	ace profile	[38]
–	audience	[5]
–	grant type	[33]

Table 13.1.: Parameters of Ticket Request and Token Request Message

Omitting the audience from the token request message may lead to vulnerabilities. *C* may receive access tokens for the wrong *S*, and then may not be able to communicate with the the intended server. Even worse, if *C* receives keying material for the server from *SAM*, *C* may communicate with the wrong server without noticing it. If a security solution does only rely on the server to enforce the overseeing principals' security decisions, *C* may reveal confidential data and accept data from an unauthorized server. As in DCAF, the audience should therefore be required in the token request message. The message then has 3 bytes of fixed data to which the size of the audience value must be added.

Another main difference between the ACE framework and DCAF is that in the latter, *C/CAM* must specify a timestamp and lifetime if it provides keying material for *C* to *SAM*. Without the lifetime, *SAM* may issue access tickets and keying material for the communication between *C* and *S* with a validity period that exceeds the validity period of the keying material that *CAM* provided. Since this is also true for the ACE framework, the token request message should be required to convey when *C*'s keying material expires. That a DCAF ticket request may be slightly larger than an ACE token request is therefore only due to DCAF offering more security. Apart from that, the ace profile parameter that is specific for the ACE framework makes the token request message 4 bytes larger. The grant type adds, where it is needed, 1 byte of fixed data and additionally data for its value.

### 13.2.1.2. Ticket Face and Access Token

The contents of the ACE framework access token are not clearly defined. The token may comprise the issuer, the intended destination, i.e., the audience, the scope, and the subject, i.e., the holder (draft-ietf-ace-oauth-authz-35, p 38). It may also contain the keying material for the communication with the client, the server nonce, and the ace profile (`cnonce`) (draft-ietf-ace-oauth-authz-35, p. ). To specify the validity, the token may provide the expiration time, the date after which the token is to be used (not before), the token lifetime and the sequence number of the token.

The DCAF ticket face must contain authorization information or *C*'s attributes, keying material for the communication with *C* or a key derivation method, a nonce or timestamp, the lifetime of the ticket, and the sequence number of the ticket or the generation time (see section 12.2.4.1). The ticket face may additionally specify the intended recipient, and the deprecated ticket sequence number.

DCAF	ACE Framework	CBOR Label
Required		
ticket face	access token	[1]
lifetime (expires_in)	–	[17]
Optional		
audience (aud)	audience	3
scope	scope	[9]
keying material	keying material	
server nonce (cnonce)	server nonce	[39]
sequence number (cti)	sequence number	7
generation time	generation time	6
deprecated ticket seq. nr.	–	[19]
–	lifetime (exi)	[40]
–	expiration time	4
–	not before	5
–	issuer	1
–	sub	2
–	ace profile	[38]

Table 13.2.: Parameters of the Ticket Face and Access Token

Since the ACE framework does not seem to require certain parameters, a minimal access token would be empty. Since this would not be very useful, we assume

that the token at least contains a key identifier in a `cnf` parameter. The DTLS profile specifies how key derivation may be used to derive the session key from this information (draft-ietf-ace-dtls-authorize-13, pp. 13–14). To achieve sufficient variation in the derived keys, the DTLS profile recommends that the access token comprises a generation time and either the expiration time or a lifetime (draft-ietf-ace-dtls-authorize-13, p. 14). A minimal access token then has 12 bytes of fixed data to which the size of the kid, of the generation time, and of the expiration time or lifetime must be added. The example below shows a minimal access token with a symmetric key including the respective CBOR labels.

```
{
  /access token/ 1: {
    /iat/ 6: 1563451500,
    /exp/ 4: 1563453000,
    /scope/ 9: ["/s/tempC", 5],
    /cnf/ 8: {
      /COSE_Key/ 1: {
        /kty/ 1: /symmetric/ 4,
        /kid/ 2: h'e7509a8c032f3bc2a8df1df476f8ef03',
      }
    }
  }
}
```

A symmetric key is not suitable to retrieve formerly stored authorization information about C (see also section 12.1.5). We therefore assume that the access token either comprises a scope or that S uses token introspection to obtain C's permissions. The scope adds 1 byte of fixed data and the size of the scope value to the access token. For introspection additional messages are needed. The introspection request that S sends to SAM must at least comprise the access token (draft-ietf-ace-oauth-authz-35, p. 32). The introspection response must at least convey if the token is currently active. To be useful for the server in our example, the response must also provide the scope. Token introspection therefore increases the amount of data that must be exchanged. The approach with the minimal data requirements is therefore to include the scope in the access token. The server must be able to determine which access token is the most recent. The server may use the generation time for this purpose to avoid additional overhead (e.g., by

also specifying the `cti` parameter). The access token then has 13 bytes of fixed data to which the size of the kid, generation time, lifetime or expiration time and scope must be added.

A minimal DCAF ticket face with the lifetime, the scope, the generation time and key derivation information has 13 bytes of data to which the size of the lifetime, generation time, scope and kid must be added. DCAF validity options require the generation time or a nonce together with the lifetime of the ticket to be present in the ticket face. Access tokens in the ACE framework must contain an expiration date or a lifetime and sequence number. Without the sequence number, the server is not able to determine which ticket is the most recent. In DCAF, the generation time may be used for this purpose. The validity requirements are therefore similar. The ACE access token optionally may comprise the issuer, subject and ace profile.

### 13.2.1.3. Ticket and Token Grant Message

We assume that the ACE token grant message must contain an access token. The ticket and the token grant message therefore consist of the access token and the client information (if present). The analysis of the client information is included in the analysis below.

Besides the access token, the token grant message may contain the token type, the lifetime, the scope, an OAuth 2.0 refresh token, and the OAuth 2.0 state parameter (draft-ietf-ace-oauth-authz-35, p. 26). It may also contain the server's RPK if asymmetric cryptography is used between C and S, and must contain the symmetric key for a symmetric solution. SAM must also include an `ace_profile` parameter if the client specified an empty ace profile parameter in the token request.

A DCAF ticket grant message may contain the server's keying material and then must provide the generation time of the ticket. It may additionally comprise the ticket lifetime, the sequence number of the ticket, the authorization information, i.e., the scope, and the sequence number of the ticket that is replaced with this

DCAF	ACE Framework	CBOR Label
Required		
ticket face	access token	[1]
Optional		
lifetime (expires_in)	lifetime	[2]
the server's keying material	the server's keying material	
scope	scope	[9]
sequence number (cti)	–	7
generation time (iat)	–	6
–	ace profile	[38]
–	token type	[34]
–	refresh token	[37]

Table 13.3.: Parameters of the Ticket Grant and Token Grant Message

ticket. The ticket grant message also contains the access ticket. Table 13.3 shows the contents of the respective grant messages.

The minimal size of the ticket and token grant message therefore equals the minimal size of the ticket face or access token (see section 13.2.1.2).

The client requires validity information to determine the lifetime of the keying material. We therefore assume that SAM provides validity information if it provides keying material for C. The ACE framework only provides a lifetime. Without an additional mechanism, the client can only estimate how long the keying material is still valid. C/CAM may e.g., start a timer when it sends the token request, and when the token grant arrives subtract the passed time from the lifetime as described for DCAF validity option 2 (see section 12.1.7.2). Minimal client information with symmetric keying material then has 11 bytes of fixed data to which the size of the lifetime and the keying material must be added. A minimal token grant message consists of access token and client information and therefore has 24 bytes of fixed data. The `ace_profile` parameter may add 3 bytes to the size of the token grant message, the `token type` another 3 bytes where necessary.

In DCAF, SAM must specify the generation time if it provides client information. The lifetime may be conveyed with the CoAP Max-Age option. Providing the generation time with the client information is more comfortable for C/CAM than subtracting the time that has passed since the ticket request was sent from the lifetime. Minimal client information specified by SAM has 11 bytes of fixed data

to which the size of the generation time and the keying material must be added. The ticket grant message has 24 bytes. Depending on the lifetime, specifying the generation time instead of the lifetime may slightly increase the message size. The DCAF sequence number in the client information is only needed for revocation. The ACE framework document does not specify a revocation mechanism.

#### 13.2.1.4. Comparison of Four-Corner and Three-Corner Architecture

DCAF implements the four-corner architecture where the client has its own authorization manager. We will now compare the messages that a constrained client must be able to send and receive with the respective messages in the ACE framework.

A minimal access request message that C sends to CAM is empty. In this case, CAM determines SAM's address, the server's attributes (the audience) and the resources that C is supposed to access for C. Table 13.4 shows a comparison of the access request and the token request.

DCAF	ACE Framework	CBOR Label
Required		
–	ace profile	[38]
Optional		
cnonce	cnonce	[39]
scope	scope	[9]
audience	audience	[5]
SAM's URI (iss)	–	[1]
–	C's keying material	
–	grant type	[33]

Table 13.4.: Parameters of Access Request and Token Request

The ticket transfer message contains the access ticket which has at least 13 bytes of fixed data. The message may additionally provide client information with the keying material for the communication between C and S, authorization rules defined by CAM, the sequence number of the client information and the generation time.

DCAF	ACE Framework	CBOR Label
Required		
ticket face	access token	[1]
Optional		
lifetime (expires_in)	lifetime	[2]
the server's keying material	the server's keying material	
cscope	–	[21]
sequence number (cti)	–	7
generation time (iat)	–	6
S' attributes (aud)	–	5
–	scope	[9]
–	ace profile	[38]
–	token type	[34]

Table 13.5.: Parameters of the Ticket Transmission and Token Grant Message

A minimal ticket transfer message only contains the access ticket. A minimal ticket transfer message containing keying material does not require additional information if the ticket lifetime is specified in the CoAP Max-Age Option. A minimal client information with only a symmetric key has 8 bytes of fixed data to which the size of the keying material must be added. The sequence number is only required for revocation. The generation time is only required for validity option 1.

Message	DCAF	ACE Framework	DCAF with CAM
Request Message	3	3	0
Access Ticket/Token	13	13	13
Grant Message with CI	24	24	21
Result C-SAM without CI	16	16	16
Result C-SAM with CI	27	27	21

Table 13.6.: Comparison of Minimal Message Sizes in Bytes

Table 13.6 shows the minimal message sizes in bytes that C and S must handle. We distinguish DCAF with combined C/CAM, the ACE framework, and DCAF with a separate CAM. SAM and S only exchange the access ticket. The data transmitted between the client side and SAM is calculated by adding the respective request message size to the grant message size. An exception is DCAF with a separate CAM but without client information (CI). If the client sends an empty access request to CAM, the ticket transfer message must contain client informa-

tion that informs the client with which server it is supposed to communicate. CAM can only omit the client information in the ticket grant message if the client sent at least the intended audience in the access request message. Therefore, C and CAM must at least exchange 16 bytes of data. As we can see, the minimal sizes of messages that are handled by the servers are the same for DCAF and the ACE framework. Also, minimal message sizes equal for the combined C/CAM and the ACE client. A client with a separate CAM has a reduced minimal message size when the ticket grant message contains client information.

In addition to the messages that are exchanged between C and the respective AM, C also must know with which server it is supposed to communicate, and must discover the server's AM. In the ACE framework, the client must determine correct audience values for the server (draft-ietf-ace-oauth-Authz-35, p. 48). C also must determine at which URI it is supposed to request access tokens for S (draft-ietf-ace-oauth-Authz-35, p. 16). C may either dynamically obtain the audience and SAM's URI or be provisioned with the respective values.

In DCAF, a client may leave the discovery of the server and the server's AM to CAM (see section 12.1.1 and 12.2.1, respectively). The client then receives all information that is necessary for the communication with the server from CAM in the normal DCAF protocol flow. An equally dynamic approach for the ACE framework would significantly increase the amount of data that clients need to send and receive.

### 13.2.2. Memory Footprint

In the following sections, we analyze the overall memory requirements of our DCAF implementation (see also section 12.8). Section 13.2.2.1 examines DCAF's non-volatile memory usage. We determine maximum stack and heap sizes in section 13.2.2.2. Our implementation uses CoAP as transfer protocol and DTLS as underlying security protocol. Section 13.2.2.3 analyzes the memory footprint of these protocols. In section 13.2.2.4, we compare the memory requirements of DCAF with those of the DTLS profile.

### 13.2.2.1. Code Size

To estimate DCAF's non-volatile memory requirements, we list the output of the RIOT makefile for the example DCAF server application. For this analysis, we use gcc version 8.3.1., RIOT version 2020.07-devel-440-gd7622-dcaf, tinydtls version 0.8.9-8c94583, libcoap version 4.2.0-develop-0e55b42f, and libdcaf version 0.1.0-458445d on an STM32 Nucleo F401 board.

```
$ make BOARD=nucleo-f401re info-objsize \
| (head -1 -; sed -ne "s/(ex .*\(dcaf\.a\))/;/\
/main\.o/{ s/(ex .*\(dcaf_s\.a\))/}; T; \
s,/\/\([^\/]*\/\)*, ,; p" )
text  data  bss   dec   hex  filename
2413   0    596  3009  bc1  dcaf.o
1992   4     88  2084  824  anybor.o
1317  36     0  1353  549  cose.o
1152   0     0  1152  480  dcaf_transaction.o
 835   0     0   835  343  aif.o
 323  140    32   495  1ef  main.o
 400   0     0   400  190  dcaf_key.o
 326   0     0   326  146  dcaf_coap.o
 272   0     0   272  110  dcaf_crypto_tinydtls.o
 172   0     0   172   ac  dcaf_utf8.o
 164   0     0   164  a4  dcaf_address.o
 146   0     0   146   92  dcaf_mem.o
  36   0     4    40   28  dcaf_prng.o
  30   1     0    31  1f  dcaf_debug.o
  24   0     0    24  18  dcaf_optlist.o
```

As we can see, the core DCAF logic in `dcaf.o` consists of 2413 bytes of code that needs to be stored in non-volatile memory. The DCAF state machine in `dcaf_transaction.o` adds another 1152 bytes. As a result, the complete DCAF implementation on an ARM Cortex M4 takes up 9602 bytes (about 9.4 KiB) of code. The other components are mainly the COSE implementation and wrappers for the libcoap integration. Our server implementation uses the light-weight CBOR library `anybor` that is developed by Olaf Bergmann. In `anybor`, data structures are only kept in memory while they are parsed during the processing of a DCAF message.

Below, we analyze DCAF on an ESP32 board. We determine the code and data sizes with the Espressif<sup>1</sup> build tool `idf.py` (ESP-IDF v4.2-dev-1660-g7d7521367 with gcc version 5.2.0). The listing below shows the output for the DCAF server application.

```
$ idf.py size-files | sed -ne '/Object/p; /dcaf/p; /cose/p;\
/aif/p; /anybor/p' | cut -c 1-24,30-43,73-
```

Object File	.data	& .bss	code	& rodata	Total
dcaf.c.o	0	592	2123	3	2718
anybor.c.o	4	88	1945	0	2037
cose.c.o	72	0	1254	17	1343
dcaf-server.c.o	4	0	679	423	1106
dcaf_key.c.o	0	0	366	0	366
aif.c.o	0	0	365	0	365
dcaf_crypto_mbedtls.c.o	0	0	290	16	306
dcaf_mem.c.o	0	0	213	36	249
dcaf_address.c.o	0	0	171	17	188
dcaf_utf8.c.o	0	0	98	0	98
dcaf_coap.c.o	0	0	93	0	93
dcaf_debug.c.o	0	4	29	0	33

As we can see, data and code sizes are similar on both platforms. The `dcaf-server.c.o` corresponds to the `main.o` of the RIOT port. The complete DCAF server implementation on an ESP32 board has 7626 bytes (about 7.4 KiB) of code.

Below, we analyze the code and data size for a typical constrained DCAF client. In its minimal configuration, it sends an empty access request to its CAM to receive all necessary information for the communication with the server.

```
$ idf.py size-files | sed -ne '/Object/p; /dcaf/p; /cose/p;\
/aif/p; /anybor/p' | cut -c 1-24,30-43,73-
```

Object File	.data	& .bss	code	& rodata	Total
dcaf.c.o	0	0	2164	27	2191
anybor.c.o	4	72	1536	0	1612
dcaf_transaction.c.o	0	0	1115	0	1115
dcaf-client.c.o	4	8	583	317	912
dcaf_key.c.o	0	0	366	0	366
dcaf_coap.c.o	0	0	326	0	326

<sup>1</sup><https://www.espressif.com/>

dcaf_address.c.o	0	0	241	24	265
dcaf_mem.c.o	0	0	213	36	249
dcaf_utf8.c.o	0	0	115	0	115
dcaf_debug.c.o	0	4	29	0	33
dcaf_prng.c.o	4	0	28	0	32
aif.c.o	0	0	0	0	0

As we can see, the DCAF client has 6716 bytes (about 6.6 KiB) of code and 500 bytes of data. It therefore has roughly the same size as a constrained server.

### 13.2.2.2. Stack and Data Usage

To determine DCAF's stack usage, we analyze the DCAF message exchange on an ESP32 board. We use the Espressif SDK FreeRTOS function `uxTaskGetStackHighWaterMark()` to determine the stack size. We call this function at the beginning of the DCAF server thread and then again after each DCAF transaction is completed, i.e., after the response to a received request is sent. The function then delivers the maximum stack size since the start of the server thread. We thus determined that the maximum stack size for the handling of DCAF messages is 2624 bytes.

For the dynamic heap usage, we analyze the data objects for which memory is allocated in DCAF. For each symmetric key that S shares with SAM, the server needs 120 bytes (including memory for the kid and SAM's address). The DCAF context, containing configuration data and transaction state for the server, requires 60 bytes. For each relationship with a client, the server needs 28 bytes for the ticket and 44 bytes for the AIF structure. The client's keying material requires another 80 bytes (32 bytes for the key and 32 bytes for the kid). Summarizing, this results in roughly 150 bytes for each ticket. A server with a single SAM that communicates with a single client therefore needs 332 bytes.

We calculate the overall data memory required by the DCAF server for the ESP32 board by adding the heap size to the data, bss and rodata segment sizes. For a minimal server with a single SAM communicating with a single client, the overall required data memory thus adds up to  $332 + 1276 = 1608$  bytes (about 1.6 KiB).

### 13.2.2.3. CoAP and DTLS Memory Footprint

Our DCAF implementation uses CoAP as transfer protocol and DTLS as the underlying security protocol. The code and data size requirements of these protocols need to be considered to determine the lower bounds of the device's capabilities. DCAF can be used with other underlying security solutions as long as those meet DCAF's requirements for underlying protocols (see section 12.1.4).

Below, we analyze the code and data size of the libcoap and MbedTLS<sup>2</sup> implementations on ESP32. They were built only for the use with symmetric keying material and only for DTLS, and without debug output. We again use the command `idf.py size-files` to determine the code and data size of the libcoap implementation:

```
$ idf.py size-files | cut -c 1-24,30-43,73-
  Object File .data & .bss code & rodata Total
  address.c.o      0      0   172      0    172
coap_mbedtls.c.o   4   1476  1578      16   3074
  net.c.o          8   3092  5371      17   8488
  coap_io.c.o      4    280  3253      0   3537
coap_session.c.o  0      0  2809      4   2813
  resource.c.o     8      0  4213     139  4360
  pdu.c.o         192     0  1485     344  2021
  uri.c.o          0      0  1632      26  1658
  coap_time.c.o    0      8   204      0   212
  str.c.o          0      0   101      0   101
  encode.c.o       0      0   144      0   144
coap_hashkey.c.o  0      0    65      0    65
  mem.c.o          0      0    22      0    22
  block.c.o        0      0   989      0   989
  subscribe.c.o    0      0    26      0    26
```

The overall code size of the libcoap implementation therefore is 22064 bytes. The size of the static data, calculated from the `.data`, `.bss`, `.rodata` columns, is 5618 bytes.

The data and code size of the MbedTLS implementation is listed below. We do not include modules that are exclusively used for Wi-Fi operations in our examination.

<sup>2</sup><https://tls.mbed.org>

```

$idf.py size-files | cut -c 1-24,30-43,73-
  Object File .data & .bss code & rodata Total
  ssl_tls.c.o 52 0 14527 347 14926
  ssl_srv.c.o 0 0 5165 76 5241
ssl_ciphersuites.c.o 0 136 228 3210 3574
  esp_sha256.c.o 0 0 2978 320 3298
  aes-internal.c.o 0 0 794 2314 3108
  md5.c.o 0 0 2721 0 2721
  md5-internal.c.o 0 0 2374 0 2374
  cipher.c.o 0 0 1780 20 1800
  cipher_wrap.c.o 0 0 365 1165 1530
aes-internal-dec.c.o 0 0 1340 0 1340
  aes.c.o 8 0 1236 0 1244
  ccm.c.o 0 0 1166 0 1166
aes-internal-enc.c.o 0 0 1088 0 1088
  aes-ccm.c.o 0 0 777 0 777
  aes-ccm.c.o 0 0 777 0 777
  sha256.c.o 0 0 544 0 544
  ssl_cookie.c.o 0 0 484 0 484
  md5.c.o 0 0 299 0 299
  aes-unwrap.c.o 0 0 253 0 253
  aes-wrap.c.o 0 0 177 0 177
sha256-internal.c.o 0 0 87 0 87
  platform.c.o 8 0 43 0 51
  sha256.c.o 0 0 51 0 51
  platform_util.c.o 4 0 28 0 32
  esp_mem.c.o 0 0 30 0 30
  esp_hardware.c.o 0 0 19 0 19

```

The listing shows that MbedTLS as used for our DCAF server has a code size of 39331 bytes and a data size of 7660 bytes. In summary, the MbedTLS and libcoap implementation have a code size of 61395 bytes (about 60 KiB) and a data size of 13278 bytes (about 13 KiB). These implementations contain many features that are not strictly necessary for the deployment on constrained devices. Data and code size can be further reduced by using implementations that were specifically optimized for a small memory footprint such as `tinydtls`<sup>3</sup> and `Erbium`<sup>4</sup>.

Below, we analyze the `Erbium` implementation included in the Cetic 6LoWPAN border router (6LBR) solution<sup>5</sup> for the Contiki operating system. We only con-

<sup>3</sup><https://projects.eclipse.org/projects/iot.tinydtls>

<sup>4</sup><https://github.com/contiki-ng/contiki-ng/tree/develop/os/net/app-layer/coap>

<sup>5</sup><https://cetic.github.io/6lbr/>

sider the server side and do not include CoAP observe and separate responses in our analysis. We use the GCC compiler toolchain for ARM embedded platforms<sup>6</sup> to determine the code and data sizes.

```
$ arm-none-eabi-size -d -B *.o |tail -n+2
  text  data  bss   dec   hex  filename
   741   16    0    757   2f5  er-coap-dtls.o
  1886   32   350   2268   8dc  er-coap-engine.o
  1807   32    4   1843   733  resources.o
   205    0    0    205    cd  er-coap-block1.o
  3323    4    5   3332   d04  er-coap.o
   690   40    0    730   2da  er-coap-res-well-known-core.o
   436   12  1552   2000   7d0  er-coap-transactions.o
    82    0    0     82    52  er-coap-udp.o
  1482    0    0   1482   5ca  key-token-store.o
   548   16    9    573   23d  rest-engine.o
```

The output shows that the Erbium implementation has a code size of 11200 bytes and a data size of 2072 bytes.

The Cetic 6LBR solution uses tinydtls as the DTLS implementation. The code and data sizes are listed below:

```
$ arm-none-eabi-size -d -B *.o |tail -n+2
  text  data  bss   dec   hex  filename
  1045    0    4   1049   419  dtls_ccm.o
  1089   24   519   1632   660  dtls_crypto.o
 13634   16   248  13898  364a  dtls.o
    32    0    0     32    20  dtls_time.o
   240   12   507   759   2f7  hmac.o
   337   12   675  1024   400  netq.o
   184   12    81   277   115  peer.o
  6640    0    0   6640  19f0  rijndael.o
   144    0    0    144    90  session.o
  1353    0    0   1353   549  sha2.o
```

Summarizing, the tinydtls implementation has 24698 bytes of program code and a data size of 2110 bytes. Together, the CoAP and DTLS implementations of the 6LBR solution have a code size of 35898 bytes (about 35 KiB) and a data size of 4137 bytes (about 4 KiB).

<sup>6</sup><https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm>

### 13.2.2.4. Code Size Comparison

In the following, we compare our DCAF server implementation with the DTLS profile implementation of the Carnegie-Mellon University<sup>7</sup>. Their server implementation is based on the Cetic 6LBR solution which we have analyzed in the previous section. We build the code for the target cc2538dk using the script that is provided by the implementation.

To estimate the code size for the DTLS profile implementation, we analyzed the changes of the 6LBR implementation before and after the DTLS profile was included (commit ad2be5f5d from 2018-21-12 and commit db559061a from 2019-04-12). The command `arm-none-eabi-size` provides the text and data sizes of the examined files. The output for the new files is listed below.

```
$ arm-none-eabi-size -d -B *.o |tail -n+2 \
|sort -k6
  text      data      bss      dec      hex filename
  1254         0         0     1254     4e6 ace-cwt.o
    546         0         0      546     222 cbor-encode.o
    544        40         0      584     248 er-coap-res-authz-info.o
    996        40         0     1036     40c er-coap-res-pair.o
    564        40         4      608     260 er-coap-res-lock.o
   3964        16        88     4068     fe4 revocation.o
    674         0         8      682     2aa dtls_helpers.o
```

Additionally, we analyze existing files for and after the DTLS profile integration with the above command. We inspected changed files to determine if they now include DTLS profile code. We then manually calculate the delta for these files. The results are given below.

text	data	bss	dec	filename
1824	0	196	2020	er-coap-dtls.o
563	0	0	563	er-coap-engine.o
209	-32	52	229	resources.o

In summary, the code size of the DTLS profile is 11138 bytes (about 11 KiB) and the data size 452 bytes. Data and Code sizes are therefore larger than those of the DCAF server implementation. Also, code reviews and tests reveal that the DCAF implementation makes some more robustness checks.

<sup>7</sup><https://github.com/SEI-TAS/ace-6lbr>

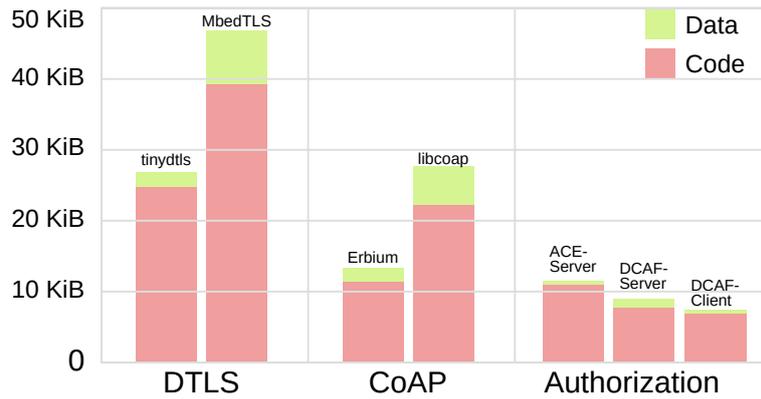


Figure 13.1.: Comparison of Code Sizes

Figure 13.1 shows a comparison of code and data sizes between DCAF and the ACE framework, and the examined DTLS and CoAP implementations.

### 13.2.3. Quantitative Evaluation Summary

The network communication analysis shows that the data sizes of the ACE framework and DCAF are very similar. In summary, the minimal amount of data that needs to be transmitted between C and SAM is—with comparable security—equal in the ACE framework and in DCAF with a combined C/CAM. If C communicates with its own CAM instead of SAM, C may send an empty access request to CAM and may receive a simplified response. The amount of data that C needs to send and transmit is then reduced.

The differences between the DCAF and ACE framework message sizes are only small, but DCAF is able to convey significantly more information in its messages: by introducing CAM, the security objectives of the client’s overseeing principals can be enforced. CAM can either directly provide COP’s authorization rules to the client or simply prevent clients from communicating with SAMs and servers that are not authorized by COP. The provisioning of COP’s authorization information is therefore directly integrated in the DCAF protocol flow. In contrast, clients using the ACE framework need an additional mechanism that provides them with their overseeing principals authorization decisions concerning SAM and S. In DCAF, CAM may also discover the server and the server’s AM for C,

which further reduces the effort for the client. The overall amount of messages that clients need to send and receive can therefore be strongly reduced.

A constrained device that is already equipped with CoAP and DTLS will likely have no difficulties to additionally use DCAF. The DCAF server implementation adds 7.4 KiB of code. A minimal server that communicates with only a single client has data requirements of about 1.6 KiB. For each additional client, another 0.15 KiB must be added. In contrast, optimized CoAP and DTLS implementations together require roughly 35 KiB of code and about 4 KiB of data. The size of the DCAF implementation is of little significance in comparison.

### 13.3. Requirements Evaluation

In part I, we introduced requirements that an authorization solution for the IoT must meet. In this section, we will analyze if and how DCAF meets these requirements. DCAF focuses on the operational phase; the secure commissioning and decommissioning of devices is out of scope. We will therefore not discuss lifecycle requirements in this section. How DCAF can be used to provide security in the whole lifecycle of a device is discussed in chapter 14.

In chapter 3, we give an overview of the characteristics of constrained devices. In section 13.3.1, we analyze if DCAF supports devices with hardware limitations that are typical for constrained environments (see also section 3.6). An important concept for the big web is REST (see section 3.4). How our solution meets REST requirements is analyzed in section 13.3.2. Chapter 4 introduces use cases for constrained environments from which we derived various types of requirements. In section 13.3.3, we will examine if DCAF is suitable for constrained environments considering the environment requirements defined in section 4.3.1. The authorization rules requirements from section 4.3.2 are analyzed in section 13.3.4. The requirements for updating these rules (see section 4.3.3) are the topic of section 13.3.5.

### 13.3.1. Hardware Limitation Requirements

As described in chapter 3, constrained devices come in various sizes and shapes and therefore may have various hardware limitations. An authorization solution must consider the various dimensions of possible constraints (see also section 3.6).

DCAF was developed for the use with CoAP and therefore meets the **CoAP Req**. Clients may register at the server to receive CoAP observe notifications after they obtained an access ticket (**Observe Req**). Also, we showed how the participating devices may use observe to receive DCAF revocation messages.

To meet the **Class 1 Req** the whole protocol stack must not exceed the data and code size capabilities of class 1 devices. DCAF can be run over protocols that were specifically designed for constrained environments such as 6LoWPAN and 802.15.4. Also, DCAF is designed to relieve both the client and the server from difficult security tasks. As described in section 13.2.2.1, the DCAF prototype implementation has about 7.4 KiB of code. Optimized CoAP and DTLS implementations additionally require non-volatile memory of 35 KiB (see section 13.2.2.3). On a class 1 device that has a storage size of approximately 100 KiB, about 57.6 KiB remain for network stack, operating system and the actual application. This may be sufficient for a simple constrained device application. A minimal DCAF server has data size requirements of about 1.6 KiB. For CoAP and DTLS, about 4 KiB of data are additionally needed. While a class 1 device with only 10 KiB of RAM will likely be able to deploy DCAF, it will be hard-pressed to use DTLS and CoAP on top of its actual application, even if it is possible to further reduce the respective implementations. We estimate the lower bound for available RAM on constrained devices to be 20 KiB in order to enable authenticated authorized communication. Such a device still can be considered to rather belong in the category of class 1 devices since it requires much less than the roughly 50 KiB of RAM that are expected to be available on class 2 devices.

Simplifying architecture components and decentralizing interactions improves the scalability (Fielding 2000, p. 32). DCAF clients are supported by their authorization managers and can therefore be comparatively simple. Also, DCAF

decentralizes the authorization manager service by separating CAM and SAM. The constrained nodes only need to store the keying material for their own authorization managers and for the endpoints with which they are currently communicating. Authentication and authorization information can dynamically be obtained as needed. DCAF thereby meets the **Scalability Req.**

DCAF introduces three validity options that enable the constrained devices to check the validity of authentication and authorization information. It provides a solution for constrained devices that have no real-time clock and are not time-synchronized with their authorization managers. All validity options impede attacks that aim at prolonging the validity period of access tickets: attackers cannot extend the lifetime of a ticket by withholding them. The validity options are not only defined for constrained servers, but also for constrained clients. DCAF also enables constrained devices to determine which access ticket is the most recent (**Clocks Req.**).

Reducing the amount of messages helps to save energy, since sending and receiving data is particularly expensive. DCAF combines the distribution of authentication data and authorization information. Also, client and server are provided with the required information within the same protocol flow. To spare the constrained endpoints costly time synchronization, DCAF introduces a simplified time synchronization that is integrated into the protocol flow (see section 12.1.7).

As we have shown in section 13.2.1, the sizes of minimal DCAF messages are slightly smaller than ACE framework messages with comparable security. The client information increases the size of the ticket transfer message that CAM sends to C, especially if it contains authorization information concerning S. But this information is required by clients that are not able to determine the server's authorization by themselves. Since the client information is optional, it can be omitted if the client already knows the server's keying material and authorization. As the authorization fundamentals evaluation shows, the information in the ticket face and the client information is required by the constrained devices to communicate securely. They do not contain unnecessary overhead and thus comply with the **Energy Req.** As we have shown in section 13.2.1.4, introducing an authorization manager on the client side reduces the amount of data that the

client needs to send and receive. DCAF therefore considers the potential energy constraints of the endpoints.

### **13.3.2. REST**

DCAF is designed for the use with CoAP which implements REST for constrained environments. Additionally, DCAF supports REST concepts where possible. It fully implements the four-corner architecture that follows the REST principles as described in section 9.4.1. In contrast to the ACE framework, constrained devices in DCAF are only closely coupled to their own authorization managers in the same security domain (see also section 12.7.2). Constrained devices do not have a relationship with foreign AMs. The loose coupling enables permissionless innovation and separate evolution of components. Unlike the ACE framework, DCAF therefore facilitates the communication over organization boundaries, which is an important factor for a successful Web of Things.

DCAF complies with the stateless server constraint where possible (see also section 12.7.1). Authorization is not possible without some authentication and authorization information, but in DCAF, servers (and also clients) only need to store authentication and authorization information about their authorization manager and for the devices with which they are currently communicating. Security associations with new peers are created dynamically. DCAF specifies how access tickets should be managed. They are only stored if they are currently needed, and are removed when they expire (see section 12.2.7, 12.2.7.1 and 12.2.7.2, respectively) or are no longer needed (see section 12.2.10). Servers and clients are notified if a ticket replaces an older ticket, so that only a single ticket needs to be stored per communication.

DCAF messages contain most of the information that is required for the authenticated authorization and are therefore mostly self-descriptive. The underlying security solution must store some state that e.g., allows the endpoints to determine if a message belongs to the current communication context and helps against replay attacks. DCAF servers does not require mechanisms as the OAuth token introspection to obtain additional information for an access ticket.

As described in section 9.3.3, the secure transmission of data over intermediaries entails some difficulties. DCAF works with both transport security and object security solutions. It clearly defines which parts of the transmitted authorization data must be protected by the underlying security solution. Also, DCAF describes how access tickets may be cached (see section 12.2.3). DCAF therefore considers the main REST concepts and implements them where possible; DCAF makes a reasonable trade-off concerning REST concepts and security (**REST Req**).

### 13.3.3. Environment Requirements

In addition to the special characteristics of constrained endpoints, the environments where they are used often have some typical characteristics.

Overseeing principal are often not able to intervene in the communication DCAF, e.g., because they are not present or the constrained devices do not have a user interface. DCAF enables autonomous constrained devices to participate in the protection of their overseeing principals' security objectives with the assistance of their authorization manager. In contrast to the ACE framework, DCAF fully supports constrained clients. DCAF therefore meets the **Absent Principal Req** for both clients and servers.

Complying with the **Offline Req** is a difficult problem. A device that has neither a connection to its overseeing principals nor to already authorized claim issuers cannot comply with the **rule completeness fundamental**. In DCAF, constrained devices need their authorization managers to obtain new authorization information about their communication partners. For this reason, a connection with their authorization manager is required. Also, the devices' authentication and authorization information for their authorization managers must be kept up to date which requires a (not necessarily direct) connection to their overseeing principal at some point. However, after the current authorization rules were obtained, clients and servers can communicate for a certain amount of time without the need to contact their authorization managers. Also, local authorization managers may be used, so that no connection to the Internet is required. In the basic protocol flow, the client obtains the access ticket, which means that the server does

not require a connection to the Internet, but only to the client. If only the server has access to its SAM, SITR may be used to obtain the access ticket, if this is really necessary.

For the **Multiple Hops Req**, the authorization solution must consider that the communication may need to be transported over multiple hops. As described above, DCAF may be used with transport security or object security solutions, and specifies which data must be protected during the transmission.

DCAF can be used in application scenarios where the client, the server or both are constrained. Protocol flows where either the client or the server is less-constrained are shown in section 12.3, which fulfills the **C2L Req**.

DCAF is designed for the communication between two constrained devices. In contrast to the ACE framework, it fully supports constrained clients: CAM belongs to the same security domain as C, acts as a mediator between C and COP, and provides C with the required authentication and authorization information that enables C to participate in the protection of its overseeing principal's security objectives. Similarly, SAM assists S to act in SOP's interest. DCAF therefore meets the **C2C Req**.

DCAF can be used to distribute the required keying material and authorization information for group communication (**Group Communication Req**) as shown in section 12.5.

How local data on the devices is stored and protected is not part of DCAF. The framework therefore cannot provide protection against physical attacks (**Capture Req**). But, DCAF enables fine-grained authorization on clients and servers and therefore helps to reduce the potential damage of such an attack; their communication partners can only send and receive data as permitted. E.g., a constrained client will only send requests that are covered by COP's authorization rules instead of having to trust in S to reject requests that are not allowed. Also, the DCAF revocation mechanism allows to quickly react when endpoints were compromised.

As constrained devices have only very limited hardware resources, they are particularly prone to DoS attacks. In DCAF, the preferred way of providing the access ticket to the server is transmitting it in the DTLS handshake. DTLS provides

some protection against DoS attacks. Also, the server can immediately discard the ticket if the handshake fails. Complying with the REST stateless server constraint (see section 13.3.2) saves storage space on the server. In DCAF, clients and servers only need to permanently store the security association with their own AM. All additional information can be obtained dynamically. DCAF messages are self-descriptive; client and server do not need to store or obtain additional information. Thereby, DCAF helps to mitigate the risk of DoS attacks (**DoS Req**).

### 13.3.4. Authorization Rule Requirements

The security objectives of the involved overseeing principals may differ as, e.g., the smart metering use case (section 4.1.4) shows. In DCAF, each constrained device has its own authorization manager that belongs to the security domain of its respective overseeing principal. Thereby, principals can directly decide about the authorization rules for their devices (**Separate Authority Req**). The authorization policies of both COP and SOP are considered by DCAF (**Distinct Interests Req**). DCAF enables fine-grained authorization on both the client and the server side and therefore meets the **Fine-Grained Authorization Req**.

How the authorization rules are configured is not part of DCAF. But in DCAF, the authorization managers act as a mediator between the overseeing principal and the constrained device. The constrained devices may not have user interfaces and displays and may have limited storage and networking capabilities which may lead to long reaction times. Their limitations make their administration tedious. For the expected large number of IoT devices, manually administrating each device is not a feasible solution. Authorization Managers are less-constrained devices and are therefore much more powerful than the constrained endpoints. They may convey authorization rules to a large number of devices. The deployment of an AM therefore makes it much easier for the overseeing principal to configure the authorization rules. Thus, DCAF meets the **Simple Configuration Req** for both clients and servers.

Access tickets and client information contain a lifetime that defines their validity. Since constrained devices may only have a limited ability to measure time,

we expect the AMs to manage time-based context constraints and provide access tickets and client information as currently required. The authorization rules provided by the overseeing principals may refer to holders with location-based attributes. CAM and SAM may obtain the attributes of their own constrained devices from their respective OVP, and may relay this information to the AMs with which they are communicating. Thereby, location-based authorization rules may be realized (**Context-Based Authorization Req**). The details of this process are not in scope for DCAF.

The delegation of permissions is part of the configuration of rules by the administrator which is out of scope for DCAF.

In DCAF, authorization rules are provided to clients and servers by their respective AMs. DCAF therefore enables autonomous servers and also autonomous clients to enforce the authorization decisions of their overseeing principals (**Autonomous Server Req** and **Autonomous Client Req**). DCAF allows for mutual authorization; no additional protocol is required. The **Mutual Authorization Req** is met.

### 13.3.5. Authorization Rule Update Requirements

DCAF allows the dynamic update of authorization rules and related keying material for both clients and servers (**Dynamic Updates Req**). The setup and management of authorization rules for their authorization managers and overseeing principals is out of scope for DCAF. We will analyze in chapter 14, how DCAF can be used for these relationships.

In DCAF, access tickets and client information have lifetimes that limit their validity period. Since clients and servers must check if the authorization rule are still valid, access is not possible outside this time period. The **Temporary Permissions Req** is met.

The DCAF revocation mechanism enables clients and servers to quickly adapt to unforeseen authorization changes, e.g., if the communication partner was compromised (**Revocation Req**).

## 13.4. Evaluation Conclusion

Security protocols that do not clearly specify how each necessary authorization and delegation tasks must be performed have multiple problems: the protocol designers may overlook flaws and vulnerabilities in the protocol, or may not have considered how certain required information is securely obtained by the actors. Where tasks are omitted from the protocol description, implementers must fill these gaps with their own interpretation. This may lead to interoperability problems and unsecure implementations. DCAF avoids these problems by specifying how each task is performed. For tasks that are not performed inside the protocol flow, DCAF describes the required security associations and the authentication and authorization data that must be obtained to ensure continuous security in the whole authorization process. DCAF also explicitly defines the data structures that ticket face and client information must contain, and specifies clearly how these pieces of information must be protected. The risk of vulnerabilities is thereby strongly reduced; In DCAF, the constrained devices are securely provided with all pieces of information that they require for an effective authentication and authorization.

With the authorization fundamental analysis of DCAF, we have shown that it is possible to write protocol specifications that describe every authorization and delegation task for each actor and each piece of data that is required for the protocol flow. ANAZEM's list of tasks enables protocol designers to explicitly specify which tasks are performed by the protocol, and which tasks must additionally be addressed, e.g., by an underlying security solution. The necessity to consider how every task may be addressed reduces the risk of gaps and omissions. Also, it offers developers a more complete view on the overall requirements of the solution and assists them with finding suitable building blocks for their application scenario. Analyzing a protocol specification with the model thus increases the security and quality of the protocol, and makes it easier to implement.

Our quantitative evaluation shows that the messages that DCAF servers must handle have a similar size as those of ACE framework servers with comparable security. Code and data sizes of our DCAF server implementation are smaller than those of the ACE DTLS profile server. But, as we have seen, autonomous

or constrained clients benefit the most from using DCAF: we have shown that message and code sizes can be significantly reduced if the client has an own authorization manager.

In contrast to solutions such as OAuth 2.0 and the ACE framework that only implement a three-corner architecture, DCAF fully supports autonomous clients: clients with their own authorization manager do not need to establish other security associations by themselves. DCAF combines the distribution of authentication and authorization information to clients and servers in a single protocol flow, which reduces code sizes, network traffic and processing time and thereby minimizes energy consumption.

For the overseeing principals, the authorization managers offer a more comfortable way to configure and manage authorization rules: these devices are more powerful, will in most cases be easier to reach, have shorter response times, and can be equipped with more user-friendly tools since they have more storage space and memory.

Since DCAF does not require clients to register with authorization managers from other security domains, it offers more flexibility and enables clients and servers to better adapt to unforeseen changes. If in DCAF a constrained device must be replaced, the AM from the other security domain does not have to be provisioned with the new device's credentials: it can still authenticate the new devices' AM and validate its authorization. Each AM vouches for the authentication and authorization data of the constrained devices for which it is responsible. This allows for immediate changes where necessary. The loose coupling allows endpoints from different security domains to evolve independently. This facilitates rapid progress, which is an important success factor for the Internet.

## 14. Authorization Transitions

In the previous chapters we introduced and analyzed authorization solutions that focus on the operational phase of the life cycle of constrained devices. A short overview of the life cycle phases is given in section 3.2. For an effective authorization solution, security must be continuous in the whole life cycle of the devices; each gap may be exploited by attackers.

Overseeing principals are the authority for their data and devices and define the authorization rules. In the task delegation architectural style, a constrained device has an authorization manager that assists with authorization tasks. The AM acts as a mediator between the device and its overseeing principal. In most cases, the constrained device will receive authorization information from its AM, and rarely or not at all communicate directly with its overseeing principal. For a constrained device, the security associations to its overseeing principal and the authorization manager are the most important security associations. An effective authorization is only possible if these relationships are securely established and managed. The permission to issue access tickets must be particularly well protected, because an entity with this permission is in complete control of the device and its data. During the life cycle of a device, the overseeing principal and the AM of a smart object may change, and the authority must be transferred from the previous to the new AM. We call this process *authorization transition*. For continuous security, authorization transitions must be carefully designed and protected.

As we discussed in section 3.1.1, the smart objects will often lack user interfaces and displays, and network communication may be the only means for overseeing principals to interact with them. Also, manually performing authorization transitions on each constrained node is not feasible for many application scenarios, especially considering the expected large number of devices. An automated

---

solution is often required. But only authorized entities must be able to initiate authorization transitions and access data that influences the authorization: an authorization solution is required.

To achieve uniform interfaces, CoAP as a REST implementation stores data representations in resources. It is an obvious choice to also use resources to store information concerning the authorization manager, and to obtain permission to change these resources using the usual authorization protocol.

In this chapter, we will show how an authorization solution such as DCAF can be used to manage the authorization permission and perform authorization transitions in the whole life cycle of constrained devices. We consider only the authorization solution, not other life cycle problems such as how a new constrained device gets access to a network. The introduced concepts are based on our earlier work on this topic (Gerdes et al. 2015d; draft-gerdes-ace-a2a).

As we have seen in chapter 4, smart objects are used in environments with varied characteristics. In some cases, the devices are installed in places where they are publicly accessible. In others, they are not easily accessible at all, e.g., because they are installed inside a wall. Smart objects may belong to a company and be maintained by professionals or, e.g., if they are part of a smart home, must be configured by overseeing principals with often only limited technical knowledge. The characteristics of the smart objects and of their environments influence how each phase of the life cycle looks like. It is therefore difficult to find a single solution that is useful for all scenarios. Instead of defining a one-size-fits-all approach, we will show how DCAF can be used for various common approaches for each phase of the life cycle.

Section 14.1 explains the tasks that are relevant for the authorization transitions. In section 14.2, we discuss work on the initial establishment of security associations. How an authorization transfer is conducted, which resources are required and how they must be managed is explained in section 14.3. Section 14.4 describes how authorization transitions are securely performed in the various phases of the life cycle by using DCAF. In some cases, e.g., if the AM broke or was compromised, the usual DCAF protocol flow cannot be used; a fallback mechanism is required which we depict in section 14.5. How DCAF may help with

software updates is discussed in section 14.6. Our findings are summarized in section 14.7.

## 14.1. Authorization Tasks

The mechanisms introduced in this chapter focus on the relationship between the constrained device and its AM. The secure establishment and the secure management of the security association with the AM enables the constrained device to validate that the AM is authorized by the device's overseeing principal, i.e., to perform the tasks **A4** and **D4**. Also, the processes may provide the constrained device with data that it requires to perform other tasks, e.g., the knowledge how the constrained device is identified to perform the tasks **A5** and **D5** for claims from the AM.

The constrained device must securely obtain the information required for the authorization transition from an authorized claim issuer. The device and the claim issuer therefore must perform the delegation tasks in conformance with the delegation fundamentals to achieve a secure solution. A difficult problem in this scenario is how the constrained device validates the authorization of the claim issuer (task **D4**).

How the overseeing principal communicates with the AM is out of scope. The AM is expected to be provided with the knowledge that it is responsible for a certain device which enables the AM to validate the smart object's authorization to delegate tasks to the AM (task **D0**). Also, the overseeing principals must inform the AM about the constrained device's attributes, i.e., they must perform task **An1** in behalf of the AM concerning the AM's constrained devices. The attributes must be kept up to date to satisfy the **claim completeness fundamental**. To satisfy the **claim issuer participation fundamental**, the AM must be provided with data that enables it to perform tasks for the constrained device, e.g., to securely obtain permissions in behalf of the device from the overseeing principal. To satisfy the fundamentals, the AM must perform the delegation tasks for obtaining authorization information concerning the device's peer from the overseeing principal.

## 14.2. Related Work

Many security solutions do not consider the whole life cycle of a constrained device. Below we discuss approaches that focus on the initial establishment of security associations.

The goal of the Bootstrapping Remote Secure Key Infrastructures (BRSKI, draft-ietf-anima-bootstrapping-keyinfra-41) is to offer a mechanism for an initial distribution (“bootstrapping”) of CA certificates without the need for user interaction (zero touch). BRSKI defines how an unconfigured device, the pledge, and a device that already is part of a network, the registrar, mutually authenticate each other and validate each other’s authorization (draft-ietf-anima-bootstrapping-keyinfra-41, p. 6). The registrar decides if the pledge is allowed to join the domain. The registrar requests a voucher from the manufacturer for the process. The voucher indicates to the pledge that the registrar is authorized. It contains an X.509 certificate that enables the pledge to authenticate the authorized registrar (RFC 8366, p. 13). The pledge is expected to have an X.509 certificate and a device ID (IDevID) that are installed during manufacturing (draft-ietf-anima-bootstrapping-keyinfra-41, p. 17). The registrar uses this certificate to authenticate the pledge. To validate the authorization of the pledge, the registrar uses the IDevID in the X.509 certificate. This ID is used to determine the type or vendor of the device, and authorization is granted based on this information (draft-ietf-anima-bootstrapping-keyinfra-41, p. 42).

BRSKI was not designed for constrained devices and may not be applicable in the IoT because parts of the protocol, in particular the large credentials and key sizes, may deplete the energy of battery-powered devices and may lead to congestion in low-power networks (cf. draft-ietf-anima-bootstrapping-keyinfra-41, pp. 11–12). An approach for distributing symmetric keys is currently missing. Constrained devices may have difficulties to interpret timestamps in vouchers and certificates, and therefore may not be able to determine if they are still valid. To mitigate this problem, the pledge’s voucher-request must contain a nonce, and if the voucher contains a nonce, the pledge must check if it matches the nonce in the voucher request. Requesting the voucher is only possible if the manufacturer can be reached. In cases where the manufacturer has gone out of business, BRSKI

can no longer be used. In these cases, so-called “nonceless” vouchers may be used for bootstrapping. But, issuing such vouchers is a security risk for new owners after a handover. If the former owner obtained a nonceless voucher, it may use it to regain control over the pledge.

The voucher provided by the MASA is supposed to enable the pledge to validate the registrar’s authorization. The MASA therefore must validate the registrar’s authorization in behalf of the pledge. How this is achieved depends on the MASA’s policies, which may differ from the policies of the pledge’s overseeing principals. The MASA may, e.g., accept voucher requests from any domain owner (draft-ietf-anima-bootstrapping-keyinfra-41, p. 74). The pledge may then communicate with a registrar that does not actually belong to the pledge’s owner. The registrar may notice with the assistance of MASA audit logs if the pledge registered with the wrong registrar and then may notify an overseeing principal (draft-ietf-anima-bootstrapping-keyinfra-41, p. 60). The OVP must be enabled to reset the pledge, most likely by physically accessing the device.

While some details are left to the developers, BRSKI is useful for the secure distribution of the initial keying material that is required to integrate a new endpoint into an existing network. Additionally, the new endpoint must be provided with information about authorized claimants (**claim issuer authorization fundamental**), and the claimants must be informed that the endpoint is an authorized delegator (**claim issuer participation fundamental**).

The resurrecting duckling security model (Stajano and Anderson 1999) was developed for the initial exchange of keying material between two devices. In this model, a constrained device, the duckling, accepts keying material only once, from the first device it encounters (the mother). Stajano and Anderson call this mechanism *imprinting*. This model may be used for the initial establishment of a security association between a constrained device and its AM. We have shown how such an imprinting process may be performed using CoAP (see Bergmann et al. 2012a).

## 14.3. Design of Authorization Transitions

DCAF implements the task delegation architectural style: each constrained device has an AM that assists it with authorization tasks. As described in section 12.1.2, constrained clients and servers must have a security association with their respective authorization manager. The keying material for communicating with the AM and additional information such as how the AM may be reached must be securely obtained. If the AM of a constrained node changes, all information concerning the AM must securely be changed.

Section 14.3.1 describes the resources that contain information concerning the authorization manager authorization. How the permission to issue access tickets to an endpoint are transferred with the assistance of DCAF authorization tickets is described in section 14.3.2.

### 14.3.1. Authorization Manager Data Resources

The information that identifies the authorization manager must be stored on the constrained device and used for the communication with the AM, e.g., to validate that a ticket face or client information stems from the AM. Authorization transitions can be performed using REST implementations such as CoAP if the required pieces of information are stored in resources. We will call resources that contain information about the AM such as its keying material or address *authorization manager data resources* or short *AM-data resources*. These resources must be particularly well protected. DCAF protects the access to resources, and can be applied here as well.

In REST, only servers have resources. To allow for authorization transitions, constrained devices that usually have the client role act as CoAP servers for the authorization transition. Clients and servers must be available at regular intervals to allow for authorization transitions or other configuration updates.

To perform authorization transitions with DCAF, constrained devices must at least store the keying material for communicating with their AM in an accessible

resource. We will call this resource the *AM-key resource*. The constrained node must use the keying material in this resource to communicate with the AM, e.g., to validate the origin of received authorization information (tasks **A3** and **A4**, see chapter 5). The keying material in the AM-key resource identifies the authorized AM. The AM-key resource therefore represents the AM's authorization.

For the communication with the AM, both symmetric and asymmetric cryptography may be used (although a symmetric solution is usually preferable for constrained devices). For symmetric solutions, the keying material that is shared between the constrained device and its AM is stored in the AM-key resource. If an asymmetric solution is used, the AM's public key is stored in the AM-key resource. Additionally, the constrained device requires its own key pair. The overseeing principal may decide to make this also changeable by providing resources for it. The shared symmetric key and the private key are confidential and the resources must be protected accordingly. All communication keys for the communication with the AM must only be changed by entities with a special authorization (see also section 14.3.2). Since there usually is no reason for the AM to read the keying material, the AM-key resource should not be readable at all, especially if it is meant for symmetric cryptography.

The data format for storing the keying material should include required additional information such as the cryptographic algorithm for which it is to be used. For DCAF, the COSE\_KEY structure (see also section 10.1.5) should be used.

In some cases, the constrained node must know how it is identified to perform the tasks **A5** and **D5**; e.g., if SAM and S use asymmetric cryptography to communicate and the access ticket is not encrypted, SAM must explicitly specify S as the destination of the ticket (see also section 12.1.2). We call the required identifying attributes *self-attributes*. Depending on the application, several constrained devices may have the same attributes if they have the same characteristics. E.g., authorization information may be destined for all light bulbs in the living room. Using shared attributes instead of unique identifiers makes the authorization mechanism more flexible which is especially useful for group communication (see also section 12.5). Self-attributes must be changed together with the AM keying material to avoid authorization problems (see also section 14.3.2). They are therefore

stored in a data structure together with the AM keying material in the AM-key resource.

Implementers may decide to also provide a resource for the AM address. The constrained devices need to know how they can contact their own AM. S must inform C how SAM is contacted for requesting an authorization ticket (see section 12.2.1). We call the resource where the AM's address is stored *AM-path*. If the address is not provisioned by the overseeing principal, it must securely be obtained from an authorized entity, e.g., a resource directory.

AM-data resources that can be used to manipulate the authorization such as the keying material or the identifier of the constrained device<sup>1</sup> must only be accessed by entities with a special authorization, e.g., the AM or the overseeing principal, to satisfy the **claim issuer authorization fundamental** (see section 5.4.1). Otherwise, the whole authorization process is in jeopardy.

### 14.3.2. Planned Authorization Permission Transfer

A special challenge in the life cycle are the phases where the overseeing principal or the authorization manager of a device changes. The authorization to issue permissions must be transferred to the new overseeing principal and the new authorization manager. In most cases, old permissions must become invalid, and security associations that are based on these permissions must be terminated. With a transition phase, a more gentle transition is possible (see section 14.3.2.3).

The new overseeing principal must provision the new AM with the necessary information for its new smart object; in particular, the AM must learn the new device's attributes, e.g., its name or location.

If the AM for a device changes, the respective AM-data resources must be changed accordingly. As for every other resource, an entity must first obtain the permission to gain access to an AM-data resource. In DCAF, authorization tickets are

---

<sup>1</sup>Applications may require additional information for the authorization that is not explicitly mentioned here but nevertheless must be protected.

usually generated by the respective AM. An obvious approach is therefore to obtain an access ticket for the AM-data resources from the current AM using the usual protocol flow. In this case, the new AM should act in the role of the requesting client.

If the previous AM is no longer available or not reachable for a ticket request, a different solution is required. A pre-configured ticket may be obtained out of band. In some cases, the new overseeing principal of a device may gain the keying material that identifies it to be authorized and can use it to change the AM-data resource.

Summarizing, the following options are available to obtain the access ticket for a planned authorization permission transfer:

- the ticket is requested from the former AM using the usual protocol flow,
- a pre-configured ticket is delivered with the smart object,
- the smart object is delivered with the AM's keying material that is then provided to the new AM. Thus, the new AM is able to generate a ticket for the device.

We distinguish former keying material that is used between the former AM and the constrained device, and new keying material that the new AM provides to the constrained device. The former keying material is stored in the AM-key resource, and changed by the the authorization permission transfer to the new keying material<sup>2</sup>. The access ticket for the authorization permission transfer may additionally contain session keys that are used by the constrained device to authenticate the new AM (task **A6**). We call AM keying material that is delivered with the constrained device delivered AM keying material (see also section 14.3.2.4).

The constrained device may require self-attributes (see 14.3.1) to validate that it is the intended destination of an access ticket. Self-attributes therefore influence the authorization. If the self-attributes change, access tickets with a different destination must no longer be valid. It is therefore not possible to use the access

---

<sup>2</sup>The former overseeing principal may decide to change the former keying material to a temporary secret for the authorization transition. In this case, we consider the temporary secret to be the former keying material.

ticket of the former AM after either the AM keying material or the self-attributes were changed. Also, if the new AM does not know how the constrained device is identified, it may not be able to issue access tickets that the constrained device accepts. Therefore, the AM keying material and the self-attributes should always be changed together.

#### **14.3.2.1. Obtaining Access Tickets From Former AM**

If an access ticket for changing an AM-data resource is obtained from the former AM, the new AM should act as the requesting client to reduce the risk of disclosing confidential keying material. The client uses the usual DCAF protocol flow to obtain the ticket and establish secure communication with the constrained device. Data that is sent in the optional unauthorized resource request cannot be protected. The new keying material must not be included in this request. The authorization information in the ticket must at least comprise the permission to change the AM-key resource. The former AM must check if the requesting client is authorized to access the resources as requested.

The tickets should contain ticket face and client information. The AM keeps the client information and transmits the ticket face to the constrained device as described in section 12.2.7. The ticket must again at least comprise the permission to change the AM-key resource. The new AM and the constrained device then use their respective parts of the ticket to establish a secure communication. The new AM then changes the AM-key resource to the new AM key. After the AM-key resource was changed, the constrained device must consider tickets including the pre-configured ticket to be invalid since they were endorsed with the former AM's keying material. The constrained device must discard these tickets and end all communications that were authorized by the former AM, maybe after a short transition phase (see section 14.3.2.3).

Access tickets for changing the AM-key resource may be valid for a very long time or even infinitely. Their main purpose is the authorization transition, and after the transition was performed and the AM keying material is changed, the ticket becomes invalid.

### 14.3.2.2. Pre-Configured Tickets

Pre-configured tickets must be provisioned to the new AM who acts as the client. The confidentiality of these tickets must always be protected. The former AM will often not know the new AM's public key and therefore cannot use it to define the new AM as the holder of the preconfigured ticket. The keying material in pre-configured tickets should be symmetric.

Delivering an access ticket with the constrained device instead of AM keying material has the advantage that the AM's confidential keying material is not transferred but only a one-time session key; it can only be used once to change the AM-key resource. Since the new overseeing principal must change the AM-key resource to gain control of the constrained device, the former AM, or an attacker that may have managed to get hold of the pre-configured ticket during the transport, can no longer issue valid tickets for the device after the authorization transition.

If the mechanism fails, i.e., if using the pre-configured ticket does not work for some reason, a fallback mechanism must be used to gain control over the device (see section 14.5).

### 14.3.2.3. Transition Phase

If all tickets are discarded at once and every ticket must be newly requested from the AM, the strain on constrained devices and the network load increases. For a more gentle transition, the old and the new AM may work in parallel for a while. During this time, the device has two AMs. The new AM is marked as responsible and is contacted for new tickets. Tickets from the former AM can reach the natural end of their validity period and gradually be replaced with tickets from the new AM. Thereby, a slow handover to the new AM is possible. Before the actual transition phase start, the old AM may already issue tickets with shorter lifetimes.

The duration of the transition phase is defined by the overseeing principals. At the end of the transition phase, AM-Key and related information of the former

AM are deleted. Tickets from the former AM are then discarded and former security associations are removed.

#### **14.3.2.4. Delivered AM Keying Material**

If the AM keying material was delivered with the device, it must be provisioned to the new AM. With the keying material, the new AM is able to issue access tickets for the constrained device.

Delivered AM keying material is always confidential and must be protected during the transfer. Delivered keying material should be symmetric because self-attributes are not required in that case. The new AM should change the device's AM-key resource that contains the keying material for the AM as soon as possible. To do so, the AM issues a ticket to itself for changing this resource. As long as the AM-key resource is not modified, the former AM, or an attacker that managed to get hold of the keying material during transport, may still control the constrained device.

#### **14.3.2.5. Changing the AM-Data Resources**

A new AM that changes the AM-key resource can choose the cryptographic algorithm for the communication between the AM and the constrained device. For the keying material that is stored on the constrained device, the AM may choose between:

- symmetric shared keying material,
- asymmetric keying material generated by the constrained device,
- asymmetric keying material generated by the AM.

If asymmetric cryptography is to be used, the constrained device must have its own key-pair and know the public key of its AM. The device may create its own keying material by itself if it has the ability to generate sufficient randomness, which is often a problem for constrained devices. If the new AM generates the

keying material, its integrity and confidentiality must be protected during the transfer, and the constrained device must validate that the keying material stems from an authorized AM. The AM's public key is not confidential, but must also stem from an authorized AM and be integrity-protected during transport.

After the AM-key resource was successfully changed, the constrained node may need to be configured further. As described in section 14.3.1, overseeing principals may decide to store additional pieces of information for the authorization in a resource to allow them to be changed over the network. The constrained device's identifying attributes, the self-attributes, directly influences the authorization and delegation if they are required to perform the tasks **A5** and **D5**. In this case, the constrained device and its AM must have a common understanding how the device is identified. For an authorization transition, the respective self-attributes must be provided to the constrained device and its new AM.

An incorrectly set network address of the AM cannot compromise the authorization, but a wrong address may make the AM unreachable and thus impair the availability of the node's service.

All data that influences the authorization should be securely changed if an authorization transition is performed (**claim issuer authorization fundamental**). If the previous overseeing principal set an AM-data resource, the new overseeing principal may need to either change or delete it. Also, new AM-data resources may be required which must be created and initialized.

The new overseeing principals must define the authorization rules for the new node, including the AM-data resources. As described in section 14.3.1, AM-data resources must only be changed by entities with special authorization, e.g., by the AM. Resources with confidential keying material should not be readable at all. Authorization rules for the new constrained devices are configured on the device's new AM.

In some cases, a planned authorization permission transfer is not possible, e.g., if the AM broke or was stolen. In this case, a fallback mechanism is required (see section 14.5).

## 14.4. Authorization Transitions in the Life Cycle

We will now describe how authorization transitions may be performed in the whole life cycle of a constrained device, and show how DCAF is used to secure the process. The sections 14.4.1 to 14.4.6 discuss how the respective phase of the life cycle must be secured.

### 14.4.1. Manufacturing

Smart Objects are built in the manufacturing phase. The manufacturer may choose to already provision the devices with the necessary information to simplify the commissioning; embedded devices such as routers are today usually provided with keying material during the manufacturing phase. The overseeing principals may later use this keying material to configure the device. For BRSKI, the device is expected to be provided with the necessary information to securely distribute keying material to the device (see also section 14.2).

A constrained device with a DCAF implementation may be equipped with an AM-data resource containing the AM's keying material and other authorization-related information during this phase (see also section 14.3.1). The corresponding key may be delivered with the device, e.g., as a Quick Response (QR) code. As an alternative, a pre-provisioned access ticket, consisting of client information and ticket face, for changing the AM-data resource may be delivered with the constrained device.

Especially if a large number of devices is sold together, the vendor may decide to deliver them with an AM device. This is a convenient solution for the new overseeing principals: instead of commissioning each device individually, they only need to configure the AM, which can then relay necessary information such as configuration data and authorization information to its constrained devices.

The vendor may also decide to provide an AM service for the smart objects. The constrained device is then coupled with the vendor's AM. The new overseeing principal does not require her own AM but can use the vendor's AM instead.

Summarizing, the following options are available for the provisioning of AM keying material on the constrained device during the manufacturing phase. This list covers the main options, but is not exhaustive; other options may be possible:

- provisioning of AM keying material only,
- provisioning of an access ticket for changing the AM-data resources,
- provisioning with AM device,
- provisioning with AM service,
- provisioning of keying material (a “master key”) that can then be used to change the AM-data resources,
- no provisioning of AM keying material.

#### **14.4.2. Commissioning**

After the manufacturing phase the smart object enters the commissioning phase, and its overseeing principal changes. If the device has not already received the AM’s keying material during the manufacturing phase, it must be provisioned now. The smart object is brought into the communication network of the new overseeing principal. The new IoT device must validate that it is allowed to communicate with other nodes in this network; also, nodes that are already part of the network must determine that they are allowed to communicate with the new device. In the commissioning phase, devices may already need to interact with endpoints from different manufacturers (see also section 4.1.3).

For some use cases, e.g., in the building automation case, the initial commissioning of components is often performed by a special commissioning staff from a different company. After the commissioning, they no longer require access to the device. We will discuss this scenario in the maintenance section.

During the commissioning phase, the constrained device must gain access to the network, e.g., using BRSKI. How this is accomplished is not part of this work.

Section 14.4.2.1 describes how the provisioning of keying material may be performed if it was not already accomplished in the manufacturing phase. In section 14.4.2.2, the required steps for coupling the constrained device with a new AM are explained.

#### 14.4.2.1. Provisioning of Keying Material During Commissioning

In the commissioning phase, the constrained device must be provisioned with all information that it requires for its work. If no keying material for an AM was provided during the manufacturing phase (no provisioning), the initial provisioning of keying material must now be performed. Since no previous security association exists, the provisioning must either be performed with an out-of-band mechanism, or a leap-of-faith protocol is required. For the out-of-band mechanism the overseeing principal makes use of a communication channel that differs from the primary means of communication and is secure. The overseeing principal may, e.g., connect to the smart object by plugging a USB cable into the device. However, the means for interaction with the smart object will in most cases be limited; a “smart” light bulb will likely neither have a display nor a USB port. Network communication may be the only way to communicate with the device (see also section 3.1.1). Out-of-band provisioning may therefore not be possible. Out-of-band mechanisms also often have the disadvantage that they require individual provisioning. If the security of the mechanism relies on the fact that only overseeing principals have physical access, this approach cannot be applied to devices that are located at publicly accessible places (see also section 14.5).

In a leap-of-faith protocol, the communication partners exchange keying material during their first message exchange. The initial contact is unsecured, but the subsequent communication is protected with the exchanged keying material. In most cases, additional measures are used to improve the security of the initial contact. A well-known example for a leap-of-faith protocol is the secure shell (SSH) protocol that can use leap of faith on the client side. To improve the security of the initial contact, SSH clients must check that the server’s host key identifies it as a server they want to communicate with (RFC 4251, p. 5). The user may, e.g., compare the fingerprint of the server’s public key with a list of keys that is given by the server’s administrators.

The resurrecting duckling security model (see also section 14.2) that was introduced by Stajano and Anderson may be used as a leap-of-faith mechanism. We adopted this approach for an imprinting process using CoAP. The process consists of three phases: the constrained device must first find its “mother” in the

discovery phase, then receive initial keying material in the imprinting phase and finally be provisioned with additional required data in the configuration phase. Stajano and Anderson suggests that the imprinting should be performed with an out of band mechanism: by touching the device with an electrical contact over which the keying material is transferred. Since smart objects may not have the required interface where the contact may be applied, this approach may not always be applicable. To have the advantages of a leap-of-faith protocol of using the same communication channel but mitigate the risk of other nodes getting hold of the device, the communication may be confined to nearby nodes during the imprinting phase, e.g., only link-local IP addresses may be used or the devices' radio power may be reduced to limit the coverage.

The out-of-band mechanism and the leap-of-faith protocol should provision all data that is required by the constrained device to validate an access token and successfully perform authorization. If the amounts of data that can be transported with these mechanisms are limited, the AM may provision only (maybe temporary) symmetric AM keying material. The AM can then use this keying material to issue an access ticket to itself and change other AM-data resources.

#### **14.4.2.2. Setup of a New AM**

As described in section 14.4.1, the device may use BRSKI to obtain keying material (a "master key"), that can then be used for the configuration of the device. This approach has the advantage that it usually does not require human interaction. This keying material may then be used to change the AM-data resource.

The manufacturer may also have already provisioned the device with the AM's keying material during the manufacturing phase. If the corresponding AM keying material was delivered with the constrained device, it must be securely provisioned to the new AM as described in section 14.3.2. Other AM-data resources that were created by the manufacturer may need to be changed when the AM-key resource is modified. Also, new AM-data resources may be required by the overseeing principal which need to be created.

Instead of the AM's keying material, the vendor may deliver the constrained device with an access ticket for the AM-key resource. The new overseeing principal

must provision the ticket to the new AM as described in section 14.3.2. Afterwards, the new AM can use the ticket to securely change the AM-data resources.

The manufacturer may decide to deliver the constrained device together with its AM. This is especially useful if a large number of devices is sold together. The constrained device is already bound to its AM, i.e., the AM's keying material is already stored in the AM-key resource. The overseeing principal configures the AM which can then provide required configuration data to the constrained node. This approach simplifies the commissioning for the overseeing principal because the configuration is conducted on the AM which is less constrained and therefore provides more options for interaction. The overseeing principal only has to interact with the AM and not with every single smart object. The AM should be instructed to change its own and the constrained devices' keying material as well as authorization-related information to make sure that the vendor no longer can control any of the devices. To change the respective AM-data resources on the constrained device, the AM issues an access ticket to itself.

#### **14.4.2.3. Using an AM Service**

The manufacturer or vendor of the smart object may provide its own AM with which the constrained device has a security association (see also section 14.4.1). The new overseeing principals may use the AM service temporarily to obtain keying material for the AM-data resources, or permanently if they do not wish to operate their own AM.

In every case, the overseeing principals first need to prove that they actually are the new overseeing principals of the device. They may, e.g., register on the vendor's website and provide a secret that was delivered with the smart object or was generated when the device was bought. Afterwards, the overseeing principals may configure authorization rules for their new smart object on the AM. While this may be convenient for the overseeing principal, the vendor or manufacturer can still control the smart object and access the data on it.

To use their own AM, overseeing principals define authorization rules that allow their AM to access the AM-data resources on the constrained device. Afterwards,

the new AM can request an access ticket from the vendor's AM and present it to the constrained device. After the AM keying material was changed, the former AM is no longer authorized to issue access ticket for the smart object.

#### **14.4.2.4. Authorization Rule Configuration**

After the authorization transition, the new AM must enforce the new overseeing principal's authorization policies. Authorization information that is provided to the constrained device in the client information or the ticket face must reflect the overseeing principals' decisions.

The AMs as less-constrained devices are usually more powerful and have more interfaces for interaction; they are therefore easier to configure than the constrained devices they are responsible for. How the overseeing principals configure authorization rules on the AMs is not part of this thesis. Since overseeing principals may not be present at the time of the communication between the constrained devices, authorization rules may be pre-configured on the AMs, e.g., over a web interface. The overseeing principal must define authorization rules for the new constrained device and also for the smart objects that are already part of the network and with which the new device is supposed to communicate.

#### **14.4.3. Operation**

In the operation phase, the smart object fulfills its purpose in life and can use authorization mechanisms such as DCAF as described in chapter 12. We already discussed the operation phase in detail in the previous chapters of this thesis.

#### **14.4.4. Maintenance**

When problems occur with a device during operation, the maintenance phase is entered. The smart object is repaired or reconfigured, or software updates are applied. Overseeing principals may want to outsource maintenance tasks. The

contractors then must be provided with temporary permissions for the time of their work.

For many maintenance tasks, the devices of the maintenance personnel may obtain access tickets from the constrained device's AM as usual. These tickets may have a shorter lifetime, depending on the required task.

If characteristics of the smart object or its environment change, the device's attributes may need to be updated accordingly. E.g., if the device has the attribute "living-room", this attribute must be changed when the device is moved to the kitchen. The device's AM must be notified of the changed attributes to ascertain that it specifies correct attributes of this device to other AMs (see also section 12.1.3).

More difficult are situations where the coupling between the constrained node and its AM must be changed. The AM of a constrained device may need to be exchanged, e.g., because the overseeing principal bought a newer model. If the change is planned, the overseeing principal may perform a planned authorization permission transfer as described in section 14.3.2. If the AM broke or was stolen, a fallback mechanism may be used on the constrained node as described in section 14.5.

Keying material should be exchanged regularly to satisfy the **correlation fundamental** and the **holder relation fundamental** (see section 5.8). To change the AM keying material, the AM issues an access ticket to itself and changes the AM-key resource on the constrained device as described in section 14.3.2. Old tickets must become invalid if the keying material is no longer secure. If possible, the AM should make preparations for the transition by issuing more short-lived tickets. The AM may use the old and new keying material in parallel for a while (as long as the old keying material is still secure) to achieve a more gentle transition (see section 14.3.2.3).

If the constrained device uses asymmetric cryptography for communication, the device's key pair may also need to be changed at some point. Since the private keying material is confidential, the constrained device should generate the new keying material by itself, if it is able to generate sufficient randomness (see also

section 14.3.2). If this is not possible, the AM and the constrained device should use their current keying material to protect and validate the confidentiality and integrity of the new key pair during transport. If the current keying material was compromised, a new commissioning or a fallback-mechanism may be used.

The software that is installed on a smart object may need to be updated, e.g., because vulnerabilities were found or to replace outdated cryptographic algorithms (see also section 5.8). We will discuss software updates in section 14.6.

#### **14.4.5. Decommissioning**

Smart objects that are no longer needed or replaced by newer components must be securely decommissioned in the decommissioning phase. The device is not necessarily destroyed. To make sure that the smart object cannot be misused after it was discarded, permissions for the node must be revoked.

Before a smart object is given away, all confidential data stored on it must be removed. For authorization, this includes keying material and authorization rules as well as confidential application data that was protected by the authorization solution. Ascertaining that the data actually is deleted and no longer accessible (secure erase) is not easy to accomplish. Manufacturers often publish documents of volatility (see, e.g., HP 2020; Xerox 2020; Juniper 2020) which usually list their products' memory components and their respective ability to store data when switched off (non-volatile). The documents may also state if the respective component contains user data and how the data may be removed. NIST SP 800-88 provides a guideline for making decisions on how to sanitize media containing confidential data (NIST SP 800-88). How data is securely removed from a smart object is not part of this work.

The overseeing principal must adapt the authorization rules on the AM: the decommissioned device no longer belongs to the devices that the AM is responsible for. The security association that the AM has with this node must be removed, including keying material and authorization rules.

The AM should revoke authorization information that was issued for the decommissioned smart object; other constrained devices that were communicating with

this smart object and their respective AMs should be notified that access tickets (client information and ticket face) for the decommissioned device are no longer valid. An example for a DCAF revocation mechanism that uses the CoAP Observe option was given in section 12.4. For this approach, the AM acts as an observe server and hosts a revocation resource. Devices that register with this resource regularly receive updates about the revocation state, and are thereby notified if permissions and claims for a constrained device are no longer valid.

Overseeing principals should consider how the decommissioning of a constrained device might affect other devices. The service that the decommissioned device provided might need to be overtaken by another device. Otherwise, other devices in the network may no longer be able to work correctly. How this is accomplished is not part of this work.

Throwing devices away is one approach to satisfy the **rule completeness fundamental** and the **claim completeness fundamental**. It may be useful if it is not possible to provide the device with new authorization rules or claims, e.g., if the keying material for controlling the device is irretrievably lost or if no software updates can be performed on the device (see also section 14.6). But if this is intended, it must be possible to throw the device away. Some devices, e.g., a light switch that is installed in a wall, may be difficult to remove.

#### 14.4.6. Handover

Security associations must be transient; if a smart object is sold, it must obey its new overseeing principal. If the AM breaks, it must be replaceable (see also Stajano and Anderson 1999, p. 4).

The handover phase can roughly be viewed as a decommissioning followed by a commissioning. A decommissioned smart object may be deployed in a different part of the network or given to a different owner. If the overseeing principal of the smart object changes, previous security associations and authorization-related configurations may need to be removed as described in section 14.4.5.

If the handover is planned, a planned authorization permission transfer may be performed as described in section 14.3.2. The new AM may obtain an access ticket

from the previous AM as described in section 14.3.2.1, before the constrained device is decommissioned. If this is not possible, the previous AM may issue an access ticket that can be used as a pre-configured ticket (see also section 14.3.2.2).

If no planned authorization transfer can be performed, a fallback mechanism is required to gain control of the device (see section 14.5).

## 14.5. Fallback Mechanism

If the AM for a smart object changes, an authorization transition must be conducted. If possible, a planned authorization permission transfer should be performed as described in section 14.3.2. In situations where the AM keying material is lost, e.g., because the AM broke or was stolen, a fallback mechanism is required on the smart objects that enables the legitimate overseeing principal to regain control over them.

A common approach for a fallback mechanism is to reset the device to factory settings. If we adopt this for our authorization transitions, the AM-data resources, in particular the AM-key resource are changed to their initial values. They can then be modified using the pre-configured access ticket or the keying material that was delivered with the device (see section 14.3.2). This approach has the advantage that only an entity that has the ticket or keying material can change the AM-data resources. It is also possible to reset the device to a setting where it can be provided with new keying material as described in section 14.4.2.1.

The smart object cannot decide if the fallback mechanism is currently required. The mechanism therefore must be available during normal operation. Attackers may abuse the fallback mechanism for a DoS attack or to even gain control over a smart object and its data. The fallback mechanism therefore needs to be well protected.

A common approach to trigger the fallback mechanism is an out-of-band channel. In the simplest case, the device provides a reset button that is pressed for a certain amount of time, e.g., ten seconds. As an alternative, new keying material

may be provisioned to the device over an additional channel, e.g., a USB cable (see also section 14.4.2.1). Out-of-band mechanisms require an additional channel which may not be available on smart objects. Also, out-of-band channels often require manual interaction, which may not be feasible, especially for large numbers of devices. In some cases, smart objects are installed in places where they are not easily accessible, e.g., inside a wall. For other applications, smart objects are placed in a location where they are publicly accessible. In this case, attackers may be able to abuse the out-of-band channel to trigger the fallback mechanism and thereby disturb the normal operation of the device.

A different approach to realize a fallback mechanism is to use additional keying material, a master key, that is only known to authorized entities, e.g., the overseeing principal or even the manufacturer of the smart object. The master key enables the overseeing principal to reset the AM-data resource to factory settings. The overseeing principal then proceeds as described in section 14.4.2.

It is also possible to use an AM service provided by the manufacturer as a fallback mechanism. The manufacturer may, e.g., have a special authorization to issue tickets for the AM-data resource. The new overseeing principal may then request an access ticket for changing the resource from the manufacturer's AM. This approach has the disadvantage that it is only usable as long as the manufacturer provides this service. Also, the manufacturer gains control over the smart object that may be abused. An attacker that manages to overtake the manufacturer's AM service may thereby gain control over a large number of devices.

## **14.6. Software Updates**

During the lifetime of a constrained device, its software may need to be changed; the software itself may have vulnerabilities or the cryptographic algorithms may no longer be secure (see also section 5.8). One possible solution is to throw devices with known vulnerabilities away. But for the IoT, where smart objects are expected to stay active for 10 or more years (see, e.g., RFC 8240, p. 3), this cannot be the only solution. Software updates may be required to satisfy the authorization and delegation fundamentals.

The software on a device influences the authorization process; e.g., the authorization protocol itself is implemented in a piece of software. Vulnerabilities may compromise the device and therefore need to be fixed. But attackers may attempt to overtake a smart object by providing it with forged software updates. A software update therefore is a claim that may influence the authorization process. The delegation fundamentals apply. Software updates must stem from an authorized source. Also, smart objects must obtain software updates as soon as they are available. An attacker must not be able to withhold software updates from them. The process of secure software updates is therefore similar to securely obtaining authentication and authorization information.

Constrained devices benefit from relying on their authorization managers for software updates. The required security associations already exist, and the smart object may already communicate regularly with its AM, e.g., to check the revocation state (see section 12.4). As for the revocation process we outlined in section 12.4, the constrained device may use observe to regularly check if it has the most recent software. If necessary, it can then obtain a newer version of the software. DCAF can be used to protect the update process; the software may be placed into a resource that may only be accessed by authorized entities. The details of the software update process are not in scope of this work. An analysis of the *Software Updates for the Internet of Things* (SUIT) information model (draft-ietf-suit-information-model-07) with the help of ANAZEM is an interesting topic for future work.

## 14.7. Conclusion

In this chapter we gave an overview of the challenges to authorization solutions in the various phases of the life cycle of a smart object, and showed how DCAF can be used to protect authorization transitions. We also explained which steps need to be performed outside the protocol to gain a secure solution.

The authorization to issue authorization information to smart objects must be well protected since otherwise attackers may gain control over the devices. Out-of-band mechanisms such as attaching a USB cable to the smart object may not

be applicable because smart objects often only have a single means of interaction: their network interface. Also, common out-of-band mechanisms often involve physical contact. Constrained Nodes are often used in places where they are exposed to others. The security of the nodes then is imperiled by physical out-of-band mechanisms.

In typical IoT scenarios such as smart homes or building automation, authorization transitions often need to be performed for a large number of devices at once. It is therefore especially important that the authorization transition solution is scalable and does not require manual interaction. We explained how data that influences the authorization such as the permission to authorize can be stored in resources and protected by DCAF. We then showed how DCAF can be used to securely transfer the permission to authorize to a new entity.

In DCAF, a smart object only requires a security association with its own Authorization Manager. Constrained devices can rely on their AM to assist with all authentication and authorization tasks. Having its own AM enables the smart object to perform tasks that would otherwise be very difficult. E.g., a constrained node will have difficulties to authenticate the source of a software update and validate its authorization on its own. If the device uses DCAF, it can rely on its AM for this purpose.

If necessary, the AM can be exchanged without the need for manual configuration of individual devices. The configuration tasks can be performed on the Authorization Managers which have the necessary user interfaces, processing power and storage space. DCAF offers usable security and continuous protection in the whole life cycle of constrained devices.

## 15. Integrating Constrained Devices into the Big Internet

Constrained devices often have difficulties to use Internet protocols that are common in the big Internet. As we have shown in the previous chapters, constrained devices and the environments in which they are used have special characteristics, and benefit from security solutions that were specifically designed to meet their requirements. Nevertheless, constrained devices must be able to communicate with devices in the big Internet to become real Internet components and avoid a silo solution for the Internet of Things. In this chapter we will outline how IoT devices using DCAF can be integrated with devices in the big Internet.

An important topic for IoT applications is the storage of application data in the cloud. We discuss problems of cloud-based solutions for the IoT in section 15.1.

The protocol stack used by constrained devices differs from the big Internet. Constrained devices use, e.g., CoAP instead of HTTP and DTLS instead of TLS. To allow for communication between the big Internet and the IoT, the less constrained nodes must either be able to use constrained level protocols, or translating intermediaries must be used. In section 15.2, we will discuss the use of intermediaries for authorization solutions, and analyze which requirements DCAF has for underlying security solutions.

IoT devices often need to be integrated into existing networks. Overseeing principals want to use their existing infrastructure to configure their devices instead of deploying a separate solution. We will show how DCAF can be integrated with OAuth 2.0 (section 15.3) and OpenID Connect (section 15.4).

## 15.1. Cloudy IoT

Constrained devices have only a limited ability to store application data such as sensor readings. Also, interaction with these devices is difficult because they have only little processing power and networking capabilities. The storage of application data on servers in the cloud appears to be an attractive solution; data can be easily stored, processed and presented to the user for comfortable viewing and easy manipulation from multiple devices. In comparison to a local server, the use of cloud services has the advantage that the maintenance can be outsourced.

The use of cloud services may be a problem if the functionality of the service that a smart object offers requires access to the cloud. In this case, the device is no longer usable and the service it provides is no longer available if the Internet connection is disturbed (see, e.g., Greenberg 2015). Another downside of cloud services is that users often have only limited control over the protection of data in the cloud. The accumulation of sensitive data may whet the appetite of attackers, which makes cloud services an attractive target for attacks. The scandal around the abuse of data of 5 billion facebook users to manipulate voters in the US election (see, e.g., Cadwalladr and Graham-Harrison 2018) shows the potential damage that even seemingly harmless information can inflict. User consent does not suffice to justify the access to vast amounts of user data. If data in the cloud is not sufficiently protected, users may lose their trust in cloud services and stop using them. For a secure IoT, overseeing principals of constrained devices must be able to decide what happens with their data in the cloud. User trust can be improved if cloud providers follow rules such as data minimization, minimal disclosure and permitting access only to parties with a legitimate claim (see also Bormann et al. 2014b).

Organizational measures for the protection of data may not always suffice. The trustworthiness of a cloud provider is not always easy to estimate. Business models often involve the processing of private user data. Also, jurisdiction concerning the protection of data in the cloud is not always clear (see, e.g., Matsakis 2018). To better protect sensitive data, the access to data on constrained devices should be restricted if possible. Where sensitive data needs to be stored in the cloud, it should be encrypted so that only the data owner can access it.

Designing the authorization manager service as a cloud service gives the overseeing principals the opportunity to comfortably configure access policies for all their smart objects at once. The overseeing principals are not required to carry a specific device with them to be able to configure their smart objects. But, the cloud provider may gain control over the constrained devices if no additional protection is applied. For an effective authorization, only the overseeing principals must be able to manipulate the AM's service and the configuration rules of their constrained devices. If an AM service in the cloud is used, its ability to issue permissions should be restricted (see also Gerdes et al. 2015c). A hierarchy of authorization managers may be used: less trusted AMs, e.g., in the cloud, may only grant a limited set of permissions that was previously defined by the overseeing principal. The risk of damage caused by the AMs can thereby be mitigated.

## 15.2. Intermediaries

One possible solution to better integrate smart objects into the big Internet is the use of translating intermediaries. Such intermediaries may also facilitate the integration of legacy protocols such as the home automation protocol FS20 (cf. Bergmann et al. 2012b). As discussed in section 9.3.3, translating intermediaries require access to at least parts of the transmitted messages. Security solutions on lower layers such as DTLS protect the whole message from the sender to the recipient, and do not allow entities on the way to read or modify message contents or headers of layers above. In this case, clients and servers may communicate with the intermediaries instead of directly with each other. Both the client and the server then need to establish a security association with the intermediary: they need to validate that the intermediary is authorized to send and receive certain messages. This approach has the advantage that clients and servers can use their respective protocol stack and no additional security solution needs to be installed.

Requiring security associations with the intermediary may not be desirable in every case; the intermediaries may not be fully trusted by the endpoints. Also, the intermediaries may become an attractive target for attacks. To solve this problem,

application layer security, i.e. an object security solution, may be used that protects only certain parts of the message while other parts can be accessed by the proxy. In this case, the object security solution must be installed on the client and the server, i.e., an additional common solution is still required. Object security solutions must not interfere with the authorization solution; if the authorization process and the subsequent communication require the protection of message content and certain header fields, the object security solution must provide this protection to be useful.

DCAF can be used with transport layer security or object security solutions as long as certain requirements are met (see also section 12.2). DCAF uses COSE data structures to transport authorization information and keying material. It can therefore be easily integrated with object security solutions that use COSE. DCAF can be used with any security solution that meets the requirements listed in section 12.1.4. *Object Security for Constrained RESTful Environments* (OSCORE, RFC 8613) is a candidate for an object solution for the IoT. An analysis of OSCORE with ANAZEM to determine if it meets the requirements and complies with the authorization fundamentals is an interesting topic for future work.

To satisfy the **rule completeness fundamental** and the **claim completeness fundamental**, endpoints must ascertain that they have the most recent permissions and claims. DCAF enables the constrained devices and the AMs to ascertain that claims from their peers are still valid. The validity period of attribute or authorization claims cannot be prolonged by withholding messages. The use of intermediaries does therefore not violate the **rule completeness fundamental** and the **claim completeness fundamental**.

### 15.3. OAuth 2.0 Integration

As described in section 2.5, OAuth 2.0 was designed to grant third-party applications limited access to web content. Overseeing principals can use OAuth 2.0 to provide their own DCAF authorization manager with authorization rules concerning AMs from a different security domain. Since the Authorization Managers

are less constrained, they can use OAuth 2.0 without modification. To obtain authorization rules, an AM acts as an OAuth 2.0 resource server. The AM from the other security domain acts as an OAuth 2.0 client. To illustrate the process, we imagine that COP wants to define authorization rules on CAM concerning SAM by using the OAuth 2.0 Code Grant flow. COP may initialize the process with the assistance of a user agent (UA), e.g., the browser on her smartphone. She triggers the client AM, e.g., by clicking on a link, which then requests access from the resource server's OAuth 2.0 authorization server (AS). The AS retrieves COP's authorization decision and issues the corresponding access token to SAM. Figure 15.1<sup>1</sup> depicts the process.

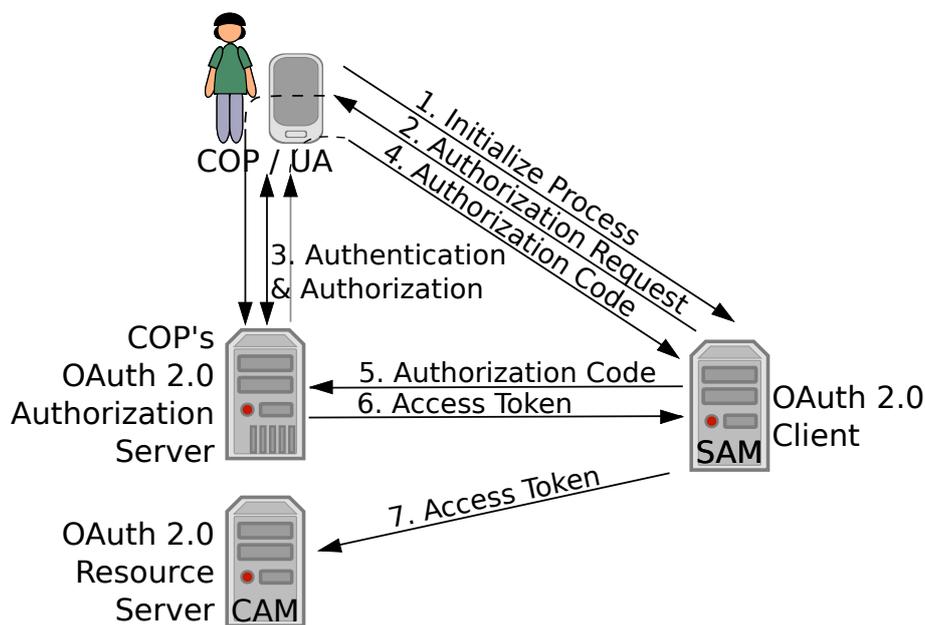


Figure 15.1.: COP defines authorization rules on CAM with OAuth 2.0

If SOP wants to authorize a CAM from a different security domain, a similar process can be used. In this case, CAM acts as the OAuth 2.0 client. It requests authorization from an OAuth 2.0 authorization server that is authorized by SOP to issue access tokens for SAM. If SOP grants authorization, the AS issues an access token to CAM that can be presented to SAM. When both CAM and SAM have obtained authorization rules from their overseeing principals, they are able to mutually validate each other's authorization.

<sup>1</sup>Based on the Authorization Code Flow figure depicted in RFC 6749, p. 24.

CAM and SAM may obtain the OAuth 2.0 access tokens prior to or within the DCAF protocol flow. C starts the DCAF protocol flow as usual by sending an access request to CAM. CAM then sends its OAuth 2.0 access token together with the ticket request to SAM. SAM validates the token and issues a DCAF access ticket that reflects SOP's authorization decisions as specified in the token. If the scopes in the OAuth 2.0 access token are not already encoded in AIF, SAM may need to translate them into this format to make them digestible for the DCAF server. If CAM specified C's asymmetric key in the ticket request, SAM binds it to the access ticket. Otherwise, SAM generates a symmetric verifier as a session key, and specifies it in the ticket face and the client information. The complete DCAF access ticket is then sent to CAM together with SAM's OAuth access token. CAM validates SAM's access token, and specifies authorization rules for C that reflect COP's authorization decisions in the token's scope. To do so, CAM may need to convert the authorization rules into AIF before adding them to the client information. The complete ticket is then sent to C. The remaining part of the protocol flow is conducted as usual: C keeps the client information and transmits the ticket face to S. With their respective parts of the ticket, C and S can communicate securely. Figure 15.2 shows how OAuth 2.0 and DCAF are performed on the respective architecture levels.

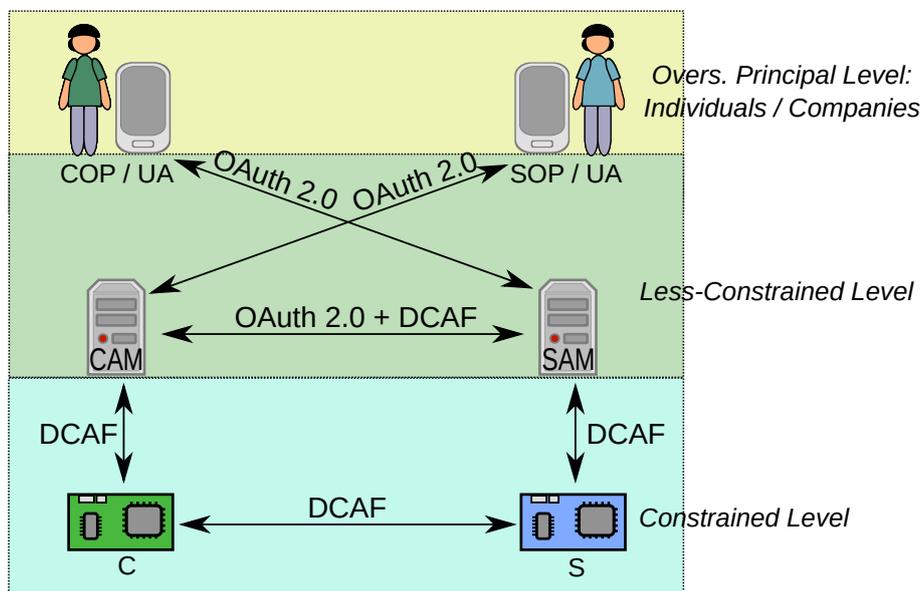


Figure 15.2.: OAuth 2.0 and DCAF

To provide the constrained devices with the necessary information to enable them to enforce their overseeing principals' security objectives, CAM and SAM need to interpret the OAuth access tokens and encode the required data in a way that the constrained devices can understand. As described above, the OAuth scopes from the access tokens may need to be translated into AIF if they are not already encoded in this format. The OAuth access tokens are bound to the AMs. The DCAF access tickets must be usable by C and S. SAM therefore must provide keying material for C and S in the access token and the client information. Constrained devices will often not be able to use OAuth 2.0 keying material since OAuth 2.0 usually uses certificates. SAM therefore provides pre-shared keys or raw public keys that are more suitable for constrained devices.

For an effective authorization, constrained devices must satisfy the **rule completeness fundamental** and the **claim completeness fundamental**; i.e. they must be able to determine if authorization information is still valid, detect which of several access tickets is the most recent, and recognize and discard replayed access tickets (see also section 12.1.7). Since DCAF tickets have sequence numbers, constrained devices are able to determine which ticket is the most recent. Constrained devices may not be able to understand timestamps in OAuth 2.0 access tokens such as the expiration time (see also section 3.1.4). DCAF enables constrained devices to determine the validity of tickets by introducing validity options that can be used by devices that only have a very limited ability to measure time (see also section 12.1.7). To translate from OAuth 2.0 tokens to DCAF tickets, the AMs must calculate the remaining validity period of the access token using the information in the token, e.g., the expiration time. The lifetime of the DCAF ticket must not exceed the remaining validity period. The AMs' OAuth 2.0 tokens may be issued for long-term use. The AM may issue DCAF tickets with much shorter lifetimes.

DCAF has stricter rules for its access tickets than OAuth 2.0. DCAF tickets are therefore easier to process for the constrained devices since less options have to be considered. More complex tasks are left to the less constrained authorization managers.

## 15.4. OpenID Connect Integration

OpenID Connect (Sakimura et al. 2014) is based on OAuth 2.0 and used for attribute validation: the OpenID provider vouches to an OpenID client that an overseeing principal at a certain user agent has certain attributes. OVPs can decide which data they want to disclose; they can define which attributes the OpenID provider may relay to a certain relying party. OpenID Connect can be combined with an OAuth 2.0 process; the client then additionally obtains an access token during the process.

An overseeing principal can use OpenID Connect to prove her identity to a DCAF AM that may stem from a different security domain. If combined with an OAuth 2.0 process as described in section 15.3, the overseeing principal may also inform her own AM about the foreign AM's authorization rules. SOP may, e.g., prove her identity to CAM with OpenID Connect and provide it with an access token for SAM with the nested OAuth 2.0 process. In this case, the OpenID Provider also acts as an OAuth 2.0 authorization server.

As an alternative, the OpenID Provider can also directly act as a DCAF Authorization Manager, and a DCAF authorization process can be nested into the OpenID Connect authentication. The following example (cf. draft-gerdes-ace-dcaf-examples-00) illustrates the process. Sarah owns a constrained device with a temperature sensor. During a heat wave she wants to monitor the temperature in her office and regularly publish it in her blog to inform others of her working conditions. She is registered with an OpenID provider (OP) that also acts as a DCAF SAM<sup>2</sup> for Sarah's blog server (S). The blog understands CoAP and DCAF, but allows only Sarah herself to post new entries. Sarah is the only overseeing principal in this case and acts as a combined COP and SOP. Sarah's constrained temperature sensor acts as a DCAF client (C). It is associated with a less-constrained CAM which acts as the relying party (RP). Sarah uses the browser of her smartphone (UA) to interact with CAM's web interface, and initialize the authentication and authorization process on CAM. With the assistance of Sarah and her UA, CAM

---

<sup>2</sup>A normal OAuth 2.0 server cannot be used in this case because it is not able to issue access tokens that constrained devices can understand. E.g., OAuth 2.0 servers use the certificate mode which is not feasible for constrained clients.

is able to authenticate with the OP and obtain a DCAF access ticket for Sarah's blog. The figures 15.3 and 15.4 depict the process. Operations concerning the authentication are represented by blue boxes. Operations for authorization have green boxes.

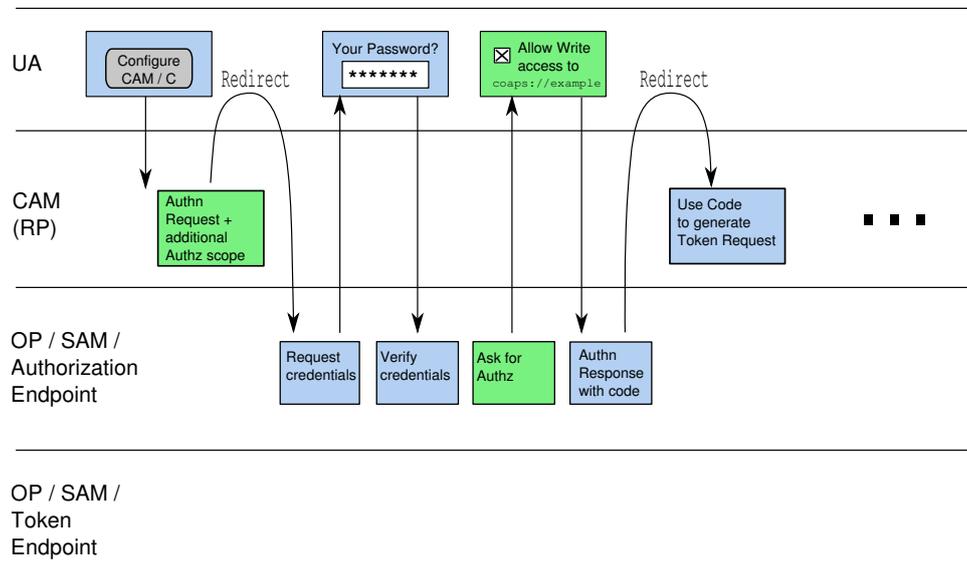


Figure 15.3.: Part 1: Authentication with OpenID Connect (first published in the PDF version of draft-gerdes-ace-dcaf-examples-00, p. 6)

CAM sends an authentication request with an additional authorization scope to Sarah's UA, the browser. The UA redirects the message to the OpenID Provider, which also acts as an OAuth 2.0 Authorization Endpoint. After OP authenticated Sarah and requested her authorization, it generates an Authentication response that it sends to Sarah's UA, together with an OAuth 2.0 code. The UA redirects the response and code to CAM. CAM uses this information to generate a token request to the OAuth 2.0 token endpoint that also acts as a DCAF SAM. The token endpoint responds with an ID token, an OAuth 2.0 access token and a refresh token. With this token, CAM can request authorization from SAM. After validating the token, SAM grants authorization to CAM. Instead of issuing an OAuth 2.0 token, SAM may send a DCAF access ticket to CAM that CAM then stores until it is needed. If C then contacts CAM in the usual DCAF protocol flow, CAM provides the ticket to C. By presenting the ticket to Sarah's blog, C is able to prove its authorization. Afterwards, C can regularly provide sensor data to the blog.

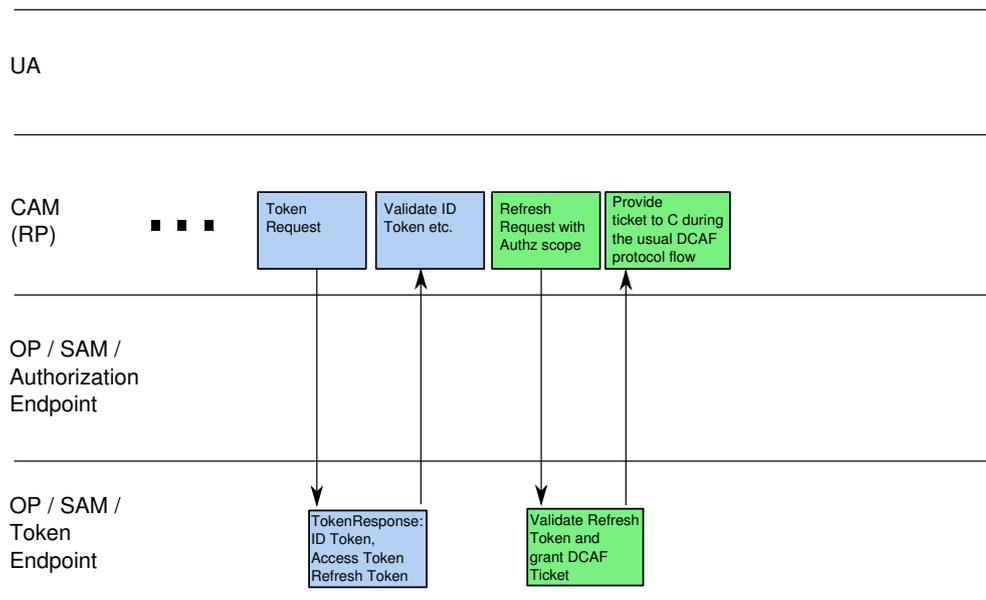


Figure 15.4.: Part 2: Authorization with OAuth and DCAF (first published in the PDF version of draft-gerdes-ace-dcaf-examples-00, p. 6)

## 15.5. Conclusion

In this chapter, we analyzed how smart objects can be integrated into the big Internet. The special characteristics of smart objects such as their large number and ubiquitousness provide new challenges concerning the management and security of these devices. Not all of these challenges can be met by technical means alone; e.g., protecting the access to vast amounts of data in the cloud is also a question of organizational and legal concern and may require regulation.

To avoid silo solutions for the IoT, smart objects must be enabled to interact with less-constrained devices. Where constrained devices require special solutions, adaptations are needed to allow for interactions with devices in the big Internet. We analyzed which requirements DCAF has on underlying security solutions. DCAF can be used with any security solution that meets the given requirements. Even constrained clients and servers that have only limited abilities to measure time cannot be tricked into accepting expired access tickets. DCAF therefore works well with intermediaries.

In many cases, IoT solutions need to co-exist with existing security solutions.

OAuth 2.0 and OpenID Connect are widely used in the big Internet. We presented examples to show how DCAF can be integrated with these solutions. The adaptation is mainly performed by the authorization managers on the less-constrained level. Overseeing principals can use OpenID Connect and OAuth 2.0 as usual with their existing network infrastructure to configure their constrained devices. Constrained devices can perform the usual DCAF protocol flow without modifications. These examples show that IoT devices that use DCAF can easily be integrated with existing security solutions for the big Internet.

## **Part IV.**

# **Conclusion and Outlook**

## 16. Conclusion

In the introduction of this thesis, we discussed Asimov's laws of robotic. In Asimov's stories, these laws were designed to ensure that robots behave correctly and to prevent humans from being harmed. These rules would be useful today, where the next generation of information processing devices consists of often not-so-smart objects with varied hardware limitations, but unfortunately, the devices lack the intelligence required to interpret such rules. In this thesis we developed fundamental principles that empower users to decide what happens with their devices and the physical influence they insert, and that can be applied to contemporary devices. Our authenticated authorization model ANAZEM is designed to define all necessary fundamentals for an effective authenticated authorization: it only allows secure solutions, but does not unnecessarily restrict their design; every secure authorization process is therefore covered by the model. To achieve an effective, continuous protection, the fundamentals must always be satisfied by every endpoint. ANAZEM defines the minimal set of tasks that endpoints must be able to perform to participate in the protection of their overseeing principals' data. All tasks must be performed to achieve a secure solution, but an endpoint may delegate certain tasks to others as long as it satisfies the delegation fundamentals defined in ANAZEM.

Where an endpoint does not have the ability to perform certain tasks by itself that are required to satisfy the fundamentals, it must rely on stronger devices for assistance. In this thesis we therefore developed the task delegation architectural style. It couples each constrained device with a less-constrained device that assists with difficult authentication and authorization tasks. Smart objects with varied hardware constraints are thereby enabled to communicate securely. Depending on the application scenario, the architectural style can be implemented

---

in various architectures. The most general is the four-corner architecture where each constrained device has its own authorization manager. We developed the DTLS profile for the OAuth 2.0-based ACE framework, which implements an architecture with a single authorization manager. We showed that the profile and the framework do not fully support constrained devices, which makes the solutions inflexible and may lead to vulnerabilities. The task delegation architectural style is fully implemented by our Delegated CoAP Authenticated Authorization Framework. In contrast to solutions such as OAuth 2.0 and the ACE framework which were developed for typical Web scenarios, DCAF not only considers the server side but also fully supports constrained and autonomous clients. DCAF's unique design thus enables each constrained device to actively participate in the protection of its overseeing principal's security objectives. Constrained devices initially only require a security association with their own authorization manager, which improves the scalability of the solution. By utilizing the capabilities of the more powerful authorization managers, DCAF enables flexible, secure communication across organization boundaries and thus provides for a true Web of Things. DCAF can be used to protect security associations in the whole life cycle of a constrained device for continuous protection. Also, DCAF can be integrated into existing security solutions for the big Internet as we have shown with prominent examples.

In this chapter we will discuss our findings and contributions. The goal of the thesis and our main research question is: *how can users be empowered to decide what happens with their smart objects and the data that is handled by them in the Internet of Things?*

To answer the question, we divided it into six subordinate questions. In the following, we will discuss how our thesis answers the research questions. We will show for each question how we contributed to answer the question, and explain the value of our contribution.

## 16.1. Defining Characteristics of Smart Objects and their Environments

Our first research question is: *which authentication and authorization problems can be identified by examining common characteristics of smart objects and the application scenarios where they are used?* To answer this question we first performed a literature research to determine typical characteristics of smart objects. Also, we participated in collecting representative use cases from companies and researchers in the field. Our findings were published as informational IETF RFC 7744. The Internet Engineering Steering Group (IESG), that must approve documents before they can become RFC, gave a particularly positive assessment of our document: we received five yes ballots (only one is required) and no rejections, and the document was deemed to be instructive and well written<sup>1</sup>. On average, informational RFCs published in 2016 had 2.23 yes ballots.

Our research shows that the hardware limitations influence the behavior of smart objects and the protocols that they can use. We discussed that smart objects may be constrained in various ways and to various degrees; important aspects are the limited RAM, storage space, energy, processing power, and networking capabilities which impair the devices' ability to store data, send and receive messages, perform calculations, and measure time. Reducing code and package sizes is an important first step, but does not suffice to fully support constrained devices. The devices must also save energy where possible, and require a method for determining the freshness of certain data that does not require each device to have an RTC. The large number of smart objects in the IoT increases the need for scalability and makes manual configuration of devices infeasible.

With the help of the use cases, we illustrated that IoT application scenarios differ from typical Web scenarios: the lack of user interfaces and displays requires the overseeing principals to use other means to communicate with their smart objects; network communication may be the only means to interact with the devices. We concluded that one of the most prominent distinctions between typical Web scenarios and IoT applications therefore is that in the Web, users typically

---

<sup>1</sup><https://datatracker.ietf.org/doc/rfc7744/ballot/>

directly control the clients while smart objects in the IoT often communicate unsupervised, i.e., their overseeing principals cannot intervene in the communication.

From the use cases, we derived requirements for authentication and authorization solutions for the IoT (see section 4.3). These requirements provide a guideline for solution designers, and enable them to develop solutions that are suitable for constrained devices and their application scenarios.

## 16.2. Required Tasks

Our second research question concerns the tasks that are required for authenticated authorization: *Which authentication and authorization tasks must a device at least be able to perform to participate in the protection of its users' data?* The tasks enable us to specify how even constrained devices can communicate securely. To determine the tasks, we first identified the fundamental requirements of the authorization process. Endpoints often require information that they do not have themselves to perform authorization. They therefore must delegate tasks to claim issuers. Since this information influences the authorization, claim issuers must be authorized by the overseeing principals, and the information must be protected accordingly. We therefore also identified the fundamentals that the participating entities must perform for task delegation. From the authorization and task delegation fundamentals we derived the authorization, authentication and task delegation tasks.

The minimal set of tasks that an endpoint must perform represents the lower bound for secure communication. We discovered that all authorization tasks must be performed for an effective authorization; if a task is omitted, the solution may no longer be secure (see section 5.6). Endpoints may delegate the tasks they are responsible for to more powerful devices. But for each task delegation all delegation tasks must be performed. Task delegation therefore may increase the burden on the delegating device and must be applied wisely. At least the attribute validation (task **An1**) is usually delegated, because endpoints cannot perform it by themselves (see section 5.4). To participate in the protection of its overseeing

principals' data, an endpoint therefore must be able to perform the authorization tasks **A2** to **A9**, and the delegation tasks **D2** to **D8** for some device (see section 5.7). An endpoint that does not have the ability to perform the necessary tasks cannot participate in the protection of its overseeing principals' data.

Reducing the effort for performing security tasks is difficult. No task can securely be omitted, and performing the delegation tasks often increases the effort for the device. One way to nevertheless facilitate secure communication for constrained devices is to reduce the number of security associations that an endpoint must establish and manage.

The goal of ANAZEM is to facilitate the design of secure and thus effective specifications of authentication and authorization solutions for solution developers. It aims at being general, i.e., to not unnecessarily restrict the design of the security solution and thus cover every secure authorization process. To fully comply with ANAZEM, a solution must satisfy all fundamentals. While the model as a theoretical construct can depict perfect security, an actual implementation cannot achieve it (see section 5.8). At the same time, a security solution that completely omits a fundamental is not effective. We identified lower bounds that define what solutions must at least do to sufficiently satisfy the fundamentals. A solution that cannot even sufficiently satisfy the fundamentals is not secure.

ANAZEM shows how a secure authorization process must be performed. It clarifies how authentication, authorization and task delegation must interrelate to generate a secure solution: endpoints must authenticate peers and claim issuers to determine if they are authorized, and claim issuers must be authorized by overseeing principals to provide claims to endpoints. By analyzing if the fundamentals were sufficiently satisfied, solution designers are able to detect errors that otherwise might only be discovered after several years. ANAZEM explicitly covers data destination verifiability, a security objective previously missing from the literature, and thereby facilitates the detection of vulnerabilities that may, e.g., enable man-in-the-middle attacks. Understanding the fundamental rules of authorization and authentication thus helps to design effective solutions with continuous security. The list of tasks that is derived from the fundamentals enables solution designers to ascertain that no important details were omitted in

the specification and to discover hidden requirements of the security protocol. Gaps in specifications may lead to vulnerabilities if they are not filled in correctly by implementers. With our analysis of the ACE framework (see also Gerdes et al. 2018a), we have shown that our model allows to detect previously unknown gaps and vulnerabilities in security protocols (see section 11.2). We informed the authors of the ACE framework of our findings, and they have improved and clarified their documents accordingly.

As the example of the DTLS profile and the ACE framework shows, designing solutions without a complete checklist leads to gaps and vulnerabilities. ANAZEM breaks the authorization process down into manageable tasks and enables us to understand the relationships between endpoints, peers, claim issuers, overseeing principals, and the information that they need to exchange to communicate securely.

Identifying the necessary tasks can also enable developers to improve the design of their solutions. E.g., by applying ANAZEM to a typical Web scenario, we have shown that clients rely on the direct interaction with their overseeing principals to perform the tasks **A2** to **A5** for their peers. If clients need to communicate unsupervised, they can only communicate securely if they are enabled to perform these tasks without the interaction with their overseeing principals. This example shows that ANAZEM provides us with valuable insight concerning the design of security solutions.

Determining if a certain task is correctly performed in a protocol is often difficult and requires security knowledge. Defining a more applicable version of ANAZEM (or parts of it), is an interesting topic for future work. Our model may e.g., be used to design or improve formal specifications against which automated checks may be performed.

### **16.3. Architectural Style and Architecture**

Our third research question concerns the architecture of security solutions: *How can architectural design principles that made the Web successful be incorporated in an*

*architecture for the Web of Things which supports smart objects in the protection of their users' data?*

A security architecture that supports constrained devices must consider that the authorization process is only secure if the authorization fundamentals are satisfied by all participating endpoints. By applying ANAZEM to authorization solutions, we discovered that solutions that were developed for typical Web scenarios are designed to protect the access to servers. Much less effort is made for the protection of the client side. The reason for this omission is that clients in Web scenarios are usually directly controlled by their users. The overseeing principals therefore directly perform most of the required tasks. In the IoT, devices often need to act unsupervised and therefore cannot rely on user input at the time of the communication. While it is possible to manually configure clients prior to the communication and frequently update authorization rules, such an approach is not a feasible solution for IoT scenarios that involve large numbers of devices. Decreasing code and packet sizes by e.g., using more efficient encodings, is an important first step to diminish the burden on smart objects, but an authorization solution for a true Web of Things has to do more to truly support constrained devices. With the assistance of our model, we have rethought the authorization process and related authentication to design an architectural style that is best suited for constrained devices. In our task delegation architectural style (see section 9.2), every endpoint that is not able to perform a necessary authentication or authorization task by itself is coupled with a manager that assists with this task. Thereby, the task delegation architectural style ascertains that the security objectives of the overseeing principals of all participating devices are considered.

The architectural style confines the possible architectures. Only in architectures that implement the task delegation architectural style all necessary tasks are performed and secure authorization is possible. The task delegation style is designed as an extension of the REST architectural style. The constrained devices only permanently need to store information about their own authorization manager, which allows the devices to comply with the stateless constraint (see section 3.4). The constrained device and its manager usually belong to the same security domain. Therefore, only devices from the same domain need to be evolved together, which enables separate evolution. The use of an authorization manager relieves

constrained devices from performing difficult tasks. Endpoints only require a security association with their own authorization manager. The AM then assists with the establishment of other security associations. By delegating more difficult tasks such as managing authentication and authorization information for devices from other security domains, the delegator component (i.e., the constrained device) is simplified, the burden on the device is reduced, and the scalability of the solution is increased.

The constrained device is the limiting factor in a communication. Its hardware characteristics and the resulting protocol stack define how such a device can communicate with others. Tasks that can be performed by a constrained device can usually also be performed by a less-constrained device. Communication between two constrained devices is the most difficult to realize. By coupling each constrained device with its own manager, the four-corner architecture enables secure constrained to constrained communication. And not only constrained devices require support. Less-constrained devices that communicate autonomously, i.e., unsupervised, benefit from being coupled with an authorization manager that assists with authentication and authorization. We introduced the four-corner architecture in our previous work on the topic (Gerdes et al. 2015c). Also, the architecture was adopted as a working group document by the ACE working group (draft-ietf-ace-actors-07).

Unless devices are able to perform every task by themselves, devices must use an architecture that implements the task delegation architectural style to communicate securely. Otherwise, important authentication or authorization tasks are not performed. The architectural style enables solution designers to develop a secure solution. The four-corner architecture allows all secure configurations of constrained and less-constrained devices and therefore is the most general architecture. It supports separate evolution better than typical three-corner architectures which require clients to check the authentication and authorization of managers from other security domains. Three-corner architectures that are typical for authorization solutions in the big Internet therefore only support a subset of scenarios. The task delegation architectural style and the four-corner architecture are the basis for a true Web of Things.

## 16.4. Design of the Authenticated Authorization Solution

After specifying the architectural style, we can now answer the fourth research question: *how must an authorization solution be designed to implement the tasks and the architecture?*

To answer this question, we first analyzed various approaches for satisfying the fundamentals. We discovered that both symmetric and asymmetric approaches can be used to perform task **A6**. Only with asymmetric approaches third party verifiability can be achieved. Also, since symmetric keys are shared between entities, solutions must be designed more carefully to ascertain that the symmetric verifier exclusively identifies entities with certain characteristics (see also section 5.4.2.1). Apart from these problems, symmetric approaches are preferable for constrained environments: the length of symmetric keys is shorter and symmetric operations are easier to compute.

We also examined how encryption can be used for data destination verifiability. We discovered that to achieve it, messages must first be encrypted and then signed, or an AEAD algorithm must be used: if the encryption represents the destination, it is part of the claim, and the claim issuer must bind it to the other claim data. Otherwise, the claim is vulnerable to man-in-the-middle attacks. For symmetric algorithms, different keying material must be used for each direction to clearly define the destination.

Other important design decisions concern the validity and freshness of messages. We found that to participate in the protection of their overseeing principals' data, endpoints must be able to measure time in some way. Also, they must be able to communicate with an authorized claim issuer regularly to obtain authentication and authorization updates and/or time information. The time requirements define a minimum hardware requirement for constrained devices: endpoints that are unable to measure the passing of time in some way cannot communicate securely.

We used the requirements we identified in the background part, the authorization and delegation fundamentals of ANAZEM and the identified design choices

to build authenticated authorization solutions for the IoT. By applying the architectural style and the fundamentals and tasks, we built and analyzed candidate solutions. We designed the DTLS profile for the ACE framework, and evaluated its suitability. We showed that the profile and the framework have several unresolved issues that need to be fixed, and that they currently provide only limited support for constrained clients. With DCAF, we developed a solution that fully implements the task delegation architectural style and that supports both constrained servers and constrained clients.

### 16.4.1. DTLS Profile

The ACE framework, developed by the IETF ACE working group, is loosely based on OAuth 2.0. We specified the DTLS profile for the ACE framework. It enables clients to request an access token from the server's authorization manager that the client can then present to the server to establish a DTLS session and gain access to the server's resources. The ACE working group achieved working group consensus on requesting publication of the ACE framework and the DTLS profile as proposed standards in May 2019<sup>2</sup>. They are now reviewed by the Internet Engineering Steering Group (IESG) before they become RFCs.

OAuth 2.0 was designed for typical Web scenarios and controls the access to servers. Accordingly, the ACE framework also focuses on the protection of the server and only implements an authorization manager on the server side. The ACE framework and the DTLS profile can therefore only be used for a limited number of scenarios. Constrained clients are hard-pressed to use the framework and its profiles without additional support. They, e.g., would need to store authorization rules for each possible S (RS), and authorization rules and keying material for each possible SAM (AS).

The security objectives of the client's overseeing principal are not considered by the ACE framework. Without additional measures, the framework can therefore only be used for scenarios where the overseeing principal directly controls the client at the time of access, but not for autonomous clients.

---

<sup>2</sup>[https://mailarchive.ietf.org/arch/msg/ace/T3AXUL6n\\_200-dTKdS007RmBjSl](https://mailarchive.ietf.org/arch/msg/ace/T3AXUL6n_200-dTKdS007RmBjSl)  
<https://mailarchive.ietf.org/arch/msg/ace/gCu-LVhjapA0tpydKDU4mH9xquk>

The ACE framework requires clients to register with SAM. Those devices may not belong to the same security domain, which makes it difficult for two entities that do not previously know each other to communicate. The required close coupling between C and SAM impedes communication across organization boundaries and separate evolution. Registering every client with every potential SAM is not a scalable solution.

By analyzing the DTLS profile with ANAZEM, we showed that the ACE framework had serious issues. Important implementation details, e.g., how servers must react to missing parameters in the access token, were missing. As it was not clear how C informs SAM with which S it intends to communicate, C may have receive credentials for the wrong S from SAM. Due to our review, most of these issues were addressed (see section 11.2). One of the remaining problems is how C determines the audience that identifies the server. If C and SAM do not have a common understanding how S is identified, C may receive an access token for the wrong server without being able to notice it.

All in all, the ACE framework offers only limited support for constrained devices. Especially autonomous clients will have difficulties to use the framework, since it offers no client authorization manager that assists with authentication and authorization tasks, and reduces the overall burden on autonomous clients.

#### **16.4.2. Delegated CoAP Authenticated Authorization Framework**

We designed the Delegated CoAP Authenticated Authorization Framework (DCAF) that fully supports constrained clients and servers. It is based on our former work on the topic (Gerdes et al. 2014). In DCAF, we implemented the task delegation architectural style: each constrained device is coupled with a less-constrained authorization manager that assists with difficult tasks. DCAF can be used with a three-corner architecture, and is—to the best of our knowledge—the only framework that implements the four-corner architecture. DCAF therefore is very flexible.

In DCAF, constrained devices only need to have a security association with their own AM, not with those from other security domains. As less-constrained devices, the AMs can more easily establish security associations with AMs from

other security domains. They are more responsive and may have user interfaces and displays. Overseeing principals therefore can more easily interact with them. AMs vouch for the attributes and maybe also the keying material of the constrained devices for which they are responsible. DCAF allows the use of unique device identifiers as attributes, but does not require them. Thereby, DCAF can help to protect the privacy of the communication partners.

Constrained devices must be provisioned with the necessary authentication and authorization information to be able to participate in the protection of their overseeing principals' data. Without an automated solution, static configurations and manual provisioning are the alternative. If static configurations are never updated, the **rule completeness fundamental** and the **claim completeness fundamental** are not sufficiently satisfied: the solution is not secure. Manual provisioning is tedious and not scalable, and not a feasible solution for the IoT, where large numbers of devices need to be maintained. In DCAF, the process of provisioning authorization rules to the client is integrated in the protocol flow. No additional messages are required. DCAF thereby fully supports autonomous (unsupervised) clients and servers. Since DCAF offers authorization on the client side, the security objectives of all participating overseeing principals are considered.

DCAF provides clients and servers with all data that they require for authentication and authorization. It not only conveys keying material but also all required semantic information that enables the devices to validate the data. DCAF introduces three validity options that allow constrained devices to check the freshness and validity of authentication and authorization data. Even constrained devices with only a limited ability to measure time are supported, and no additional time synchronization mechanism is required. DCAF thus provides a mechanism that allows even those constrained devices to determine if and how long authentication and authorization information is still fresh.

In RESTful architectures, servers are especially prone to DoS attacks since they are the passive part in the communication. Providing a resource to which clients convey their access tickets increases the risk of DoS attacks: attackers may try to flood servers with access tickets. To reduce this problem, DCAF introduces a mechanism where the access ticket is transported in the DTLS handshake. In this

case, servers can discard the ticket if the client is not able to finish the handshake. Additionally, DCAF describes in detail how access tickets must be managed by the server. Only the most recent access ticket for a communication is stored. Deprecated tickets are discarded, which reduces the burden on the server.

For the design of our framework, we analyzed every authentication and authorization task that the participating actors must perform. By applying ANAZEM to the protocol, we showed that each of the necessary tasks for authenticated authorization is covered by DCAF: it is either directly specified, or DCAF demands that the task must be performed outside the protocol flow. Expectations concerning existing security associations between actors and requirements for underlying protocols are clearly specified. DCAF thereby enables developers to avoid gaps and vulnerabilities in the overall security solution.

DCAF's design addresses the smart objects' limited ability to store large amounts of keying material and related semantic information: the devices only need to store the keying material for their authorization manager. By coupling each constrained device with its own manager, smart objects can participate in security mechanisms that would otherwise be very difficult to realize, such as the secure distribution of keying material for group communication or of notifications for software updates. We introduced a revocation mechanism for DCAF based on observe where the AMs collect revocation information from other security domains, and constrained devices only need to observe their AMs' revocation resource. Additionally, DCAF allows clients to delegate server and SAM discovery to CAM: a client can send an empty access request to CAM to receive all necessary information to communicate with a server of COP's choosing. Offloading these tasks significantly simplifies the client's design. DCAF's support for constrained devices therefore goes beyond reduced code and messages sizes.

We designed an open source implementation of DCAF<sup>3</sup> where the main components, i.e., the constrained devices and their managers, were implemented as a proof of concept (see also section 12.8).

To determine DCAF's suitability for constrained environments, we analyzed the messages handled by C and S, and measured code and data sizes of our imple-

---

<sup>3</sup><https://dcaf.science>

mentation. We showed that DCAF messages are similar in size to ACE framework messages with comparable security. The code and data sizes of our DCAF server implementation is smaller than an implementation of the ACE DTLS profile server. At the same time, DCAF provides more support for authenticated authorization on the client side: with the client authorization manager, COP's authorization decisions can be enforced. As described above, clients can also delegate server and SAM discovery to CAM. Dynamic discovery as well as the secure distribution of authentication and authorization information can be provided within the DCAF protocol flow without increasing code and message sizes. The client's complexity can be kept at a minimum without the need to sacrifice flexibility.

DCAF allows for secure constrained to constrained communication. Coupling each constrained device with its own authorization manager allows for secure RESTful communication across organization boundaries. DCAF thereby enables a true Web of Things.

## 16.5. Life Cycle

Authorization solutions usually focus on the operation phase. But for continuous security, the whole life cycle of smart objects must be protected. Our next research question therefore is: *How can secure authorization transitions be realized to achieve continuous protection of smart objects in their whole life cycle?*

For an effective authorization solution, security associations must securely be initiated, managed and terminated. The relationship between the constrained device and its authorization manager must be particularly well protected because of the important role the AM has for the constrained device: authorization can only be effective if this relationship is securely established and managed. For continuous security, authorization transitions, where the AM of a device changes, must be carefully designed and protected. We analyzed each phase in the life cycle of a constrained device and explained the organizational and technical measures for securing the authorization transitions. The introduced mechanisms are based on earlier work on the topic (Bergmann et al. 2012a; Gerdes et al. 2015d).

A common way to realize authorization transitions is to rely on physical contact, e.g., plugging a USB cable into the device or interacting directly with the device's user interface. In the IoT, smart objects may be located in places where they are publicly accessible, or installed where they are not accessible at all, e.g., inside a wall or the human body. Also, manual provisioning is not a feasible solution if large numbers of devices need to be configured. We showed how DCAF can be used to protect the authorization permission, i.e., the security association to the constrained device's AM. The usual DCAF protocol flow can be used for required changes: no additional protocol is required. In most cases, the AM can act in its usual role as mediator between the overseeing principal and the smart object, which makes the configuration more comfortable. Authorization transitions can be automated with DCAF, which provides a scalable solution.

DCAF can also assist with other security-related problems in the life cycle of a smart object: over time, problems in the software itself or the cryptographic algorithms become known which lead to vulnerabilities. Throwing devices with outdated software away cannot be the only solution in the IoT, where the smart objects are expected to stay active for 10 or more years. We showed that the coupling with the AM provides a feasible solution for triggering software updates: the AM can alert the constrained device if an update is available.

We have shown that using the task delegation architectural style where each constrained device is coupled with its own authorization manager facilitates continuous security in the whole life cycle of a smart object. With DCAF, constrained devices can protect their security associations in all phases of the life cycle. The amount of code that is required for security can thus be reduced. DCAF offers a lightweight, automated and scalable security solution for the Internet of Things that provides continuous security.

## 16.6. Integration

As we have discussed in this thesis, smart objects with limited hardware resources have difficulties to use protocols that are common in the big Internet.

Nevertheless, constrained devices must be enabled to communicate with less-constrained devices to avoid a silo solution for the Internet of Things. Our last research question regards the integration of smart objects into the big Internet: *How can constrained devices using light-weight IoT Security solutions communicate securely with devices in the big Internet?*

There are multiple interesting facets to this question. Less-constrained devices may use constrained-level protocols. DCAF is very flexible and can be used in various architectures. With DCAF, we provided a framework that can be used to protect the communication between constrained and less-constrained devices that use the same protocols for their communication.

The protocol stack used by constrained devices in the IoT and less-constrained devices in the big Internet often differs. IP is the unifying factor that abstracts from the layers below. But differences in the upper layer protocols may still make the integration of smart objects difficult. In cases where the less-constrained devices cannot be bothered to install constrained-level protocols such as CoAP or DTLS, translating proxies may provide a solution. Intermediaries that can access selected application level data (see section 10.1.5) can be realized with object security solutions. In DCAF, the requirements for the underlying security solution are clearly defined. All security solutions that meet these requirements can be used with DCAF, including transport layer and object security solutions. DCAF's validity options were designed in a way that the validity period of access tickets and client information cannot be prolonged by withholding messages. Our framework therefore works well with intermediaries.

The use of cloud services is attractive for the IoT since it allows users to easily view and manipulate their data. But cloud services also provide a threat to the security of their users' data: the accumulation of sensitive data make cloud servers attractive targets for attackers. Also, the cloud providers themselves may have an interest in the data which goes beyond the offered service and makes them act against their customers' wishes. We have shown that with a fixed set of predefined authorization rules, the risk of abuse from less trustworthy authorization managers in the cloud can be mitigated (see section 15.1).

A very important aspect for the deployment of IoT devices is the use of existing

infrastructure. To avoid additional effort for maintaining smart objects, their authorization solution must fit to the existing security solution. Because of its flexible design, DCAF can be integrated into existing security solutions for the big Internet as we have shown with the prominent examples OAuth 2.0 and OpenID Connect. The advantages of both worlds can thereby be combined. We have demonstrated how OAuth 2.0 tokens can be translated to DCAF tickets and explained how the protocol flows fit together. Thus, devices that are too constrained to use security protocols on the less-constrained level, can nevertheless be integrated into the existing security infrastructure if they use DCAF. Its flexibility, its true support of constrained devices on both the client and the server side and its easy integration make DCAF the optimal choice for the Internet of Things.

## 16.7. Standardization

Much of the research for this thesis was conducted to guide the standardization efforts concerning authentication and authorization for constrained environments in the IETF. The concepts and implementations presented in the dissertation are therefore designed for interoperability and integration with other Internet protocols. Our contributions to the security of constrained environments led to the nomination to serve on the IETF Internet of Things Directorate <sup>4</sup>.

In the course of the work, we participated in founding the Authentication and Authorization in Constrained Environments (ACE) working group and helped defining the scope and goals of the WG by contributing to the charter <sup>5</sup>. Moreover, the research presented in this thesis substantially influenced the core documents of the ACE working group. The contributions include authorship and editorial work for the ACE use cases document (RFC 7744). The architecture proposed in this thesis was adopted by the working group (draft-ietf-ace-actors-07). Our DTLS profile for the ACE framework (draft-ietf-ace-dtls-authorize-13) was also adopted by the working group, and currently awaits publication as RFC.

---

<sup>4</sup><http://trac.tools.ietf.org/area/int/trac/wiki/IOTDirWiki>

<sup>5</sup><https://datatracker.ietf.org/wg/ace/charter/>

## 17. Outlook

The model for authenticated authorization evaluation described in this thesis aims at assisting solution designers and developers to understand the security requirements of their solution and enables them to detect omissions. Protocol designers that apply ANAZEM during the development could avoid gaps and vulnerabilities from the beginning. Also, describing in the protocol specification which tasks are performed by the protocol itself and which tasks additionally need to be performed outside the protocol would help readers to understand the purpose of the protocol and its role in the overall solution. Choosing the required building blocks for their solution would then be easier for solution designers. Analyzing more protocols with ANAZEM is an interesting topic for future work.

In the model's current form, much security knowledge is required to decide if the model's authorization and delegation fundamentals are satisfied by a solution. Future research should pursue the question how ANAZEM – or parts of it – can be simplified for less experienced solution designers and implementers. Developing a tool box or improving existing tools would be a valuable next step for analyzing solutions and thus improving their security.

Another topic for future research is the update frequency for authorization and authentication data. In section 5.8, we discussed how the fundamentals can be satisfied. To sufficiently satisfy the **rule completeness fundamental** and the **claim completeness fundamental**, authorization and attribute claims must be frequently updated. However, determining the required update frequency is difficult as it depends on various factors. Overseeing principals would benefit from a method that assists them with defining the acceptable update frequency for their application scenario.

This thesis focused on the secure distribution of authorization rules and corresponding keying material. Before the authorization process can be initiated, the client first needs to discover a server that is suitable for communication. There are several mechanisms that allow a client to find servers, e.g., CoAP service discovery or resource directories (see section 3.5.2). But with these methods, a large number of servers may be discovered, and not all of them may be authorized by COP. To avoid the overhead of initiating the authorization process for each of these potential servers only to discover that they are not approved, it would be useful to include semantic information in the resource description in the form of attributes that are meaningful for the authorization. An interesting topic for future research therefore is how service discovery and resource directories may assist unsupervised clients (or their authorization managers) to find authorized servers. In particular, it would be interesting to determine which information a resource description or a resource directory entry would need to contain and how it must be protected.

The ACE framework currently offers only little support for unsupervised clients. These clients will have difficulties to communicate securely, which threatens the overall security of the solution. We designed DCAF to use parameters that are similar to those of the ACE framework. To enable constrained clients to securely interact with the ACE framework, future research should focus on developing a profile that implements DCAF for the ACE framework.

**Part V.**

**Appendix**

## Appendix A.

# Comparison between ANAZEM and the BAN Logic

The BAN logic (Burrows et al. 1990) is a prominent example for the use of logical inference to determine the security of authentication protocols. It provides statements, and rules to derive knowledge from these statements. To show the capabilities of ANAZEM, we will compare it to the BAN logic. Our model does not use logical inference. The fundamentals and tasks cannot easily be compared to statements and derivation rules. We therefore analyze the authentication protocol Kerberos with both approaches and compare the results.

In Kerberos, a third party, the Kerberos Key Distribution Server called  $S$ , is used to negotiate a shared secret between two communication partners  $A$  and  $B$  (Burrows et al. 1990, p. 25). If we apply ANAZEM,  $S$  is performing attribute validation (task **An1**) as a delegation task for  $A$  and  $B$ . The goal is to securely distribute the symmetric key  $K_{AB}$  to  $A$  and  $B$ . This key is supposed to act as a verifier for  $A$  to  $B$  and as a verifier for  $B$  to  $A$ . To securely delegate a task in our model, the Kerberos clients ( $A$  and  $B$ ) as the delegators must perform the tasks **D2** to **D8**. Claim issuer  $S$  must perform task **D0** and task **D1**, and the delegated task **An1** for both  $A$  and  $B$ .

The protocol flow is given below<sup>1</sup>.  $T_S$ ,  $T_A$  and  $T_B$  are timestamps generated by  $S$ ,  $A$  and  $B$ , respectively.  $L$  is a lifetime defined by  $S$ .  $K_{AB}$  is a shared key for

---

<sup>1</sup>The Kerberos protocol flow looks different today; Burrows et al. describe a former version in their work.

A and B.  $K_{AS}$  and  $K_{BS}$  are the keys shared between A and S, and B and S, respectively.

1.  $A \rightarrow S : A, B$
2.  $S \rightarrow A : \{ T_S, L, K_{AB}, B, \{ T_S, L, K_{AB}, A \} K_{BS} \} K_{AS}$
3.  $A \rightarrow B : \{ T_S, L, K_{AB}, A \} K_{BS}, \{ A, T_A \} K_{AB}$
4.  $B \rightarrow A : \{ T_A + 1 \} K_{AB}$

Additionally, Burrows et al. list a number of statements that they assume to be true for the analysis:

1. A believes that it shares  $K_{AS}$  with S and can communicate securely with it.
2. S believes that it shares  $K_{AS}$  with A and can communicate securely with it
3. S believes that A shares  $K_{AB}$  with B and can communicate securely with it
4. A believes that S controls keys shared between A and B.
5. A believes that  $T_S$  is fresh.
6. B believes that it shares  $K_{BS}$  with S and can communicate securely with it
7. S believes that it shares  $K_{BS}$  with B and can communicate securely with it.
8. B believes that S controls keys shared between A and B.
9. B believes that  $T_S$  is fresh.
10. B believes that  $T_A$  is fresh.

We will retrace the analysis of Burrows et al. and compare it with ANAZEM.

## A.1. Message 1

Burrows et al. omit the first message in their analysis since it “does not contribute to the logical properties of the protocol”. In this message, A informs S about the peer for which it requires attribute validation and thereby transmits the input parameter B for the task. To perform task **D0**, S must check if the sender is authorized to provide it. Since the message is not protected, S cannot validate that it actually stems from A. But S transmits parameter B back to A in message 2. A is therefore able to notice if the input parameter is wrong. Thus, everyone is authorized to send message 1.

## A.2. Message 2

In message 2, S sends a message to A that comprises two parts: it contains a claim for A ( $\{T_S, L, K_{AB}, B\}K_{AS}$ ), and nested into it a claim for B ( $\{T_S, L, K_{AB}, A\}K_{BS}$ ). For A, S claims that the entity with verifier  $K_{AB}$  has the attribute B. In B's part, S claims that the entity with the attribute A has the verifier  $K_{AB}$ . S is responsible for task **D0**, the delegated task **An1** and task **D1** for A and B.

S must perform attribute validation (task **An1**) for A and B before generating message 2, must validate that  $K_{AS}$  is a verifier for A (is only known to A and S), and must validate that  $K_{BS}$  is a verifier for B. Miller et al. make suggestions how this process may look like (Miller et al. 1987, p. 9). However, these measures are not part of the analyzed protocol and are not validated here. We therefore define **Assumption 1: S has performed or delegated attribute validation for A and B, and validated that  $K_{AS}$  is a verifier for the entity with attribute A, and that  $K_{BS}$  is a verifier for the entity with attribute B (task An1)**. If this assumption is not true, the security of Kerberos may be breached. This assumption is similar to the assumptions 2 and 7 of Burrows et al.

To comply with the **claim issuer participation fundamental**, S must validate A's authorization to receive the message (task **D0**). S therefore must perform the authorization tasks for A. Since S performs the delegation for A and B, S must participate in protecting the security objectives of their overseeing principals ( $Princ_A$  and  $Princ_B$ ). S might also have its own overseeing principal  $Princ_S$ .  $Princ_A$ ,  $Princ_B$  and  $Princ_S$  directly or indirectly define which entities are authorized to use S's service. How the authorization rules for A are obtained and validated is not part of Kerberos and cannot be validated. To make the protocol secure, **Assumption 2** must be correct: **S obtained and validated authorization rules for the entity with the attribute A, and validate that this entity is authorized to use S's service (tasks A2 to task A5, A7 and task A8 for task D0 concerning A)**. If A is authorized to use S's service, A is authorized to receive message 2 if it actually has the attribute A.

Since  $K_{AB}$  is a shared key, its confidentiality must be protected. Only A, B and S must know this key. Otherwise, an unauthorized entity might get hold of the

key and the protocol fails. Burrows et al. do not consider in their analysis that S must perform task **D0** before sending message 2. The BAN logic therefore cannot detect confidentiality problems. If S has validated that A has the verifier  $K_{AS}$  (**Assumption 1**) and A is authorized to use the service that S provides (**Assumption 2**), S enforces the authorization (task **A9**) by encrypting message 2 to A with  $K_{AS}$  (task **A6**), as long as the used encryption is secure. We therefore define **Assumption 3: The cryptographic mechanisms that are used in the protocol are secure.**

To perform task **D1**, S must bind the required claim information together and provide an endorsement for A. The claim statement is the attribute B. S specifies the destination by using the key  $K_{AS}$  that A and S share to encrypt the message. The holder is defined by the verifier  $K_{AB}$ . Additionally, the claim for A contains information that A can use to determine the freshness and validity of the message. S binds the statement, destination, holder, and additional information together by encrypting the claim with  $K_{AS}$ . A key that is shared between two entities can only provide limited data destination verifiability, since a message that is encrypted with the key could have been sent by each of these entities. Since A does not send a message that is similar to message 2 from S, A is able to notice if an attacker reflects a message back to A. Implementers must make sure that A does not accidentally accept its own message. To more fully satisfy the **claim destination fundamental**, S could explicitly mention A in the claim.

Additionally, S must validate B's authorization to receive the part of the message that A will later relay to it. To do so, S performs the authorization tasks for B: S must validate that B actually has the attribute B, and that S's overseeing principals allow the distribution of session keys to B. Kerberos is only secure if **Assumption 4** is correct: **S obtained and validated authorization rules for the entity with the attribute B and validated that this entity is authorized to use S's service (tasks A2 to task A5, task A7 and A8 for task D0 concerning B).**

If **Assumption 1** and **Assumption 4** are correct, S enforces B's authorization (task **A9**) to read the contents of part 2 of message 2 by encrypting this part of the message with  $K_{BS}$  (task **A6**). S must bind the claim information together for task **D1**. The message for B contains attribute A as the claim statement, the key  $K_{AB}$  as the

holder's verifier, and some additional information that enables B to determine the validity and freshness of the claim. S binds the information together by encrypting it with  $K_{BS}$ , that also represents the claim destination. The shared key  $K_{BS}$  can only provide limited data destination verifiability. Since B does not send messages that are similar to message 2 from S, B is able to notice if it receives its own messages. S could more fully satisfy the **claim destination fundamental** if it would explicitly mention B in the claim.

### A.2.1. Validating the Claim in Message 2

Burrows et al. apply the message-meaning rule to the message to determine its origin. In ANAZEM, A must validate that the pieces of information that are required for the claim validation relate to each other and stem from the same claim issuer (task **D3**), and that this claim issuer is authorized to be a claim issuer (task **D4**).

The message-meaning rule defined by Burrows et al. states that if an entity P believes that it shares a key with another entity Q, and P sees a message that is encrypted with this key, P can derive that Q actually said this message at some point. Applied to Kerberos, A can believe that S is the origin of the contents of message 2 since S encrypted it with  $K_{AS}$  and A believes that it shares this key with S.

In ANAZEM, A must validate that the statement, destination, holder and the lifetime and timestamp relate to each other and stem from the same claim issuer to perform task **D3**. Message 2 contains the attributes of the holder (B) and its verifier ( $K_{AB}$ ), and is encrypted with  $K_{AS}$ . A can validate that the information in the message stems from the same entity, as long as **Assumption 3** is correct and the encryption is secure.

### A.2.2. Freshness and Validity of S's Claim in Message 2

A must validate the freshness and validity of the claim. In the BAN logic, A must believe that a message is fresh to be entitled to believe that S believes what is

stated in the message. In ANAZEM, we distinguish between the validity of the claim which is part of task **D2**, and the freshness of the message which is part of task **D4**.

Without the freshness validation, an attacker could intercept the message from S and replay an old message from S to A. Thereby A could be tricked into accepting an old session key  $K_{AB}$  which could have been compromised in the meantime. The Needham-Schroeder protocol (Needham and Schroeder 1978) used random nonces instead of timestamp and lifetime and was vulnerable to such attacks. Denning and Sacco showed that this problem can be mitigated by the introduction of timestamps (cf. Denning and Sacco 1981, p. 534). Kerberos is derived from the Needham-Schroeder protocol and considers the proposed modifications (RFC 4120, p. 5).

The timestamp and lifetime that are contained in the message are used to validate the freshness and validity of the claim. Burrows et al. combine the timestamp and lifetime that S generated for A to the timestamp  $T_A$ , and list in their assumptions that A is entitled to believe that  $T_A$  is fresh. By applying their nonce-verification rule they can then derive that A is entitled to believe that S still believes the content of the message.

In ANAZEM, A obtains S's attribute claim about B by receiving message 2. To perform task **D2**, A must determine if it has received all relevant currently valid claims. In Kerberos, clients communicate with a single Key Distribution Server S (Miller et al. 1987, p. 15) and thus only the claims of this server need to be obtained.

A can validate that the timestamp and lifetime are bound to message 2 (see section A.2.1). As long as A and S have clocks that are time-synchronized, A can determine the validity of the message using the timestamp and the lifetime. Also, if several claims exist, A is able to determine the most recent claim. Miller et al. assume that the devices that use Kerberos are time-synchronized (Miller et al. 1987, p. 5). To comply with the **claim issuer authorization fundamental**, entities that provide time information must be authorized claim issuers. We define **Assumption 5: A and S have time-synchronized clocks**.

The lifetime in the claim allows A to calculate when the claim becomes invalid. The claim is therefore not infinitely valid and the **claim completeness fundamental** is sufficiently satisfied. Kerberos cannot completely satisfy the **claim completeness fundamental**. The longer the lifetimes, the greater the risk that B or  $K_{AB}$  were compromised without A's knowledge. The degree to which the fundamental is satisfied could be improved by short lifetimes or by introducing a revocation mechanism.

For task **D4**, A must perform the authorization tasks for S. Task **A6** requires A to validate that message 2 actually stems from S and is not a replay (see section 5.2.3.3). If **Assumption 5** holds, the timestamp allows A to determine if the message is fresh (see also section 10.3).

### **A.2.3. Validating S's Authorization to Send Message 2**

S must be authorized to provide session keys for the communication between A and B. The BAN logic uses the jurisdiction rule to derive that A is entitled to believe that  $K_{AB}$  is a session key for communicating securely with B. In ANAZEM, task **D4** must be performed before A can accept the claim from S.

The BAN logic focuses on authentication and does not explicitly mention authorization. But the authors state in their fourth assumption that S controls keys shared between A and B (Burrows et al. 1990, p. 27). S thus is an authority for these keys and "is to be trusted on this matter" (cf. Burrows et al. 1990, p. 20). This actually means they assume that S is authorized to provide these keys.

The jurisdiction rule states that if P is entitled to believe that Q is an authority for X and that Q still believes X, then P is entitled to believe X (Burrows et al. 1990, p. 22). By applying the jurisdiction rule in their analysis, the authors derive that if S stated  $K_{AB}$  and S is an authority for session keys for A and B, A is entitled to believe that  $K_{AB}$  is a session key to communicate with B.

In ANAZEM, A must perform the authorization tasks for S to validate that S is authorized to provide the claim in message 2 (task **D4**). How A performs these tasks for S is with the exception of task **A6** not part of Kerberos and thus cannot

be analyzed. We define **Assumption 6: A has obtained and validated that the entity with the attribute S is authorized to perform attribute delegation for A (tasks A2 to A5, A7 and A8)**. This is similar to the fourth assumption of Burrows et al. Additionally, we define **Assumption 7: A has securely obtained  $K_{AS}$  that is only known to A and S and relates to the entity with the attribute S (task An1)**. This is similar to assumption 1 of Burrows et al.

It will be very difficult to generate a message with meaningful content without the knowledge of  $K_{AS}$ . Since  $K_{AS}$  is a shared key, A might have encrypted the message itself. However, since A never sends a similar message in Kerberos, it can determine that it is not the origin of message 2. Encrypting the statement with a shared key is a weak proof that S actually stated the claim as demanded by the **claim origin fundamental**. Thus, A has a weak assurance that S provided the information in the message, including  $K_{AB}$ , and can perform task **A6**. The **correlation fundamental** could be more fully satisfied if the sender of the message is explicitly mentioned in the claim.

If **Assumption 6** and **Assumption 7** hold, A can validate that S is authorized to send message 2 by decrypting it with  $K_{AS}$ .

#### A.2.4. Destination Validation

A must validate that the message was really meant for it (task **D5**). If A does not perform this task, it might be tricked into accepting a session key that was meant for a different communication. E.g., if A received  $T_S, L, K_{BC}, B$ , it would assume that it shares  $K_{BC}$  with B. Burrows et al. do not consider data destination verifiability in their logic.

If the message was encrypted with  $K_{AS}$  and **Assumption 7** is correct, A can determine that the message was either meant for itself (if S is the origin) or for S (if A sent the message itself). Since no protocol message that A sends to S equals message 2, A never sends such a message. A therefore has a weak assurance that it actually is the destination of the message and can perform task **D5** by decrypting the message with  $K_{AS}$ . As stated above, the **claim destination fundamental** could be more fully satisfied if A was explicitly mentioned as the destination of the claim.

### A.2.5. Evaluating the Claim

As stated above, clients in Kerberos only communicate with a single Key Distribution Server. A therefore only needs to consider the newest claim from S (task D8).

### A.3. Message 3

Burrows et al. only consider B's side when analyzing message 3. But to participate in the protection of the overseeing principals' security objectives, A must validate that B is authorized to receive the message before sending it to B. The first part of the message stems from S and is only relayed by A. Since this part of the message is already protected by S, everyone is authorized to receive it.

For the second part of the message ( $\{ A, T_A \}_{K_{AB}}$ ), A generates a timestamp and encrypts it together with its own attribute A. A should make sure that it really wants to communicate with B before sending message 3, since there is no reason to finish the authentication process otherwise (message 1 was not encrypted, it might have been sent by anyone). The goal of the Kerberos protocol is to securely distribute a session key between two clients. An entity is therefore authorized to receive message 3 if it actually has the attribute B. If **Assumption 3** and **Assumption 5** to **Assumption 7** hold, A securely obtained  $K_{AB}$  as a verifier for B. A ensures that only B can read the nonce by encrypting it with  $K_{AB}$ .

With message 3, A proves that it can use  $K_{AB}$  and claims that its timestamp is  $T_A$ . The claim statement is  $T_A$  and the destination is set by defining the source A. The claim has no holder. A endorses the claim by encrypting it with  $K_{AB}$ .

Including attribute A in the message helps to distinguish this part of the message from message 4 and thus makes it possible to validate the source and destination of message 3 and 4.

### A.3.1. Validating S's Claim

B must validate the origin of the claim in the first part of message 3. Burrows et al. use the message meaning rule for that. In ANAZEM, task **D3** must be performed to determine if all relevant claim information is bound together.

Burrows et al. assume in their sixth assumption that B is entitled to believe that it shares  $K_{BS}$  with S. The message meaning rule applies: If B shares  $K_{BS}$  with S and sees a claim that is encrypted with  $K_{BS}$ , B can believe that S at some point stated this claim.

In ANAZEM, B must validate that the claim destination, the holder and its verifier, and the timestamp and lifetime are bound together and stem from the same claim issuer. Our model therefore specifies which information a claim must contain. The statement of the claim is the attribute B. The holder of the claim is represented by the key  $K_{AB}$ . S encrypted the claim with  $K_{BS}$ , which is a weak assurance that B is the intended destination. The encryption binds the statement, destination and holder together and represents S's endorsement of the claim. If the encryption is secure (**Assumption 3**), B can thereby perform task **D3**.

### A.3.2. Freshness and Validity of S's Claim

B must validate that the claim by S is fresh and valid. Otherwise old messages can be replayed and B can be tricked into accepting old session keys. Burrows et al. use the nonce-verification rule for that. In ANAZEM, the validity of the claim is validated in task **D2**. The freshness of the message is validated in task **D4**.

In their ninth assumption, Burrows et al. assume that B is entitled to believe that  $T_S$  is fresh. By applying the nonce-verification rule to message 3, they derive that B is entitled to believe that S still believes the claim.

In ANAZEM, B must validate that S still believes the claim. B obtains S' claim about  $K_{AB}$  with message 3 (task **D2**). Timestamp and lifetime are bound to the claim (see also section A.3.1). By validating timestamp and lifetime, B can be

fairly sure that the key is fresh, as long as A and B are time-synchronized. We define **Assumption 8: B and S have time-synchronized clocks**.

B must also determine that it received all relevant claims. Kerberos uses only a single key server, so B only needs to consider claims from S. But although B can validate that the claim from S is fresh, it is not able to determine if a more recent claim exists. The **claim completeness fundamental** may be more fully satisfied if short timestamps are used or if a revocation mechanism is defined.

To perform task **D4**, B must perform the authorization tasks for S. For task **A6**, B must determine if the claim actually stems from S and is not a replay. If assumption **Assumption 8** holds, B can determine if the message is fresh using the timestamp.

### A.3.3. Validating S's Authorization

B must validate that S is authorized to provide the information in message 3, in particular  $K_{AB}$ . Burrows et al. aim at achieving this by applying the jurisdiction rule. In ANAZEM, task **D4** must be performed.

In their eighth assumption, Burrows et al. assume that B is entitled to believe that S controls keys for A and B, and thus is to be trusted on  $K_{AB}$ . The jurisdiction rule applies and yields that B is entitled to believe that  $K_{AB}$  is a valid session key.

In ANAZEM, B must validate S's authorization to provide  $K_{AB}$  to satisfy the **claim issuer authorization fundamental**. B's overseeing principal must have approved authorization rules for S and these rules must actually relate to S. How the authorization tasks for S are performed is not part of the Kerberos protocol and thus cannot be validated. The tasks must be performed outside the protocol. We therefore define **Assumption 10: B has obtained and validated that the entity with the attribute S is authorized to perform attribute delegation for B (tasks A2 to A5, A7 and A8)**. This is similar to the eighth assumption of (Burrows et al. 1990).

To perform task **A6** for  $S$ 's claim in message 3,  $B$  needs to know if  $K_{BS}$  is a verifier for  $S$ . How this is accomplished is not defined in Kerberos. We define **Assumption 9:  $B$  performed or delegated attribute validation for the entity with the attribute  $S$  and validated that  $K_{BS}$  is only known to  $B$  and  $S$  (task  $An1$ )**

It will be difficult to create a meaningful message without the knowledge of  $K_{BS}$ . If the encryption mechanism is secure (**Assumption 3**) and **Assumption 9** is correct,  $B$  has therefore validated that either itself or  $S$  is the origin of the claim in message 3. Since  $B$  does not create a similar message in the protocol, it thereby has a weak assurance that  $S$  stated the part of message 3 that is encrypted with  $K_{BS}$ .  $B$  thus can perform task **A6** for task **D4**.

If **Assumption 9** and **Assumption 10** are correct,  $S$  is authorized to provide the claim in message 3.

#### **A.3.4. Destination Validation for $S$ 's claim**

$B$  must validate that the two parts of the message were actually meant for it (task **D5**) to comply with the **claim destination fundamental**. Burrows et al. do not consider data destination verifiability in their analysis.

The first part of the message is encrypted with  $K_{BS}$ . If a message is encrypted with this key and **Assumption 9** and **Assumption 3** are correct, the message was created either by  $S$  or by  $B$  itself.  $B$  does not create a similar message in Kerberos.  $B$  therefore has a weak assurance that it is the intended destination of  $S$ 's claim in message 3 if it decrypts the message with  $K_{BS}$ .

#### **A.3.5. Evaluating $S$ 's Claim in Message 3**

Kerberos clients only need to consider claims of a single key distribution server (see also section A.2.1).  $B$  therefore only needs to consider the newest claim from  $S$ .

### A.3.6. Validate that S's Claim Relates to A

B validates that it actually is communicating with an entity with the attribute A by validating that the claim provided by S actually relates to A (**D6**). Otherwise an attacker may impersonate A. Burrows et al. do not explicitly mention this in their analysis.

To determine that B actually currently communicates with A, messages to and from A must be protected. For the protection of normal messages, the delegation fundamentals apply (see also section 5.5). B must therefore validate that the claim from A in the second part of message 3 is protected accordingly. If the claim is valid and meant for B, B can perform task **D6** for the claim from S.

#### A.3.6.1. Validating A's claim

The statement, destination and additional information in A's claim must be bound together and stem from the same claim issuer. The claim statement is A's attribute. The destination is provided by the shared key  $K_{AB}$ . The timestamp is additional information that allows B to validate the freshness of the claim. B validates that the information in the claim is bound together by decrypting the message with  $K_{AB}$  (task **D3**).

#### A.3.6.2. Validating Freshness and Validity of A's claim

If **Assumption 5** and **Assumption 8** hold, A and B are time-synchronized with S and therefore are also time-synchronized with each other. If **Assumption 3** holds, the timestamp is bound to A's attribute by the encryption with  $K_{AB}$ . B can thereby determine that the message from A is fresh and not a replay, which is required for task **D2**.

A is only supposed to send a single message 3 in Kerberos. B therefore must only consider this message (task **D8**).

### A.3.6.3. Validating A's Authorization

The goal of the Kerberos protocol is reached if A and B communicate securely using  $K_{AB}$ . Concerning Kerberos, A is authorized to send message 3 if actually has the attribute A. If all assumptions hold, B has securely obtained  $K_{AB}$  from S. Since B does not send a similar message, A can provide a weak assurance that it has  $K_{AB}$  by encrypting its messages with this key, if **Assumption 3** is correct. B validates that A is authorized to send the second part of message 3 by decrypting it with  $K_{AB}$  (task **D4**).

Kerberos is not an authorization protocol. B's overseeing principals may not want B to communicate with A. B must obtain its overseeing principals' authorization rules to communicate with A before exchanging application data with A.

### A.3.6.4. Destination Validation for A's Claim

The claim is encrypted with  $K_{AB}$ . If the key was securely obtained, i.e., if all assumptions hold, only A, B and S can create this claim. S never uses  $K_{AB}$ , and B does not create a similar message<sup>2</sup>. B therefore has a weak assurance that it is the intended destination of the claim by decrypting it with  $K_{AB}$ .

If the second part of message 3 is a valid claim from A and meant for B, B has validated that it is actually currently communicating with A. If B wants to communicate securely with A, it must validate every message to and from A accordingly.

## A.4. Message 4

Before B sends message 4 to A, it must validate that A is authorized to receive this message. Since the timestamp in message 4 is not confidential, everyone is

<sup>2</sup>Message 4 is similar to the second part of message 3, but the latter additionally contains attribute A. Otherwise, B would be not able to distinguish it from a message 4 it sent itself. An attacker that listened in on the communication could then combine message 4 with the first part of message 3 and get  $\{ T_S, L, K_{AB}, A \}_{K_{BS}}, \{ T_A + 1 \}_{K_{AB}}$ . With this message, B would be tricked into generating  $\{ T_A + 2 \}_{K_{AB}}$ , because it would not be able to distinguish the attacker from A. The **claim origin fundamental** would thereby be violated.

authorized to receive it. If B is not interested in communicating with A, it should not send message 4 to A.

B must create message 4 as a claim. The delegation fundamentals apply. The claim does not have a holder. The destination of the message is set by the shared key  $K_{AB}$ . The message does not have statement and therefore is not actually a claim. B additionally provides the timestamp from message three, incremented by one. The purpose of the message is that B shows that it is able to use  $K_{AB}$ . B uses it to bind the information together (task **D1**).

#### **A.4.1. Validate that S's Claim Relates to B**

A must validate that it is communicating with the entity that actually is the holder of S's claim (task **D6**). Otherwise an attacker can impersonate B. Burrows et al. do not consider this in their logic.

A must perform the delegation tasks for messages from B to validate that they are protected, actually stem from B and are meant for A.

##### **A.4.1.1. Validating B's Claim**

To perform task **D3**, A must check if the relevant claim information is bound together and stems from the same entity. If A and B securely obtained the keying material, i.e., all assumptions hold, A performs task **D3** by decrypting the message with  $K_{AB}$ .

##### **A.4.1.2. Validating Freshness and Validity of B's Claim**

Message four contains the timestamp that A sent in message three, incremented by one. Thereby, A can determine if the message is fresh. A's overseeing principals should configure a maximum lifetime. Messages that exceed the lifetime should be discarded.

Using the timestamp, A can validate that the message is not a replay which is required for task **D4**.

B is only supposed to send a single message four. A therefore only needs to consider this message (task **D8**).

#### **A.4.1.3. Destination Validation for B's Claim**

Since the message from A in message 3 contains A's attribute, it can be distinguished from message 4. If **Assumption 3** is correct and the encryption is secure, A has a weak assurance that it is the intended destination of the claim (task **D5**).

#### **A.4.1.4. Validating B's authorization**

Concerning Kerberos, an entity is authorized to send message four if it actually has the attribute B. If all assumptions hold, A and B securely obtained  $K_{AB}$ , and A knows that it shares this key with the entity with the attribute B. Since A never sends such a claim, A has a weak assurance that an entity with the attribute B actually stated the claim (task **D4**).

Before exchanging application data with B, A must validate that B is authorized to send and receive the data. To obtain the authorization rules from A's overseeing principals, an authorization mechanism may be required.

If A performed all delegation tasks for the claim and ascertained that the claim is valid for the communication with B, it has determined that S's claim in message three actually refers to the entity that sent message four. For the subsequent communication with B, A must perform the delegation tasks for every message that it exchanges with B to ensure that it still communicates with B (task **D6**).

## Appendix B.

### DTLS Profile Analysis

In the following, we will use ANAZEM to evaluate the DTLS profile of the ACE framework. We will determine if the profile satisfies the model's authorization fundamentals by analyzing the messages that are exchanged between the actors. The profile does not have to provide a solution for every task, but it must indicate security problems and should suggest solutions. If details are not defined in the profile, e.g., if a certain task is not in scope, we define assumptions that must be met to provide a secure solution. If the assumptions contain actions, we presume that these actions are performed in compliance with the authorization fundamentals. The analysis was performed on version 01 of the DTLS profile draft (draft-ietf-ace-dtls-authorize-01) and version 06 of the ACE framework (draft-ietf-ace-oauth-authz-06). A updated summary of our findings can be found in section 11.2.

Authorization solutions depend on the quality of the cryptographic mechanisms they use, including protocols such as DTLS. Evaluating these mechanism is not in scope for this work. We therefore assume them to be sufficiently secure, unless stated otherwise. We define **Assumption 1: the protocols and cryptographic mechanisms used in the profile are sufficiently secure.**

#### B.1. Unauthorized Resource Request

Before sending a resource request, C must validate that S is authorized to receive the message. When C starts the first communication, it does not yet have a secu-

rity association with S, and is therefore not able to protect the message. Resource requests may contain sensitive data, e.g., the resources C wants to access on S. C must not transmit sensitive request data in the unauthorized resource request.

The DTLS profile should point out that no confidentiality can be provided for data in the unauthorized resource request. It should suggest alternatives such as using resource directories to discover SAM, or provide a special resource for requesting SAM information as described in section 12.2.1 for DCAF.

We define **Assumption 2: C does not transmit sensitive request data in the unauthorized resource request.**

S rejects requests that are not covered by an access token or were received on an unprotected channel. The only exception is the `/authz-info` resource that we will analyze separately in section B.7. S responds with a SAM information message to an unauthorized resource request, but does not process data from the request. Therefore, everyone is authorized to send an unauthorized resource request to S.

## B.2. SAM Information Message

If S received an unauthorized resource request, it sends a rejection message to C that includes information about its SAM. The SAM information message comprises SAM's address as an absolute URI (see RFC 3986). S may also add a nonce to the message for validating later messages from C.

Since S has no security association with C at this point, it cannot protect data it sends to C. Therefore, S must not send sensitive data in the SAM information message. SAM's address and the nonce are not confidential.

C cannot validate S's authorization to send the SAM information message and cannot be sure that the message contains the correct URI. C must establish a security association with SAM, and the data in the SAM information message is not relevant for the authentication. But a manipulated URI might result in an availability problem if C is not able to contact the correct AM.

### B.3. Token Request

SAM provides C with the client information that C requires to authenticate S (see also figure B.1). SAM therefore influences the authorization process, which makes it a claim issuer for C. C therefore must perform the delegation tasks for SAM. If C provides input for the delegation to SAM in the token request, C must protect this information accordingly.

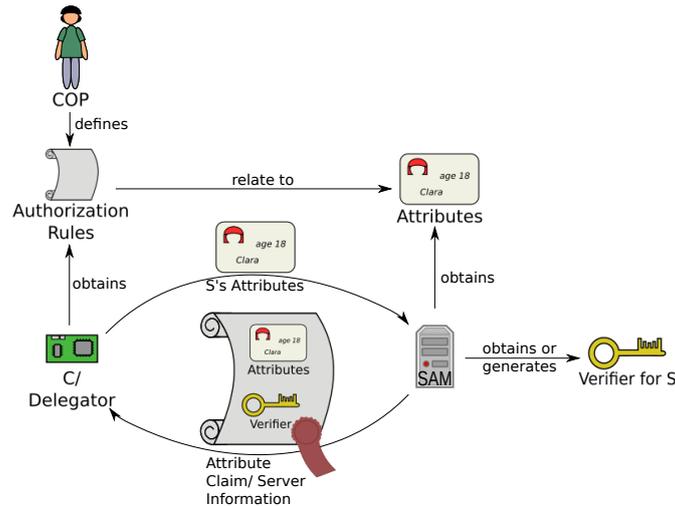


Figure B.1.: SAM Generates Client Information for C

C must send input for the delegation only to authorized entities, i.e., if the token request contains confidential data, C must encrypt it to ensure that only an authorized SAM is able to read it. The token request may contain a key identifier or, in the RPK mode, C's raw public key. This data will likely not be confidential. If C includes a symmetric key in the request, the request is confidential. C might include request data in the message to indicate to SAM which resource it wants to access. This information may be confidential.

C specifies the server that it wants to communicate with in the token request. Since this data is input data for the delegation, C must protect its integrity and authenticity to enable SAM to perform task **D0**. Also, C must determine corresponding access token response stems from an authorized SAM. C might need to make sure that the access token response stems from the same SAM that received

the access token request (see also section B.6). Since the token request may contain confidential data, C should ensure that it can only be accessed by authorized entities.

To validate SAM's authorization, C must obtain authorization rules about SAM from COP. The DTLS profile recommends to provide C with a list of "trustworthy authorization managers" (draft-ietf-ace-dtls-authorize-01, p. 14), but it does not specify which information this list must contain, or how it must be protected. If implemented correctly, COP can use such a list to define which entities are authorized SAMs. To perform task **A1**, COP must validate that SAM is meant to provide access tokens and client information (authorization rule) to C (destination) for resources on S (context) and bind this information to a verifier for SAM (holder), e.g., a symmetric key that C shares with SAM or SAM's RPK. This information must be approved by the overseeing principal, securely be provided to C, and integrity-protected, i.e., no unauthorized entity must be able to alter the list. The profile must provide more details about the information that the list must contain.

We define **Assumption 3: COP authorized SAM to provide access tokens and client information to C for resources on S (task A1)**.

C must obtain the authorization rules (e.g., the list of authorized SAMs) that are currently valid for SAM (task **A2**). C must know if SAM is no longer authorized to be an authorization manager and decide about resources for S, and must be informed if SAM's keying material changes. To sufficiently satisfy the **rule completeness fundamental**, SAM's authorization must only be valid for a certain time and/or C must be able to receive revocation messages. C must validate that the authorization rules were approved by an authorized COP (tasks **A3** and **A4**), and that COP means C to enforce these rules (task **A5**). How these tasks are accomplished is not part of the DTLS profile.

We define **Assumption 4: C has obtained authorization rules for SAM that are up to date (are continuously updated/revoked), were approved by an authorized COP and are meant to be enforced by C (tasks A2-A5)**. The authorization rules refer to a SAM with certain credentials.

C must validate that the SAM it communicates with actually is the entity that is authorized. C must have obtained credentials for SAM and have authorization rules from SOP that refer to an entity with these credentials. For task **A6**, C must validate that SAM can use these credentials. To do so, C must rely on the communication security solution. TLS or DTLS help with task **A6** if the authorization rules relate to the keying material or attribute that is exchanged in the handshake. Object security solutions for ACE are still work in progress at the time of writing in July 2018 and are not in scope of this work. If an object security solution is used, C might validate SAM's authorization to receive the token request by encrypting it with SAM's credentials.

The ACE framework and the DTLS profile provide little detail how the communication between C and SAM must be protected. The ACE framework demands that data exchanged with SAM must be integrity-protected and encrypted, and that C and SAM must perform mutual authentication (draft-ietf-ace-oauth-06, p. 14). The framework defines that, unless specified otherwise by the profile, X.509 certificates are used to authenticate SAM (draft-ietf-ace-oauth-06, p. 15). A constrained client may have difficulties with this approach. The DTLS profile states that C must protect the access token request in the PreShared-Key mode (draft-ietf-ace-dtls-authorize-01, p. 10), but does not provide information about this in the RawPublicKey mode. The profile should define more clearly how the communication between C and SAM must be protected in each mode, in particular, what kind of keying material is used for the protection, and how this keying material is obtained. If the wrong keying material is used, or the keying material was not obtained securely, C may communicate with an unauthorized SAM.

We define **Assumption 5**: **C has validated that the authorization rules refer to the SAM that will get the token request using the attributes or credentials that are used in the communication security solution (task A6).**

To perform task **A7**, C must validate that SAM is authorized to provide claims for this S. The profile does not specify how this is accomplished. Scenarios with multiple overseeing principals are out of scope; no additional effort is required to perform task **A8**.

We define **Assumption 6**: **C** has validated that the context of the authorization rules for SAM matches the current context with SAM, e.g., that SAM is authorized to provide claims concerning this **S** (task A7).

We will evaluate how SAM performs the authorization tasks for **C** in section B.4.1

## B.4. Access Token Generation

**S** is not able to determine **C**'s authorization on its own and delegates the authorization tasks to SAM. SAM must validate **C**'s authorization on behalf of **S**, and perform the delegation tasks **D0** and **D1**. For the latter, SAM generates an authorization claim, i.e., the access token, that can be used by **C** to access **S**. SAM also performs tasks for **C** by providing the client information and the respective access token for the correct server (see also section B.5). **C** specifies the server it wants to communicate with in the token request. SAM must check if this information actually stems from **C** to comply with the **claim issuer participation fundamental**. Therefore, the integrity and authenticity of the access token request must be validated. The server that **C** requested must be the destination of the access token, and the token must be issued to the requesting **C**. The process is depicted in figure B.2.

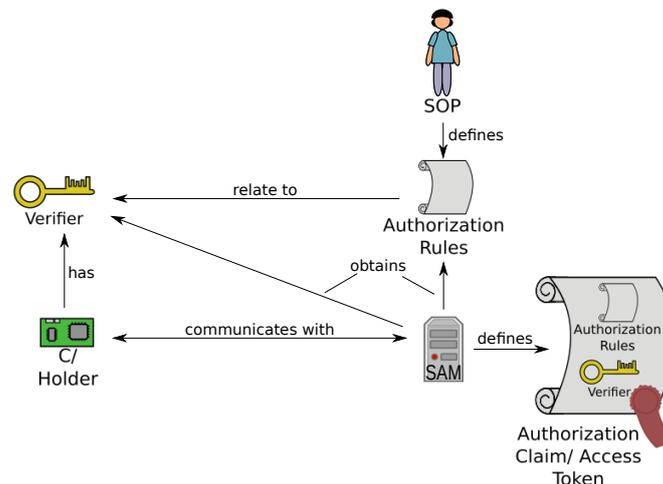


Figure B.2.: SAM generates the Access Token

### B.4.1. SAM Validates C's Authorization

Before sending an access token response, SAM must validate C's authorization:

1. C is authorized to send the information in the access token request (task **D0** of C's task delegation)
2. C is authorized to access a resource on S as requested (delegated task for S).
3. C is authorized to receive the access token response including the client information (task **D0** for C and task **An1** for S).

We will discuss how SAM validates C's authorization to receive the access token response in section B.5.

From SAM's perspective, C is authorized to send an access token request if it is authorized to get an access token for S. C is authorized to get the access token for S if SOP allows it, and if C actually requested the token. SAM must perform the authorization tasks in behalf of S for the C that sent the token request. Also, SAM must validate that it actually is communicating with this C for the access token request and the access token response.

SAM and its overseeing principal SOP must perform the authorization tasks in behalf of S. To perform task **A1**, SOP must define authorization rules for C that specify the actions that C is allowed to perform (authorization rule), the resource on S that is accessed (context), and the SAM that is meant to enforce this rule (destination). This information must be bound to a verifier for C, i.e., a symmetric key that SAM shares with C, or C's RPK. All this information must be endorsed by SOP and integrity-protected.

SAM must obtain the authorization rules that are currently valid for C, i.e., SAM must know if the permissions for C changed or if C is no longer authorized at all (task **A2**). SAM must then validate that the authorization rules stem from an authorized SOP (tasks **A3** and **A4**) and that SAM is meant to enforce the authorization (task **A5**).

The ACE framework specifies that SOP's authorization rules can be preconfigured on SAM, or obtained with an OAuth flow (draft-ietf-ace-oauth-authz-06,

p. 10), but provides no details about this process. We assume that the tasks A1 to A5 are out of scope. The framework or the profile should point out that SAM must check if the obtained authorization rules refer to the C for which SAM generates the access token.

We define **Assumption 7: SAM has obtained authorization rules for C concerning the access to S that are up to date, were approved by an authorized SOP and are meant for SAM (tasks A1-A5).**

As described in section 5.2, the overseeing principals define authorization rules because of certain attributes in the physical world, and a continuous link between the entity that has these attributes and the entity that the endpoint currently communicates with must be made. Task **An1** must be performed to validate that C actually has certain attributes, e.g., that it is located in the living room and belongs to Bob. This task can be delegated, but must be performed at some point. The ACE framework assumes that the client has been registered with SAM, and that SAM thereby obtained credentials for C (draft-ietf-ace-oauth-authz-06, p. 10). The framework does not clearly state what the purpose of this registration is. We assume that SAM thereby obtained an attribute claim that allow to link the authorization rules to an entity with certain credentials, or even an authorization claim that directly links authorization rules to the credentials. The registration process, that must include the delegation of task **An1** and the delegation tasks for the obtained claims, is out of scope for the ACE framework and the DTLS profile.

The framework and the profile should clearly state what the purpose of the registration process is, i.e., that SAM must thereby obtain credentials for C that can be linked to authorization information for C. Only then can these credentials enable SAM to identify an authorized C.

We define **Assumption 8: SAM obtained attribute or authorization claims that are up to date and link C's credentials to SOP's authorization rules for C on S (task An1).**

SAM must check if the token request actually stems from a certain C. TLS or DTLS can help SAM with task **A6** if the authorization rules for C refer to the verifier (the key or attribute) that was transmitted in the DTLS handshake. If **Assumption 8**

holds, SAM obtained authorization rules that refer to C's credentials. C proves to SAM that it actually has the credentials by using them in the handshake. As described in section 10.1.4, TLS and DTLS derive distinct keys for each communication direction from the master secret. Also, TLS and DTLS use sequence numbers for replay protection. SAM can therefore validate that the message stems from C if TLS or DTLS is used between SAM and C.

If an object security solution is used between C and SAM, C must protect the integrity and authenticity of the token request by providing a cryptographic signature or a MAC using its credentials. If the same symmetric key is used for both directions, both entities could generate the MAC. Additional measures are required to ensure that the token request is not a reinserted message that was originally generated by SAM. If C provided a signature, SAM performs task A6 by validating that the authorization rules that were defined by SOP refer to the credentials that C used to create the signature. The object security solution must provide replay protection, and SAM must validate that the token request is fresh.

If the RawPublicKey mode is used between C and S, SAM must obtain C's RPK during the registration process. The DTLS profile specifies that C must provide its RPK in the token request, and that the access token must be bound to this RPK (draft-ietf-ace-dtls-authorize-01, pp. 9–10). However, it does not mention that SOP provided authorization rules for the entity with this RPK, and that the RPK must also be used to protect the integrity and authenticity of the token request. Otherwise, an attacker might, e.g., trick SAM to generate valid access tokens for various clients that can then be used to perform a DoS attack on S. If C uses the RPK in the security solution, there is no need to additionally transport it in the payload of the token request.

In the PreSharedKey mode, SAM generates the PSK. C may request an updated access token from SAM by specifying the PSK that was formerly used between C and S. SAM must validate that the last access token that it issued to C for the communication with S is actually bound to this PSK before it binds the new access token to this PSK. SOP's authorization rules for C must refer to C's credentials, not to the PSK that C shares with S.

The framework demands that the communication between C and SAM is encrypted and integrity-protected (draft-ietf-ace-oauth-authz-06, p. 14). The profile

must additionally mention that C must use the credentials that identify it as an authorized C to ensure the integrity and authenticity of the access token request. Also, SAM might use these credentials to protect the confidentiality of the access token response. For the RPK mode, the credentials must be C's RPK. The profile should demand the use of DTLS or object security for the communication between C and SAM not only in the PSK mode, but also in the RPK mode.

We define **Assumption 9: SAM validates, with the help of the keying material that is used in the underlying security solution, that SOP's authorization rules relate to the C that SAM communicates with and that sent the token request (task A6).**

If SOP defined a context in which the authorization rules for C are valid, SAM must check if the current context fits to the defined context. E.g., SOP may want C to only access certain resources on S. For task A7, SAM must therefore validate that C is allowed to access the resources as requested.

We define **Assumption 10: if context constraints were defined for C's communication with SAM, e.g., that C's permission are only valid for a certain S, SAM ascertained that the current context matches the defined context (task A7).**

Scenarios where multiple overseeing principals define authorization rules for SAM are not in scope of the profile. The ACE framework indicates that only SOP's access control policies are considered (draft-ietf-ace-oauth-Authz-06, p. 50). No additional effort is required for task A8.

#### **B.4.2. SAM Performs the D0 and D1 for S**

The access token may contain confidential information. E.g., it might contain the authorization rules for C, or, in the PreSharedKey mode, the symmetric key for S. SAM must therefore validate S's authorization to get it (task D0, see section 5.4.2). The ACE framework defines that C either specifies the server that it wants to communicate with in the token request, or C and SAM have pre-established a default server (draft-ietf-ace-oauth-Authz-06, p. 16). C and SAM must have a common understanding how the server is specified, i.e., S must have attributes

that are meaningful for the authorization and that are known to SAM and C. To enhance the security and facilitate interoperability, the ACE framework should specify how the server is represented (see also section B.5). It must at least point out the problem and should give examples.

The ACE framework assumes that S registered with SAM (draft-ietf-ace-oauth-authz-06, p. 10), and that SAM and S obtained keying material that allows them to communicate securely, but no details about this process are provided. We assume that this task is out of scope for the framework and the profile.

We define **Assumption 11: SAM validated that SOP authorized the S that C attempts to access to delegate tasks and receive access tokens from SAM (task D0); SAM obtained the respective keying material as a verifier for S (task An1).**

If the access token contains sensitive data, e.g., the symmetric key, only an authorized S must be able to read it. In this case, SAM must encrypt the token before transmitting it via C to S. As described in section 11.1.2, the default token format for the ACE framework is the CWT format. To protect the confidentiality of the access token, SAM may use an encrypted CWT (RFC 8392, p. 10). SAM must use the keying material that it obtained for S for the encryption.

The profile must mention that SAM must encrypt the access token if it contains sensitive data, so that only an authorized S can decrypt it. It should explain how SAM is supposed to do this.

We define **Assumption 12: SAM assured that the information in the access token can only be accessed by an authorized S (task A6 of task D0).**

Task **D1** represents the generation of the access token. To perform this task, SAM must define the statement of the claim, the endpoint for that the claim is intended, in this case S, and the holder of the claim, in this case C. These parts of the claim information must then be bound together and endorsed by SAM.

The purpose of the DTLS profile is to enable S to validate using the access token if C is authorized to access a resource on S. The token therefore must contain the information that S requires as the claim statement. As described in section 11.1.2, access tokens in the ACE framework contain a `scope` parameter that indicates

authorization rules. SAM must have obtained authorization rules concerning C for S from SOP, and use these rules to define the scope for the access token (task **D1a**). If binary authorization (see section 2.4) is used, the scope may be omitted. Neither the ACE framework nor the DTLS profile offer recommendations on how SAM must define the scope. Only for the PSK mode the framework recommends the access tokens to be scoped to specific permissions (draft-ietf-ace-oauth-authz-06, p. 33).

The ACE framework and/or the DTLS profile must mention that the access token must contain authorization rules defined by SOP or a reference to them in the token's scope, unless the token is only meant for binary authorization or a default scope was defined.

We define **Assumption 13: SAM provides the authorization rules (or a reference to them), as defined by SOP for C's access to resources on S, in the scope of the access token (task D1a).**

The ACE framework recommends that the access token includes the audience (`aud`), i.e., the intended recipient (draft-ietf-ace-oauth-authz-06, p. 33). SAM and S must have a common understanding which audience identifies S (draft-ietf-ace-oauth-authz-06, p. 49). SAM thereby defines the claim destination (task **D1b**).

To define the holder of the claim, SAM must bind a verifier for C to the token (task **D1c**). In the `RawPublicKey` mode of the DTLS profile, the access token must be constructed in a way that S can associate the token with the client's raw public key. The RPK then acts as a verifier for the holder. In the `PresharedKey` mode, the symmetric key acts as a verifier for the holder.

In the `RawPublicKey` mode of the DTLS profile, SAM has obtained authorization rules for an entity with C's RPK if **Assumption 7** and **Assumption 9** hold. SAM then performs task **D1c** by specifying C's RPK in the token's `cnf` structure as the reference to the holder of the claim.

In the `PreSharedKey` mode, SAM generates the symmetric key for C and S (draft-ietf-ace-dtls-authorize-01, p. 10). As described above, C might have specified a key in the token request for access token updates; SAM must validate that this key actually is a valid verifier for C and S. SAM then specifies the PSK in the

cnf structure of the access token (draft-ietf-ace-dtls-authorize-01, p. 10), thereby generating the reference to the holder to perform task **D1c**.

SAM must endorse the information that it provides in the access token. The ACE framework demands that the access token is integrity-protected by SAM (draft-ietf-ace-oauth-authz-06, p. 12), but does not state how this is accomplished.

CWTs can be protected using a MAC or a cryptographic signature (RFC 8392, p. 9). If **Assumption 11** holds, SAM has obtained a verifier for S. If the verifier is a symmetric key, SAM can endorse the token by generating a MAC using this key. Since both SAM and S can generate the MAC, the solution must ensure that SAM cannot be tricked into accepting messages it formerly generated and that were reinserted in the communication by an attacker. This can be realized by using a distinct write key for each direction as it is done in TLS and DTLS. If SAM and S do not share a key, SAM can sign the token using its private key. In this case, the respective public key must be provided to S to enable it to validate the token (see also section B.8).

Unfortunately, the ACE framework and the DTLS profile do not specify that SAM must endorse the access token for S. The profile must mention that SAM must endorse the token using, e.g., a COSE MAC or a COSE signing object, to enable S to validate the authenticity of its contents.

We define **Assumption 14: SAM endorsed the content of the token for S using, e.g., a COSE MAC or signature (task D1)**.

The complete access token is depicted in figure B.3.

## B.5. Client Information

We described above how SAM performs authorization tasks for S and wraps the resulting information for S in the access token. But SAM also is a claim issuer for C: SAM provides C with the keying material and semantic information for

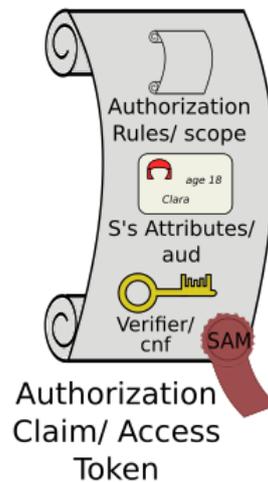


Figure B.3.: Access Token

the communication with S (task **An1**), which C requires for task **A6**. In the Pre-SharedKey mode, SAM generates the symmetric key for the communication between C and S and adds this to the client information. In the RPK mode, SAM provides C with the server's RPK (draft-ietf-ace-oauth-authz-06, p. 23).

SAM must perform task **D0** for C. From COP's perspective, C is authorized to receive the client information for S if it sent the access token request. SAM therefore must validate the integrity and authenticity of the token request. If **Assumption 9** holds, SAM validated that the token request was sent by C.

If the client information contains sensitive data, SAM must protect its confidentiality. To do so, SAM must use C's verifier in the security solution. If **Assumption 8** holds, SAM has obtained credentials for C that refer to the authorization rules that SOP defined for C. If the respective claims are still up to date, SAM can use the credentials to confidentiality-protect the access token response. For an object security solution, SAM can use C's credentials to encrypt the message. If TLS or DTLS are used, SAM has validated that it is communicating with C if **Assumption 9** holds. It can then enforce the authorization by sending the access token response over this TLS or DTLS connection.

We define **Assumption 15: SAM validates C's authorization to receive the client**

**information (task D0). If the client information contains sensitive data, SAM protects its confidentiality using C's credentials.**

To perform task **D1a**, SAM must define a claim statement. The ACE framework distinguishes COP and SOP; C and S therefore may have distinct principals. To achieve the **main authorization directive**, the authorization rules must be approved by the overseeing principal. SAM only obtains authorization rules from SOP (draft-ietf-ace-oauth-06, p. 50). It therefore cannot provide C with COP's authorization rules, but only help with the attribute validation (task **An1**) for task **A6**. The client information may also contain a scope, but it solely reflects SOP's authorization policies and is only informational. The claim therefore is an attribute claim with S as the holder. The claim statement must contain S's attributes.

SAM provides C with a verifier for S, either a PSK or S's RPK. SAM must validate that the S with the attributes that C provided in the token request has a certain verifier (task **An1**). To do so, SAM and C must have a common understanding with which server C wants to communicate (see also section B.4.2). SAM and C must know certain attributes of S that exclusively identify authorized servers, e.g., that S is the light switch that belongs to Alice and is located in her living room. Using these attributes, COP determines whether C is supposed to communicate with this server, and SAM provides the credentials and the access token that are necessary for the communication. **If SAM and C do not have a common understanding about S, SAM might provide credentials and access tokens for the wrong server, and C would not be able to detect the mistake.** The process where SAM provides C with the wrong credentials for S is depicted in figure 11.3.

We define **Assumption 16: SAM and C have a common understanding about S's attributes: if C asks SAM for a server with certain attributes, SAM provides the respective credentials and access token for this server.**

If **Assumption 11** holds, SAM has obtained credentials for the server that C attempts to communicate with as described in section B.4.2. The client information has the same format as the access token. SAM provides the server's verifier in a `rs_cnf` structure: it contains the server's RPK or the PSK that SAM generated (draft-ietf-ace-oauth-06, p. 19). If **Assumption 16** holds, SAM thereby performed task **D1c**.

The claim must refer to *S*'s attributes as the claim statement **D1a**. SAM should explicitly state *S*'s attributes in the client information. If SAM does not do that, the client information must be bound to the token request; the access token response is not self-descriptive in this case.

CoAP uses tokens to match responses to requests (RFC 7252, p. 34). As long as the integrity and authenticity of messages that are exchanged between SAM and *C* is protected, and the CoAP tokens are sufficiently unique, SAM can thereby bind the server's attributes from the access token request to the client information in the access token response.

While it is possible to link the client information to the requested server if *C* can link SAM's response to *C*'s request, the ACE framework and the DTLS profile should demand that the attribute that identifies *S* is given in the access token response.

SAM must specify the claim destination (task **D1b**). Otherwise *C* might be tricked into accepting an access token that was meant for a different *C*. SAM might specify the destination by explicitly stating an attribute that exclusively identifies the *C* that made the token request and is known to both SAM and *C*, e.g., the client ID. As an alternative, SAM can use keying material for *C* for this task. If **Assumption 9** holds, SAM obtained keying material for the *C* that sent the token request. If SAM set up a TLS or DTLS connection with *C* and *C* used its credentials in the handshake, SAM performs task **D1b** by encrypting the claim with its (D)TLS write encryption key.

If an object security solution is used, SAM might use *C*'s credentials to encrypt the key. If asymmetric keys are used, the token must first be encrypted and then signed. If SAM signs the client information before the encryption, no data destination verifiability can be achieved by encrypting the message: a malicious client that received client information could decrypt it and encrypt it for a different *C*. Also, if symmetric keys are used for the communication, different keys should be used for each direction, because otherwise the destination is not clear. In this case, additional measures are required to define the destination. The DTLS profile should define how data destination verifiability is achieved for the client information.

We define **Assumption 17: SAM clearly defined the intended destination of the client information (task D1b).**

To finish task **D1**, SAM must bind the claim information together and endorse it. SAM must use a communication security solution that provides integrity and message authentication. If a transport layer security solution is used between SAM and C, and **Assumption 9** holds, SAM has a TLS or DTLS connection with the correct C. SAM can then use its (D)TLS MAC key (or for AEAD cipher suites its encryption key) to generate a MAC of the client information.

For the object security solution, SAM uses its own private key to sign the client information, e.g., a COSE signature object, if an asymmetric solution is used. If SAM has obtained symmetric keying material for the communication with C, and **Assumption 8** holds, SAM can use it in the object security solution to generate a MAC, e.g., a COSE MAC object. Additional precautions are necessary to ensure that SAM cannot be tricked by client information that is reinserted into the communication.

The profile should explain that SAM must bind the pieces of information required by C together and protect its integrity and authenticity.

We define **Assumption 18: SAM bound the parts of the claim information together in the client information and endorsed them for C (task D1).**

## **B.6. Access Token Response**

SAM provides the client information to C in the access token response. SAM must send the access token response only to the C that is authorized by SOP and that sent the access token request. If the assumptions 7 to 10 hold, SAM performed the authorization tasks in behalf of S for the C that sent the token request. SAM must determine if the authorization rules for C are still up to date before sending the access token response. SAM might need to protect the confidentiality of the client information contained in the access token response to perform task **D0** and to enforce the authorization as described in section B.5. If **Assumption 15**

holds, SAM ensured that the client information in the access token response is transmitted only to the C that is authorized by SOP and that sent the access token request.

C must perform the delegation tasks **D2** to **D8**. For task **D2**, C must validate that the claim is up to date. One way to achieve this is that C knows when SAM generated the token and how long it is supposed to be valid (see also 10.2). The ACE framework specifies that the client information may contain an `expires_in` parameter (draft-ietf-ace-oauth-authz-06, p. 20). In OAuth 2.0, this parameter describes the lifetime of the access token in seconds from its generation (RFC 6749, p. 35). To comply with the **claim completeness fundamental**, the ACE framework should define a distinct validity period parameter for the client information or specify that it always has the same validity period as the access token. The `expires_in` parameter is optional. If it is missing, the client information may be valid indefinitely, and the solution is not secure.

The `expires_in` parameter is not sufficient to satisfy the **claim completeness fundamental** if the client cannot determine when the access token was generated. The access token may contain an `iat` parameter that specifies when the access token was issued (RFC 8392, p. 6; RFC 7519, p. 10), but C may not be able to validate this data since it is meant for S. Also, as described in section B.4.2, SAM must confidentiality-protect the access token if it contains sensitive data such as the shared key. C may therefore not be able to read the information in the token.

If the underlying security solution provides replay protection, C can thus ensure that the client information is fresh. To also ensure the validity, C could additionally set a timeout when sending the token request, and only wait for the token response from SAM until the timeout is reached. C can then determine that the client information was not sent by SAM longer ago than the timeout period. With the information in the `expires_in` parameter, C would then be able to determine how long the client information is valid if it continuously keeps the time. However, a constrained client might need to sleep most of the time and may turn off its clock while sleeping (see also section 3.1.4). The ACE framework should point out this problem and suggest a solution.

**If the client cannot validate that the client information is up to date, the claim completeness fundamental is not satisfied, and the client may use credentials**

for **S** that are no longer valid. The confidentiality and integrity of messages are then in jeopardy. There is no indication that the ACE framework is aware of this problem or willing to address it.

The ACE framework must point out that **C** must validate that the client information is still valid before sending a request to the server. The problem space must be described and a solution should be suggested.

We define **Assumption 19: C validated that it has obtained a claim concerning S's attributes and credentials from SAM that is currently valid (task D2).**

For task **D3**, **C** must validate that the parts of the client information are bound together and stem from the same SAM. If **Assumption 18** holds, SAM bound the pieces of claim information together and endorsed them. **C** validates that they stem from the same source by checking the MAC or signature. If SAM did not provide **S**'s attributes in the client information, **C** must additionally validate that the access token response is bound to the access token request. CoAP defines tokens to match responses to requests (RFC 7252, p. 12). If the tokens are sufficiently unique and the integrity and authenticity of the access token response is protected, **C** can thus validate that the client information refers to the server for which **C** requested the access token by comparing the CoAP tokens of the request and the response. In this case, the access token response is not self-descriptive. SAM should explicitly state **S**'s attributes in the client information.

The ACE framework or the DTLS profile must state that **C** must be able to validate that the client information refers to the server that **C** wants to communicate with. **C** must either check that **S**'s attribute which SAM provided in the client information matches the **S** that **C** requested, or **C** must validate that the CoAP token in the access token response matches the token in the access token request.

We define **Assumption 20: C validates that the required pieces of claim information about S are bound together and endorsed by SAM (task D3).**

For task **D4**, **C** must validate SAM's authorization to provide the client information. We described how **C** validates SAM's authorization to receive the access token request in section B.3. If **Assumption 3** and **Assumption 4** hold, **C** obtained authorization rules from COP that refer to a SAM with certain credentials. But to

satisfy the **rule completeness fundamental**, C must have authorization rules that are up to date, and might need to perform the tasks **A2** to **A5** again for the new rules. **COP must ascertain that the authorization rules for SAM on C are up to date, e.g., by performing regular updates.**

We define **Assumption 21: C obtained and validated the most recent authorization rules for SAM from COP (tasks A2 – A5).**

C must validate that SAM actually has the credentials that the authorization rules refer to. For a transport security solution, SAM has proven this if the credentials were used in the handshake and the access token response was protected in the TLS or DTLS connection with a MAC. For an object security solution, C can validate it if SAM provided a signature that was made with these credentials. If symmetric keys are used in the object security solution, different keying material should be used for every direction to ensure that a message created by C cannot be reinserted to trick C. If **Assumption 18** holds, SAM signed the client information.

The DTLS profile should specify that C must validate that the access token response actually stems from the SAM that is authorized by COP and to which C sent the access token request.

We define **Assumption 22: C validated that the access token response was sent by the SAM that is authorized by COP and that was the intended recipient of C's access token request (task A6).**

For task **A7**, C must validate that SAM is authorized to provide client information for the S that C wants to communicate with. The authorization information that C obtained from COP for SAM must contain this information.

We define **Assumption 23: C validated for the access token response that the context of the authorization rules for SAM matches the current context, e.g., that SAM is authorized to provide claims concerning S (task A7).**

How COP provides authorization information to C is out of scope for the ACE framework and the DTLS profile. We assume that C is controlled by a single overseeing principal and no additional effort is required for task **A8**. By performing the authorization tasks for SAM, C finishes task **D4**.

For task **D5**, C must validate that it is the destination of the claim. SAM might name an attribute of C, e.g., the client ID, to set the destination in the claim. If the integrity and authenticity of the claim is protected, C performs task **D5** by validating that the attribute actually refers to C. If C and SAM use transport layer security, C can validate that it is the destination of the claim by validating that SAM encrypted the message using its (D)TLS encryption key.

C cannot yet validate the holder of the claim. We will evaluate how C performs task **D6** in section B.10.

Obtaining client information from several claim issuers is not in scope for the DTLS profile. We assume that C does not need to evaluate claims from various sources (task **D8**).

## **B.7. Access Token Transmission**

C can transmit the access token to S by sending a `POST` request to the `/authz-info` resource on S. For the `PreSharedKey` mode, the access token can also be provided in the DTLS handshake. The information in the access token stems from SAM. If **Assumption 12** holds, SAM protected the access token so that it can only be accessed by an authorized S. C does not have to additionally confidentiality-protect it.

S cannot validate C's authorization to send the access token if C and S do not yet communicate securely. If the access token is transmitted to the `authz-info` resource on S, the ACE framework specifies that S must check the validity of the access token and respond with a `2.01 (created)` if the access token is valid. It takes more effort for the server to validate the access token than for C to provide it. The `/authz-info` resource therefore makes S particularly vulnerable to DoS attacks. Also, it is not clear how S manages the accepted access tokens. S must store access tokens as long as they are valid. Since S cannot authenticate C, an attacker that eavesdrops on the unprotected channel between C and S can reinsert the access token with very little effort. If the access token is appended to a list of valid access tokens, an attacker can thereby flood the memory or storage space on S. If

previous access tokens are replaced by newly received tokens, the attacker might trick *S* to use reinserted older tokens if no countermeasures are in place.

DTLS offers countermeasures against DoS attacks (see RFC 6347, p. 15). The risk for *S* is therefore mitigated if the access token is transmitted in the DTLS handshake. Also, *C* proves that it has the credentials that the access token refers to by completing the handshake. *S* therefore can discard the access token after the handshake if *C* is not authorized. A solution that transports the access token in the handshake is therefore more resilient against DoS attacks.

Where possible, *S* should validate *C*'s authorization to sent access tokens to reduce the risk of DoS attacks. *S* should not accept access token updates for *C* on an untrusted channel. We will analyze how *S* validates *C*'s authorization below.

## B.8. Access Token Validation

SAM assists *S* in checking *C*'s authorication and provides the required information in the access token. *S* must perform the delegation tasks **D2–D8** to finish the delegation. SAM and *S* must ensure that *S* always has the most recent authorization rules for *C* to comply with the **rule completeness fundamental**. If **Assumption 7** holds, SAM has obtained the most recent authorization rules from SOP, and if **Assumption 13** holds, SAM provided these authorization rules or a reference to them in the access token. *S* must check if it receives the most recent authorization rules. At this point, *S* has to face several problems: after a new token was issued, an attacker might resend the previous token to *S*. Also, access token updates might not reach *S*, e.g., because the client decides to not forward them to *S*, or because the connection between SAM and *C* is disturbed. During the communication with *C*, *S* must validate that the access token is still valid.

The ACE framework defines three methods how *S* can validate that the access token is still up to date:

1. SAM includes an expiration time in the access token,
2. *S* performs an introspection request or

### 3. SAM uses a sequence number in the access token

For the first approach, SAM includes an `exp` parameter that states the point in time when the access token will expire. In this case, S must have an internal clock that is synchronized with SAM's clock. SAM and S must regularly use time synchronization and therefore need connectivity to each other or a claim issuer that provides the current time, e.g. a Network Time Protocol (NTP, RFC 5905) server.

In the second approach, S delegates the validation that the access token is still valid by sending a token introspection request to SAM. SAM's response contains the parameter `active` which states whether the access token is valid or not. SAM and S must perform the delegation tasks for the token introspection. To achieve the **claim completeness fundamental**, S must validate that the response from SAM is up to date (task **D2**). There is no indication that the ACE framework attempts to solve this problem. One possible solution is that SAM includes a timestamp in the response. But then it would have been easier for SAM and S to use an expiration time in the access token. As an alternative, S might include a random value in the introspection request that SAM must reflect in the response. In this case, a timeout must be defined, and S must not accept a response after the timeout is reached. If C obtained SAM's address with an unauthorized resource request, S may have added a nonce to the response (see section 11.1.1) that can be used in the same way but without the burden of the additional introspection messages. If token introspection is used, the framework must point out that the token introspection messages must be replay-protected and that S must be able to determine if the token introspection response is up to date. Token introspection puts additional strain on servers and increases the risk of DoS attacks (see also section 11.2.1.2).

In the third approach, each access token has a sequence number. S must keep track of the sequence numbers and only accept tokens if they are within a certain range of the sequence number of the most recently received access token (draft-ietf-ace-oauth-authz-06, p. 32). In this case, access tokens are infinitely valid if an attacker manages to intercept new access tokens. E.g., C might decide to not relay new tokens to S if they reduce C's permissions. S therefore must discard access tokens after a certain time. S cannot participate in protecting SOP's security

objectives if it is not able to measure the passing of time in some way (see also section 10.2). The ACE framework must mention that in addition to checking the sequence numbers, S must start a timeout after receiving a token and discard the token after the timeout is reached.

We define **Assumption 24: S validated that it obtained an access token from SAM that is up to date.**

For task **D3**, S must validate that the claim information, i.e., the statement, destination and holder, stems from the same source. If **Assumption 13** holds, SAM provided the claim statement, i.e., the authorization rules, or a reference to it in the scope in the access token. The claim destination is specified with the `aud` parameter, and the holder of the claim is identified by the `cnf` parameter. S must validate that all this information is bound together by SAM. Unfortunately, the ACE framework does not clearly specify which pieces of information the access token must contain and that S must validate that they stem from the same SAM. It mentions that S must validate that access token is integrity-protected by SAM (draft-ietf-ace-oauth-authz-06, p. 12), but the DTLS profile does not specify how S is supposed to do that. If **Assumption 14** holds, SAM endorsed the content of the token for S. S must validate that SAM endorsed the whole content of the access token by checking its MAC or signature.

The DTLS profile must specify that S validates that the whole content of the access token was endorsed by the same SAM by checking the MAC or signature of the token.

We define **Assumption 25: S validated that all information in the access token is bound together by a MAC or signature**

To perform task **D4**, S must validate that the access token stems from an authorized (trusted) SAM. We will analyze how S performs the authorization tasks for SAM in section B.9.

For task **D5**, S must validate that it is intended destination of the claim. As stated in section B.4.2, the ACE framework demands that SAM and S have a common understanding which audience identifies S. If **Assumption 25** holds, S can validate that it is the intended destination of the access token by checking the `aud` parameter.

The ACE framework demands that S must reject requests to the `authz-info` resource that contain an access token with an audience that does not match S. S thereby performs task **D5**. The DTLS profile specifies that the same approach is used if the token is transmitted in the DTLS handshake (draft-ietf-ace-dtls-authorize-01, p. 12).

S cannot yet perform task **D6** when the access token is transmitted to it. We will analyze how S performs this task in section B.10.

Scenarios where S receives claims from multiple Authorization Managers for the same authorization process are not in scope for the ACE framework. Therefore, no additional work is necessary for task **D8**.

## **B.9. S validates SAM's authorization**

To perform task **D4**, S must validate that the access token stems from an authorized ("trusted") SAM. SOP must have defined authorization rules and their intended destination, context and holder (task **A1**) and provided them to S.

We define **Assumption 26: SOP approved of SAM being an authorization manager for S in this context (task A1)**.

SOP and S must ensure that S always has the most recent authorization rules for SAM (task **A2**). S must validate that the authorization information stem from the same SOP (task **A3**) and that it is authorized to provide them (task **A4**). Also, S must validate that it is meant to enforce the authorization rules (task **A5**). How these tasks are performed is not in scope for the ACE framework. The framework should mention that S must have been securely provisioned with the knowledge which entities are authorized to be authorization managers (e.g., during server registration).

We define **Assumption 27: S obtained authorization rules for SAM that are up to date (are continuously updated/revoked), were approved by an authorized SOP and are meant to be enforced by S (tasks A2–A5). The authorization rules refer to a SAM with certain credentials**.

S must validate that the endpoint that endorsed the access token actually is an authorized SAM. If **Assumption 14** holds, SAM endorsed the access token, e.g., by generating a MAC or signature. S can then perform task **A6** by checking the endorsement.

The ACE framework states that S must verify that the access token stems from the “right” SAM (draft-ietf-ace-oauth-Authz-06, p. 51) but does not state how S accomplishes this. Neither the ACE framework nor the DTLS profile clearly demand that S must authenticate the origin of the access token. **If S does not validate the origin of the token, anyone can define access tokens and the authorization solution is ineffective.**

We define **Assumption 28: S validated that the authorization rules from SOP refer to the SAM that endorsed the access token (task A6).**

For task **A7**, S must validate that the current context fits to the context of the authorization rules for SAM, e.g., that SAM is allowed to be authorization manager for a certain resource on S or a specific C. We assume that this task is not in scope for the ACE framework.

We define **Assumption 29: S validated that the current context matches the context of the authorization rules as defined by SOP, e.g., that SAM is meant to be authorization manager for a certain resource on S (task A7).**

We assume that scenarios with multiple overseeing principals are out of scope for the ACE framework and no additional effort is necessary to perform task **A8**.

## **B.10. Authorized Communication**

C and S must ensure that the communication data is protected as defined by their overseeing principals. Before sending a request to S, C must validate that S is authorized to receive it. COP must have defined authorization rules and their intended destination, context and holder.

We define **Assumption 30: COP approved authorization rules for S that are to be enforced by C in this context (task A1).**

C must obtain authorization rules for S that are up to date (task **A2**), must validate that the authorization rule stem from an authorized COP (tasks **A3** and **A4**), and are meant to be enforced by C (task **A5**). C must validate that the context in the authorization rules matches the current context (task **A7**). If there are multiple permissions, C must evaluate them and determine the correct authorization (**A8**).

We define Assumption 31: **C obtained and evaluated authorization rules for S from an authorized COP that are up to date and are meant to be enforced by C in this context. The authorization rules refer to certain attributes of S (tasks A2–A5 and A7–A8).**

It is not clear how COP and C perform these tasks. The ACE framework does not give any indication that COP's interests are considered in the authorization process (see also section 11.2.2).

To perform task **A6**, C must ascertain that it communicates with an S that actually is authorized. To satisfy the **correlation fundamental**, a continuous link between COP's authorization rules and the server that C communicates with must exist (see also figure 5.12). If **Assumption 16** holds, C and SAM have a common understanding about S's attributes. S is authorized if it can use the credentials that SAM provided to C in the client information, and if the authorization rules that were defined by COP refer to the S that is specified in the client information, i.e., has the same attributes. C must therefore validate that

1. the attributes of the server that the authorization rules refer to matches the attributes of the server that the client information refers to.
2. S uses the credentials that SAM specified in the client information to finish the DTLS handshake between C and S.
3. C always uses the subsequent DTLS connection when C sends or receives messages that require authorization in the communication with S.

The ACE framework specifies that C uses the credentials that SAM provided in the client information to establish a DTLS connection with S, but it does not point out that C must validate that the client information actually refer to the S that C wants to communicate with and that is authorized by COP. Without this

validation, the integrity and confidentiality of the transmitted messages may be breached.

We define Assumption 32: **C validated that the client information refers to an S that is authorized by COP and that the S that C communicates with is able to use the credentials that define the holder in the client information (task A6 and task D6).**

To enforce the authorization, C must only communicate with S as specified by COP in the authorization rules (task A9).

S obtains the authorization rules for C from SAM. If assumptions 24–29 hold, S validated that the authorization rules stem from SAM and are fresh, and that this SAM is authorized to provide the authorization rules. S thereby performed the authorization tasks A2 to A5 for C.

For task A6, S must validate that C actually is authorized. To do so, it uses the information in the access token: if C is able to use the credentials that are given in the `cnf` parameter of the access token, the authorization information in the access token's scope refers to C.

To perform task A7, S must validate that the current context matches the intended context. We assume that context constraints are out of scope for the ACE framework and the DTLS profile.

We define Assumption 33: **S validated that the current context matches the context intended by SOP for C's authorization (task A7).**

If various authorization rules were defined by SOP concerning C's authorization, S must evaluate them and determine the correct authorization. For this purpose, S must be aware what the scope means. The ACE framework demands that the access token is rejected if it contains an unknown scope (draft-ietf-ace-oauth-authz-06, p. 31). We assume that it is out of scope for the ACE framework and the DTLS profile how S evaluates the scope if it contains various elements. It would be useful if SAM performed this task for S by evaluating the elements in the scope and providing S with simplified authorization rules. A constrained S might have difficulties to store and interpret complex rules.

We define **Assumption 34: S evaluated the authorization rules and determined C's correct authorization (task A8).**

S must enforce the authorization as defined by SOP (task A9). To do so, it must only allow C to access resources, i.e., S must only send and receive resource representations, if this is covered by the authorization rules. The DTLS profile demands that S must check if incoming requests are authorized (draft-ietf-ace-dtls-authorize-01, p. 7).

## Appendix C.

### DCAF Alternative Architectures

The normal DCAF protocol flow is used for scenarios with two constrained or autonomous devices with distinct authorization managers. In some scenarios, actors can be combined in one device. Constrained devices with the same owner may share an AM (see section C.1). More powerful devices may be co-located with their AM, as we will show in sections C.2 and C.3.

#### C.1. Combined CAM and SAM

In scenarios where C and S belong to the same security domain, they have a single overseeing principal and may have a single authorization manager. CAM/SAM provides authentication and authorization information to both C and S. COP/SOP may decide that binary authorization is sufficient if their own devices communicate with each other. But there may still be reasons why clients are only allowed to access certain servers or a server may only exchange information with certain clients. Overseeing principals must not rely only on the server to protect the communication, because clients then may disclose confidential information or accept unreliable information from unauthorized servers.

Both C and S must have established a security association with CAM/SAM as described in section 12.1.2 before they can use DCAF. Since CAM and SAM are combined, the ticket request message and the ticket grant message are omitted. The resulting protocol flow for a combined CAM/SAM is depicted in figure C.1.

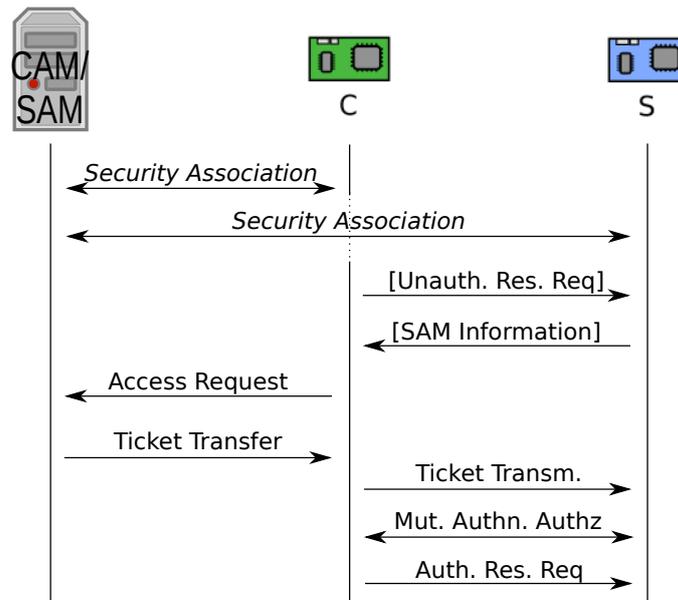


Figure C.1.: Combined Client Authorization Manager and Server Authorization Manager

In scenarios with a combined CAM/SAM, C may know that S has the same Authorization Manager and use its own CAM/SAM address in the access request. Unless C is certain that S uses validity option 1 to determine the freshness of an access token, C should send a request to the AM-info resource. S will then respond with a SAM information message. C uses the information in the SAM information message to generate the access request message as described in section 12.2.2. The communication between C and CAM/SAM must be integrity- and confidentiality-protected.

When CAM/SAM receives the access request, it must ensure that COP/SOP defined authorization rules for S concerning C. It then generates the ticket face as described in section 12.2.4.1. For the client information, CAM/SAM must use the authorization rules that COP/SOP defined for C concerning S. It creates the client information as specified in section 12.2.6. Since CAM and SAM are combined, CAM/SAM can set lifetime, sequence number, and validity information for C. The complete access ticket, comprising client information and ticket face, is sent to C in the ticket transfer message.

As in the basic protocol flow, C transfers the ticket face to S (see also section

12.2.7), and uses the client information to establish a security association with S as described in section 12.2.9.

## C.2. Combined Client and CAM

A client that resides on a less-constrained device may have an integrated CAM. COP provides the authorization rules to C/CAM, e.g., by directly controlling the device and making the authorization decisions. As in the basic protocol flow, SAM provides the required keying material for the communication with the server. This is the scenario that the ACE framework currently attempts to address (see also chapter 11).

For a combined C and CAM, the access request message and the ticket transfer message are not required. Figure C.2 shows the protocol flow for a combined C and CAM.

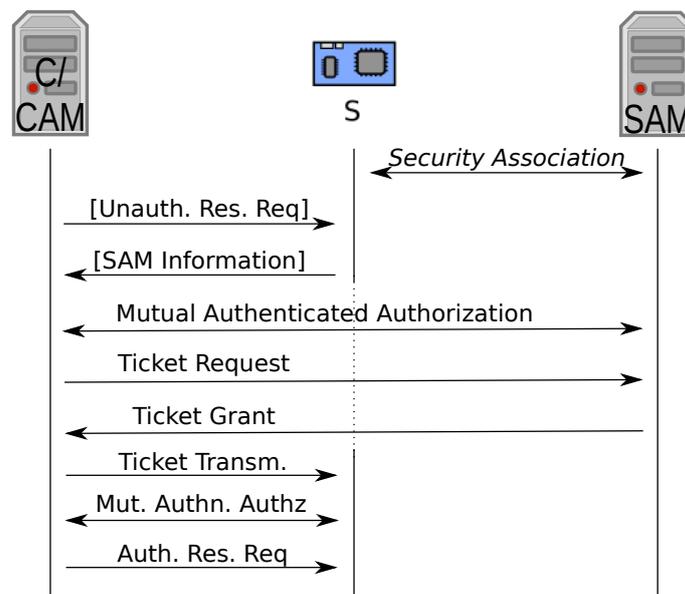


Figure C.2.: Combined Client Authorization Manager and Client

In this protocol flow, C/CAM creates the ticket request message after obtaining SAM's address, to request an access ticket from SAM. C/CAM must only send

the ticket request message if COP decided that SAM is authorized to provide keying material for C/CAM's communication with S. Also, COP must define which actions C is allowed to request.

The ticket request message contains the usual parameters as described in section 12.2.3. C/CAM must use the information from the SAM information message, i.e., SAM's address and all nonces, in the ticket request message. Additionally, C/CAM specifies the desired resources and actions for the access ticket. As specified in section 12.1.3, SAM must have obtained keying material and authorization rules about CAM from an authorized claim issuer.

As in the basic protocol flow, C/CAM and SAM must mutually authenticate each other and validate each other's authorization. SOP must have provided authorization rules for C/CAM to SAM. SAM must validate that the authorization rules refer to the C/CAM that SAM is communicating with. The confidentiality and integrity of the communication between C/CAM and SAM must be protected.

When C/CAM receives the ticket grant message, it must validate that the message stems from a SAM that is authorized to provide access tickets and keying material for the communication with S.

After that, C/CAM attempts to transmit the access ticket and establish a security association with S as described in sections 12.2.7 and 12.2.9.

C/CAM must consider COP's authorization rules before accessing a resource: it must only access resources that COP intends it to access.

### **C.3. Combined Server and SAM**

A server may be combined with the server authorization manager, e.g., if overseeing principals want to grant a constrained client access to a server that they control directly. If C knows that S and SAM are combined, it may use the Server-Initiated Ticket Request (see section 12.6)<sup>1</sup>, since this reduces the burden on con-

---

<sup>1</sup>SITR is prone to DoS attacks, but a less-constrained combined S and SAM is less vulnerable.

strained clients. Otherwise, C and CAM use the normal protocol flow as described in section 12.2.

If S/SAM receives an unauthorized resource request, it returns its own address as the SAM address in the SAM information message to inform C and CAM which endpoint they must contact for a ticket. SOP must have provided authorization information for CAM to S/SAM, e.g., by giving her confirmation before data is sent or received, or by configuring authorization rules on S/SAM. When S/SAM receives a ticket request message, it may send an empty ticket face together with the client information in the ticket grant message. The ticket transmission and the establishment of the security association is performed as described in section 12.2.

## Appendix D.

### DCAF Data Structure

DCAF messages transport the pieces of authentication and authorization information that the endpoints require for a secure authorization process. We use the CBOR web token format for message payloads. In the following, we describe how the parameters are used in DCAF. We provide CBOR labels for each parameter. Where those labels are not yet standardized, they are marked as TBD (to be defined).

- audience** The ACE framework distinguishes the `audience` and the `aud` parameter. To achieve consistency, we reuse these parameters. The client and CAM use the `audience` parameter (CBOR label 5, see draft-ietf-ace-oauth-Authz-35, p. 31) in the access request and ticket request messages to specify the attributes of the server. It may, e.g., contain the absolute URI of the server. CAM may also provide the server's attributes to C in the ticket transfer message with this parameter. SAM uses the `aud` parameter (CBOR label 3, see draft-ietf-ace-oauth-Authz-35, p. 10 and RFC 8392, p. 7) in the ticket face to specify the intended destination.
- cnf** (confirm, CBOR label 8, RFC 8747, p. 11) CAM may use this parameter to specify C's public key in the ticket request message. In the ticket face, SAM specifies the keying material or key derivation information for the communication with C in the `cnf` parameter. SAM also uses this parameter to specify the symmetric keying material for the communication with S in the client information.

---

<b>cnonce</b>	(CBOR label 39 (TBD)), draft-ietf-ace-oauth-authz-35, p. 18) The nonce that the server may specify in the SAM information message. The nonce then must also be present in the subsequent messages of the protocol flow. The <code>cnonce</code> is used by the server to validate the freshness of the access ticket for validity options 2 and 3 (see section 12.1.7).
<b>cscope</b>	(client scope, CBOR label 21 (TBD)) The authorization information representing COP's authorization rules that CAM provides to C concerning S. CAM may provide an AIF object in the <code>cscope</code> parameter that comprises the actions that C is allowed to perform on resources on S.
<b>cti</b>	(CBOR label 7 RFC 8392, p. 6) The sequence number of the ticket face or client information.
<b>dseq</b>	(deprecated ticket sequence number, CBOR label 19 (TBD)) SAM may specify the sequence number of the ticket that is to be replaced in the ticket face. It enables S to find the ticket that is to be replaced. C does not require a deprecated ticket sequence number, but, if necessary, uses the server's attributes to find the ticket that is to be replaced.
<b>iat</b>	(issued at, CBOR label 6 RFC 8392, p. 6) CAM uses this parameter to specify an issuing time in the ticket request to SAM. Also, SAM specifies the issuing time in the access ticket with this parameter. SAM or CAM may use this parameter to provide the issuing time of the client information.
<b>expires_in</b>	(lifetime, CBOR label 17 (TBD)) The lifetime of the access ticket or the client information. It contains the remaining lifetime in seconds. CAM may also use this parameter to specify the remaining lifetime of C's public key. In the client information, CAM or SAM may instead provide the remaining lifetime using the CoAP Max-Age option.

<b>rs_cnf</b>	(CBOR label 41 (TBD), draft-ietf-ace-oauth-params-13, p. 9) SAM may specify the server's RPK in the client information with this parameter.
<b>sam</b>	(CBOR label 1 (TBD), draft-ietf-ace-oauth-Authz-35, p. 31) The absolute URI of the server authorization manager. S may specify it in the SAM information message to inform C at which URI it should request the access ticket. C may specify the URI in the access request message to CAM.
<b>scope</b>	(CBOR label 9 (TBD) draft-ietf-ace-oauth-Authz-35, pp. 56–57) The authorization information representing SOP's authorization rules that SAM provides to S about C. DCAF recommends the use of AIF to represent authorization information. In this case, the scope contains an AIF object which describes which actions C is allowed to perform on which resources. Additionally, C and CAM may use the scope structure to specify in the access request message and the ticket request message which resources C wants to access.
<b>sub</b>	(CBOR label 2, RFC 8392, p. 6) The description of the client's characteristics. It is specified by CAM in the ticket request message to enable SAM to provide more fine-grained authorization rules to S.
<b>ticket_face</b>	(CBOR label 1 (TBD), draft-ietf-ace-oauth-Authz-35, p. 18) The ticket face represents the server's part of the access ticket. It must contain a <code>scope</code> structure (which may be empty), the lifetime parameter <code>expires_in</code> , and the ticket's sequence number <code>seq</code> . It may additionally contain the keying material for C in the <code>cnf</code> parameter, the intended recipient of the ticket <code>aud</code> and the replaced ticket number <code>dseq</code> . If the ticket face is protected using COSE, it contains a COSE object.

# Bibliography

- M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. In *1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136. IEEE Computer Society, May 1994. DOI 10.1109/RISP.1994.296587.
- M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A Calculus for Access Control in Distributed Systems. *ACM Trans. Program. Lang. Syst.*, 15(4):706–734, Sept. 1993. ISSN 0164-0925. DOI 10.1145/155183.155225.
- N. J. Al Fardan and K. G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013. DOI 10.1109/SP.2013.42.
- Amazon. Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region, 2017. URL <https://aws.amazon.com/message/41926/>. Online; accessed 2020-09-16.
- G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella. Energy Conservation in Wireless Sensor Networks: A Survey. *Ad Hoc Networks*, 7(3):537–568, 2008. DOI 10.1016/j.adhoc.2008.06.003.
- R. Anderson. Why Information Security is Hard – An Economic Perspective. In *Seventeenth Annual Computer Security Applications Conference*, pages 358–365, New Orleans, LA, USA, 2001. DOI 10.1109/ACSAC.2001.991552.
- R. Anderson. *Security Engineering*. John Wiley & Sons, 2nd edition, 2008.
- R. Anderson and S. Fuloria. Who Controls the off Switch? In *2010 First IEEE International Conference on Smart Grid Communications*, pages 96–101, Oct 2010. DOI 10.1109/SMARTGRID.2010.5622026.

- R. Anderson and T. Moore. The Economics of Information Security. 314(5799): 610–613, 2006. DOI 10.1126/science.1130992.
- R. Anderson and R. Needham. Programming Satan’s Computer. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, pages 426–440. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-49435-5. DOI 10.1007/BFb0015258.
- M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, Aug 2017. USENIX Association. ISBN 978-1-931971-40-9.
- I. Asimov. Runaround. *Astounding Science Fiction*, 29(1):94–103, 1942.
- D. Basin, C. Cremers, and C. Meadows. *Model Checking Security Protocols*, pages 727–762. Springer International Publishing, Cham, 2018. DOI 10.1007/978-3-319-10575-8\_22.
- M. Berger-de Leon, T. Reinbacher, and D. Wee. *McKinsey*, Mar 2018. URL <https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-iot-as-a-growth-driver>. Online; accessed 2020-09-17.
- O. Bergmann, S. Gerdes, S. Schaefer, F. Junge, and C. Bormann. Secure Bootstrapping of Nodes in a CoAP Network. In *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 220–225, 2012a. DOI 10.1109/WCNCW.2012.6215494.
- O. Bergmann, K. T. Hillmann, and S. Gerdes. A CoAP-Gateway for Smart Homes. In *2012 International Conference on Computing, Networking and Communications (ICNC)*, pages 446–450, Jan 2012b. DOI 10.1109/ICCNC.2012.6167461.
- T. Berners-Lee. The World Wide Web: Past, Present, Future, 1996. URL <http://www.w3.org/People/Berners-Lee/1996/ppf.html>. Online; accessed 2020-09-16.

- T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. STD 66/RFC 3986, January 2005. DOI 10.17487/RFC3986. Internet Request for Comments.
- N. Bilton. Nest Thermostat Glitch Leaves Users in the Cold. *The New York Times*, Jan 2016. URL <http://www.nytimes.com/2016/01/14/fashion/nest-thermostat-glitch-battery-dies-software-freeze.html>. Online; accessed 2020-09-17.
- M. Bing. CoAP Attacks In The Wild. *Netscout*, Jan 2019. URL <https://www.netscout.com/blog/asert/coap-attacks-wild>. Online; accessed 2020-09-17.
- M. Bishop and M. Dilger. Checking for Race Conditions in File Accesses. *Computing Systems*, 9(2):131–152, Mar 1996.
- Bitdefender. Ring Video Doorbell Pro Under the Scope. Nov 2019. URL <https://labs.bitdefender.com/2019/11/ring-video-doorbell-pro-under-the-scope/>. Online; accessed 2020-09-28.
- S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492, May 2006. DOI 10.17487/RFC4492. Internet Request for Comments.
- M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The Role of Trust Management in Distributed Systems Security. In J. Vitek and C. Jensen, editors, *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 185–210. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66130-6. DOI 10.1007/3-540-48749-2\_8.
- C. Bormann. SWORN: Secure Wake on Radio Nudging. draft-bormann-t2trg-sworn-03, October 2019. Internet-Draft (Work in Progress).
- C. Bormann. An Authorization Information Format (AIF) for ACE. draft-ietf-ace-aif-00, July 2020. Internet-Draft (Work in Progress).
- C. Bormann and P. Hoffman. Concise Binary Object Representation (CBOR). RFC 7049, October 2013. DOI 10.17487/RFC7049. Internet Request for Comments.

- C. Bormann and Z. Shelby. Block-Wise Transfers in the Constrained Application Protocol (CoAP). RFC 7959, August 2016. DOI 10.17487/RFC7959. Internet Request for Comments.
- C. Bormann, A. P. Castellani, and Z. Shelby. CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Computing*, 16(2):62–67, March 2012. ISSN 1089-7801. DOI 10.1109/MIC.2012.29.
- C. Bormann, M. Ersue, and A. Keranen. Terminology for Constrained-Node Networks. RFC 7228, May 2014a. DOI 10.17487/RFC7228. Internet Request for Comments.
- C. Bormann, S. Gerdes, and O. Bergmann. Clearing off the Cloud over the Internet of Things. W3C/IAB Workshop on Strengthening the Internet Against Pervasive Monitoring (STRINT). Position Paper, 2014b. URL <https://www.w3.org/2014/strint/papers/28.pdf>. Online; accessed 2020-09-23.
- C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor. CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets. RFC 8323, February 2018. DOI 10.17487/RFC8323. Internet Request for Comments.
- C. Bormann, M. Ersue, A. Keranen, and C. Gomez. Terminology for Constrained-Node Networks. draft-bormann-lwig-7228bis-06, Marc 2020. Internet-Draft (Work in Progress).
- R. Bouboushian. Avant-Garde Sex Toys Caught in Data Dilemma. *Courthouse News Service*, Sep 2016. URL <https://www.courthousenews.com/avant-garde-sex-toys-caught-in-data-dilemma/>. Online; accessed 2020-09-17.
- M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *Digital Equipment Corporation Systems Research Center*, Feb 1989. Report 39.
- M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, Feb 1990. ISSN 0734-2071. DOI 10.1145/77648.77649.

- CA/Browser Forum. Baseline Requirements Documents (SSL/TLS Server Certificates). URL <https://cabforum.org/baseline-requirements-documents/>. Online; accessed 2020-09-17.
- CA/Browser Forum. Baseline Requirements Certificate Policy for the Issuance and Management of Publicly-Trusted Certificates. Version 1.6.8, Mar 2020. URL <https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.6.8.pdf>. Online; accessed 2020-09-17.
- C. Cadwalladr and E. Graham-Harrison. Revealed: 50 Million Facebook Profiles Harvested for Cambridge Analytica in Major Data Breach . *The Guardian*, March 2018. URL <https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election>. Online; accessed 2020-09-17.
- A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP). RFC 8075, February 2017. DOI 10.17487/RFC8075. Internet Request for Comments.
- V. Cerf, P. S. Ryan, and M. Senges. Internet Governance is Our Shared Responsibility. *A Journal of Law and Policy for the Information Society*, 2013.
- E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague. OAuth Demystified for Mobile Application Developers. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14*, pages 892–903, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 978-1-4503-2957-6. DOI 10.1145/2660267.2660323.
- E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
- W. Colitti, K. Steenhaut, and N. D. Caro. Integrating Wireless Sensor Networks with the Web. In *Proceedings of the 2011 Workshop on Extending the Internet to Low power and Lossy Networks*, pages 1–5, 2011.
- Comodo. Update 31-mar-2011, Nov 2011. URL <https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html>. Online; accessed 2020-09-17.

- D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008. DOI 10.17487/RFC5280. Internet Request for Comments.
- Crescent, 2017. URL <http://crescentfrequency.com/products/quartz-crystals>. Online; accessed 2020-09-17.
- S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. STD 86/RFC 8200, July 2017. DOI 10.17487/RFC8200. Internet Request for Comments.
- D. E. Denning and G. M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536, Aug 1981. ISSN 0001-0782. DOI 10.1145/358722.358740.
- W. Denniss, J. Bradley, M. Jones, and H. Tschofenig. OAuth 2.0 Device Authorization Grant. RFC 8628, August 2019. DOI 10.17487/RFC8628. Internet Request for Comments.
- T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. DOI 10.17487/RFC5246. Internet Request for Comments.
- W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976. ISSN 0018-9448. DOI 10.1109/TIT.1976.1055638.
- D. Dolev and A. C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, Mar 1983. DOI 10.1109/TIT.1983.1056650.
- V. Dukhovni. Opportunistic Security: Some Protection Most of the Time. RFC 7435, December 2014. DOI 10.17487/RFC7435. Internet Request for Comments.
- P. Dutta and A. Dunkels. Operating systems and network protocols for wireless sensor networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):68–84, Jan 2012. ISSN 1364-503X, 1471-2962. DOI 10.1098/rsta.2011.0330.

- P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a Wireless Sensor Network Platform for Detecting Rare, Random and Ephemeral Events. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, IPSN '05*, pages 497–502, Piscataway, NJ, USA, 2005. IEEE Press. DOI 10.1109/IPSIN.2005.1440983.
- C. Eckert. *IT-Sicherheit*. Oldenbourg Verlag, 8th edition, 2013.
- ECM Electronics 2017. URL [http://www.ecmelectronics.co.uk/quartz\\_crystals.html](http://www.ecmelectronics.co.uk/quartz_crystals.html). Online; accessed 2020-09-17.
- C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, September 1999. DOI 10.17487/RFC2693. Internet Request for Comments.
- J. Ellsmoor. Smart Cities: The Future Of Urban Development. *Forbes*, May 2019. URL <https://www.forbes.com/sites/jamesellsmoor/2019/05/19/smart-cities-the-future-of-urban-development/>. Online; accessed 2020-09-27.
- P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279, December 2005. DOI 10.17487/RFC4279. Internet Request for Comments.
- S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization. RFC 3281, April 2002. DOI 10.17487/RFC3281. Internet Request for Comments.
- S. Farrell and H. Tschofenig. Pervasive Monitoring Is an Attack. RFC 7258, May 2014. DOI 10.17487/RFC7258. Internet Request for Comments.
- FDA. Cybersecurity Vulnerabilities Identified in St. Jude Medical’s Implantable Cardiac Devices and Merlin@home Transmitter: FDA Safety Communication. Jan 2017. URL <https://www.fda.gov/medical-devices/safety-communications/cybersecurity-vulnerabilities-identified-st-jude-medicals-implantable-cardiac-devices-and-merlinhome>. Online; accessed 2020-09-28.
- N. Ferguson, B. Schneier, and T. Kohno. *Cryptography Engineering*. Wiley Publishing, Inc., 2010.

- D. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-based access control*. Artech House, 2003.
- R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Dissertation, University of California, Irvine, 2000.
- R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002. DOI 10.1145/514183.514185.
- L. Francis, G. Hancke, K. Mayes, and K. Markantonakis. Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones. In O. Yalcin and S. Berna, editors, *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *Lecture Notes in Computer Science*, pages 35–49. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16821-5. DOI 10.1007/978-3-642-16822-2\_4.
- FTC 2015. Internet of Things. Privacy and Security in a Connected World, 2015.
- E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
- O. Garcia-Morchon, S. Kumar, and M. Sethi. Internet of Things (IoT) Security: State of the Art and Challenges. RFC 8576, April 2019. DOI 10.17487/RFC8576. Internet Request for Comments.
- Gartner. Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020. Aug 2019. URL <https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io>. Online; accessed 2020-09-28.
- S. Gerdes. Managing the Authorization to Authorize in the Lifecycle of a Constrained Device. draft-gerdes-ace-a2a-01, September 2015a. Internet-Draft (Work in Progress).
- S. Gerdes. Server-Initiated Ticket Request. draft-gerdes-ace-dcaf-sitr-00, October 2015b. Internet-Draft (Work in Progress).
- S. Gerdes and O. Bergmann. Security Requirements for Managing Smart Objects in Home Automation. In A. Timm-Giel, J. Strassner, R. Agüero, S. Sargento,

- and K. Pentikousis, editors, *Mobile Networks and Management: 4th International Conference, MONAMI 2012, Hamburg, Germany, September 24-26, 2012, Revised Selected Papers*, pages 231–243. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-37935-2. DOI 10.1007/978-3-642-37935-2\_18.
- S. Gerdes, O. Bergmann, and C. Bormann. Delegated CoAP Authentication and Authorization Framework (DCAF). draft-gerdes-core-dcaf-authorize-00, July 2013. Internet-Draft (Work in Progress).
- S. Gerdes, O. Bergmann, and C. Bormann. Delegated Authenticated Authorization for Constrained Environments. In *IEEE 22nd International Conference on Network Protocols (ICNP)*, pages 654–659, Oct 2014. DOI 10.1109/ICNP.2014.104.
- S. Gerdes, O. Bergmann, and C. Bormann. Delegated CoAP Authentication and Authorization Framework (DCAF). draft-gerdes-ace-dcaf-authorize-04, October 2015a. Internet-Draft (Work in Progress).
- S. Gerdes, O. Bergmann, and C. Bormann. Examples for Using DCAF with Less Constrained Devices. draft-gerdes-ace-dcaf-examples-00, July 2015b. Internet-Draft (Work in Progress).
- S. Gerdes, C. Bormann, and O. Bergmann. Keeping Users Empowered in a Cloudy Internet of Things. In R. Ko and K.-K. R. Choo, editors, *The Cloud Security Ecosystem*, pages 231–247. Syngress, Boston, 2015c. ISBN 978-0-12-801595-7. DOI 10.1016/B978-0-12-801595-7.00011-2.
- S. Gerdes, R. Hummen, and O. Bergmann. Autorisierungsmanagement für das Internet of Things. In P. Schartner, K. Lemke-Rust, and M. Ullmann, editors, *D-A-CH Security 2015*, pages 397–408. Syssec, 2015d.
- S. Gerdes, O. Bergmann, C. Bormann, G. Selander, and L. Seitz. Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE). draft-ietf-ace-dtls-authorize-01, July 2017. Internet-Draft (Work in Progress).
- S. Gerdes, O. Bergmann, and C. Bormann. Avoiding Gaps in Authorization Solutions for the Internet of Things. *Proceedings of the NDSS Work-*

- shop on Decentralized IoT Security and Standards (DISS) 2018*, 2018a. DOI 10.14722/diss.2018.23006.
- S. Gerdes, O. Bergmann, C. Bormann, G. Selander, and L. Seitz. Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE). draft-ietf-ace-dtls-authorize-04, Sep 2018b. Internet-Draft (Work in Progress).
- S. Gerdes, L. Seitz, G. Selander, and C. Bormann. An architecture for authorization in constrained environments. draft-ietf-ace-actors-07, October 2018c. Internet-Draft (Work in Progress).
- S. Gerdes, O. Bergmann, C. Bormann, G. Selander, and L. Seitz. Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE). draft-ietf-ace-dtls-authorize-13, September 2020. Internet-Draft (Work in Progress).
- D. Goodin. When Coffee Makers are Demanding a Ransom, You Know IoT is Screwed. *Ars Technica*, Sept 2020. URL <https://arstechnica.com/information-technology/2020/09/how-a-hacker-turned-a-250-coffee-maker-into-ransom-machine/>. Online; accessed 2020-09-28.
- M. Goodwin. Improving Revocation: OCSP Must-Staple and Short-lived Certificates. *Mozilla Security Blog*, Nov 2015. URL <https://blog.mozilla.org/security/2015/11/23/improving-revocation-ocsp-must-staple-and-short-lived-certificates/>. Online; accessed 2020-09-17.
- G. S. Graham and P. J. Denning. Protection: Principles and Practice. In *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference, AFIPS '72 (Spring)*, pages 417–429, New York, NY, USA, 1972. Association for Computing Machinery. DOI 10.1145/1478873.1478928.
- A. Greenberg. Hackers Remotely Kill a Jeep on the Highway—With Me in It. *Wired*, Jul 2015. URL <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>. Online; accessed 2020-09-17.

- J. A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile. IEEE 802.15.4: a Developing Standard for Low-Power Low-Cost Wireless Personal Area Networks. *IEEE Network*, 15(5):12–19, Sept 2001. DOI 10.1109/65.953229.
- O. Hahm, E. Baccelli, H. Petersen, M. Wählisch, and T. C. Schmidt. Demonstration Abstract: Simply RIOT — Teaching and Experimental Research in the Internet of Things. In *IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, pages 329–330, 2014. DOI 10.1109/IPSN.2014.6846787.
- E. Hammer. OAuth 2.0 and the Road to Hell, Jul 2012. URL <https://archive.is/UYOsY>. Online (archived); accessed at 2020-09-17.
- D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012. DOI 10.17487/RFC6749. Internet Request for Comments.
- K. Hartke. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641, September 2015. DOI 10.17487/RFC7641. Internet Request for Comments.
- T. Hartwich. Entwurf und Implementierung eines Systems zur gesicherten Übertragung von Daten für die automatisierte Autorisierung von Kommunikationspartnern in Constrained Environments. Diploma Thesis, Universität Bremen, 2015.
- J. Heuer, J. Hund, and O. Pfaff. Toward the Web of Things: Applying Web Technologies to the Physical World. *Computer*, 48(5):34–42, May 2015. ISSN 0018-9162. DOI 10.1109/MC.2015.152.
- S. Hilton. Dyn Analysis Summary Of Friday October 21 Attack. *Dyn Company News*, Oct 2016. URL <https://web.archive.org/web/20161106103929/http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/>. Online (archived); accessed 2020-09-18.
- R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL Landscape: A Thorough Analysis of the X.509 PKI Using Active and Passive Measurements. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement*, IMC

- '11, pages 427–444, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310130. DOI 10.1145/2068816.2068856.
- HP. HP Designjet Printers - Certificate of Volatility, 2020. URL <https://support.hp.com/us-en/document/c00032325>. Online; accessed 2020-09-16.
- J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, September 2011. DOI 10.17487/RFC6282. Internet Request for Comments.
- R. Hummen, H. Shafagh, S. Raza, T. Voigt, and K. Wehrle. Delegation-based Authentication and Authorization for the IP-based Internet of Things. 06 2014. DOI 10.13140/2.1.4506.0480.
- IEEE Std. 802.15.4-2006. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), 2006.
- IoT Analytics. State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating, August 2018. URL <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>. Online; accessed 2020-09-17.
- ISO/IEC 24760-1. Information Technology — Security Techniques — A Framework for Identity Management, 2011.
- ISO/IEC 80000-13. Quantities and Units, 2008.
- T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social Phishing. *Communications of the ACM*, 50(10):94–100, Oct 2007. ISSN 0001-0782. DOI 10.1145/1290958.1290968.
- S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Transaction on Database Systems*, 26(2):214–260, June 2001. ISSN 0362-5915. DOI 10.1145/383891.383894.
- M. Jones, J. Bradley, and N. Sakimura. JSON Web Token (JWT). RFC 7519, May 2015. DOI 10.17487/RFC7519. Internet Request for Comments.

- M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenig. CBOR Web Token (CWT). RFC 8392, May 2018. DOI 10.17487/RFC8392. Internet Request for Comments.
- M. Jones, L. Seitz, G. Selander, S. Erdtman, and H. Tschofenig. Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs). RFC 8747, March 2020. DOI 10.17487/RFC8747. Internet Request for Comments.
- J. Jonsson. On the Security of CTR + CBC-MAC. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography*, pages 76–93. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-36492-4.
- Juniper. Statements of Volatility for Juniper Network Devices, 2020. URL [https://www.juniper.net/documentation/en\\_US/release-independent/junos/topics/reference/compliance/statement-of-volatility.html](https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/compliance/statement-of-volatility.html). Online; accessed 2020-09-16.
- J. Kasten, E. Wustrow, and J. Halderman. CAge: Taming Certificate Authorities by Inferring Restricted Scopes. In A.-R. Sadeghi, editor, *Financial Cryptography and Data Security*, volume 7859 of *Lecture Notes in Computer Science*, pages 329–337. Springer Berlin Heidelberg, 2013. DOI 10.1007/978-3-642-39884-1\_28.
- A. Kerckhoffs. La Cryptographie Militaire. *Journal des Sciences Militaires*, 9:5–38, February 1883.
- N. Koblitz and A. Menezes. Critical Perspectives on Provable Security: Fifteen Years of "Another Look" Papers. *Advances in Mathematics of Communications*, 13(4):517–559, 2019. ISSN 1930-5346. DOI 10.3934/amc.2019034.
- H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010. DOI 10.17487/RFC5869. Internet Request for Comments.
- H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997. DOI 10.17487/RFC2104. Internet Request for Comments.
- B. Krebs. FBI: Smart Meter Hacks Likely to Spread. *Krebs on Security*, Apr 2012. URL <https://krebsonsecurity.com/2012/04/fbi-smart-meter-hacks-likely-to-spread/>. Online; accessed 2020-09-17.

- V. Kumar. What the Future of Industry 4.0 Holds for Manufacturing? *IndustryWired*, Jun 2020. URL <https://industrywired.com/what-the-future-of-industry-4-0-holds-for-manufacturing/>. Online; accessed 2020-09-16.
- N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919, August 2007. DOI 10.17487/RFC4919. Internet Request for Comments.
- B. W. Lampson. Protection. *Proc. Fifth Princeton Symposium on Information Sciences and Systems*, pages 437–443, Mar. 1971. Reprinted in *Operating Systems Review*, 8, 1, January 1974, pp. 18–24.
- B. W. Lampson. Practical principles for computer security. In *NATO Security through Science Series - D: Information and Communication Security, Software System Reliability and Security*. IOS Press, January 2007. URL <https://www.microsoft.com/en-us/research/publication/practical-principles-for-computer-security/>. Proceedings of the 2006 Marktoberdorf Summer school.
- A. Langley. Further Improving Digital Certificate Security. *Google Security Blog*, Dec 2013. URL <https://security.googleblog.com/2013/12/further-improving-digital-certificate.html>. Online; accessed 2020-09-18.
- A. Langley. Maintaining Digital Certificate Security. *Google Security Blog*, Mar 2015. URL <https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html>. Online; accessed 2020-09-18.
- S. Larson. A Smart Fish Tank Left a Casino Vulnerable to Hackers. *CNNtech*, Jul 2017. URL <http://money.cnn.com/2017/07/19/technology/fish-tank-hack-darktrace/index.html>. Online; accessed 2020-09-18.
- C. Laughman, K. Lee, R. Cox, S. Shaw, S. Leeb, L. Norford, and P. Armstrong. Power Signature Analysis. *IEEE Power and Energy Magazine*, 1(2):56–63, Mar 2003. ISSN 1540-7977. DOI 10.1109/MPAE.2003.1192027.
- B. Laurie. Certificate Transparency. *Communications of the ACM*, 57(10):40–46, Sept. 2014. ISSN 0001-0782. DOI 10.1145/2659897.

- B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, June 2013. DOI 10.17487/RFC6962. Internet Request for Comments.
- S. B. Leeb, S. R. Shaw, and J. L. Kirtley. Transient Event Detection in Spectral Envelope Estimates for Nonintrusive Load Monitoring. *IEEE Transactions on Power Delivery*, 10(3):1200–1210, Jul 1995. ISSN 0885-8977. DOI 10.1109/61.400897.
- N. Li, B. N. Grosf, and J. Feigenbaum. Delegation Logic: A Logic-based Approach to Distributed Authorization. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):128–171, Feb. 2003. ISSN 1094-9224. DOI 10.1145/605434.605438.
- G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer Berlin Heidelberg, 1996. ISBN 978-3-540-49874-2. DOI 10.1007/3-540-61042-1\_43.
- M. Machulak and J. Richer. User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization. Technical report, 2018. URL <https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html>. Online; accessed 2020-09-19.
- A. Malhotra, M. Van Gundy, M. Varia, H. Kennedy, J. Gardner, and S. Goldberg. The Security of NTP’s Datagram Protocol. In *Financial Cryptography and Data Security*, pages 405–423. Springer International Publishing, 2017. ISBN 978-3-319-70972-7. DOI 10.1007/978-3-319-70972-7\_23.
- C. B. Margi, M. A. Simplicio Jr., M. Näslund, B. T. de Oliveira, P. S. L. M. Barreto, R. Gold, G. T. de Sousa, and T. C. M. B. Carvalho. Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds. In *2010 Proceedings of 19th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2010. DOI 10.1109/ICCCN.2010.5560028.
- B. Marr. What Everyone Must Know About Industry 4.0. *Forbes*, Jun 2016. URL <https://www.forbes.com/sites/bernardmarr/2016/06/20/what-everyone-must-know-about-industry-4-0/>. Online; accessed 2020-09-16.
- P. Marrapese. Security Cameras Vulnerable to Hijacking. 2020. URL <https://hacked.camera/>. Online; accessed 2020-09-28.

- L. Matsakis. Microsoft's Supreme Court Case has Big Implications for Data. *Wired*, Feb 2018. URL <https://www.wired.com/story/us-vs-microsoft-supreme-court-case-data/>. Online, accessed 2020-09-18.
- P. McDaniel and S. McLaughlin. Security and Privacy Challenges in the Smart Grid. *IEEE Security and Privacy*, 7(3):75–77, May 2009. ISSN 1540-7993. DOI 10.1109/MSP.2009.76.
- D. McGrew. An Interface and Algorithms for Authenticated Encryption. RFC 5116, January 2008. DOI 10.17487/RFC5116. Internet Request for Comments.
- C. A. Meadows. *Formal Verification of Cryptographic Protocols: A Survey*, pages 133–150. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-49236-8. DOI 10.1007/BFb0000430.
- A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC press, 1996.
- S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos Authentication and Authorization system. In *Project Athena Technical Plan*, 1987.
- D. Mills, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, June 2010. DOI 10.17487/RFC5905. Internet Request for Comments.
- B. Möller, T. Duong, and K. Kotowicz. This POODLE Bites: Exploiting the SSL 3.0 Fallback. *Security Advisory*, 2014. URL <https://www.openssl.org/~bodo/ssl-poodle.pdf>. Online; accessed 2020-09-24.
- G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, September 2007. DOI 10.17487/RFC4944. Internet Request for Comments.
- J. H. Moor. What is Computer Ethics? *Metaphilosophy*, 16(4):266–275, 1985. DOI 10.1111/j.1467-9973.1985.tb00173.x.
- G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, Apr. 1965.

- B. Moran, H. Tschofenig, and H. Birkholz. An Information Model for Firmware Updates in IoT Devices. draft-ietf-suit-information-model-07, June 2020. Internet-Draft (Work in Progress).
- R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, Dec. 1978. ISSN 0001-0782. DOI 10.1145/359657.359659.
- C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120, July 2005. DOI 10.17487/RFC4120. Internet Request for Comments.
- J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez. IPv6 over BLUETOOTH(R) Low Energy. RFC 7668, October 2015. DOI 10.17487/RFC7668. Internet Request for Comments.
- J. Nightingale. Fraudulent \*.google.com Certificate. *Mozilla Security Blog*, Aug 2011. URL <https://blog.mozilla.org/security/2011/08/29/fraudulent-google-com-certificate/>. Online; accessed 2020-09-18.
- NIST. FIPS PUB 199. Standards for Security Categorization of Federal Information and Information Systems, Feb 2004. DOI 10.6028/NIST.FIPS.199.
- NIST IR 8105. Report on Post-Quantum Cryptography, 2016. DOI 10.6028/NIST.IR.8105.
- NIST SP 800-57. NIST Special Publication 800-57 – Recommendation for Key Management—Part 1: General (Revision 5), 2020. DOI 10.6028/NIST.SP.800-57pt1r5.
- NIST SP 800-88. NIST Special Publication 800-88 – Guidelines for Media Sanitization (Revision 1), 2014. DOI /10.6028/NIST.SP.800-88r1.
- M. Nottingham. Web Linking. RFC 8288, October 2017. DOI 10.17487/RFC8288. Internet Request for Comments.
- NRO. Free pool of ipv4 address space depleted. 2011. URL <https://www.nro.net/news/ipv4-free-pool-depleted>. Online; accessed 2020-09-18.

- V. G. Oklobdzija, V. M. Stojanovic, D. M. Markovic, and N. M. Nedovic. *Digital System Clocking: High-Performance and Low-Power Aspects*. John Wiley & Sons, 2003. DOI 10.1002/0471723703.
- Open Connectivity Foundation. OCF Specification Overview, Feb 2019. URL <https://openconnectivity.org/wp-content/uploads/2019/03/2.-OCF-Architecture-Introduction.pdf>. Online; accessed 2020-09-17.
- Open Mobile Alliance. LwM2M v1.1, 2019. URL [http://openmobilealliance.org/release/LightweightM2M/Lightweight\\_Machine\\_to\\_Machine-v1\\_1-OMASpecworks.pdf](http://openmobilealliance.org/release/LightweightM2M/Lightweight_Machine_to_Machine-v1_1-OMASpecworks.pdf). Online; accessed 2020-09-16.
- S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh. Formal Verification of OAuth 2.0 using Alloy Framework. In *2011 International Conference on Communication Systems and Network Technologies*, pages 655–659, 2011. DOI 10.1109/CSNT.2011.141.
- J. Postel. User Datagram Protocol. STD 6/RFC 768, August 1980. DOI 10.17487/RFC0768. Internet Request for Comments.
- J. Postel. Internet Protocol. STD 5/RFC 791, September 1981a. DOI 10.17487/RFC0791. Internet Request for Comments.
- J. Postel. Transmission Control Protocol. STD 7/RFC 793, September 1981b. DOI 10.17487/RFC0793. Internet Request for Comments.
- M. Pritikin, M. Richardson, T. Eckert, M. Behringer, and K. Watsen. Bootstrapping Remote Secure Key Infrastructures (BRSKI). draft-ietf-anima-bootstrapping-keyinfra-41, April 2020. Internet-Draft (Work in Progress).
- E. Rescorla. *SSL and TLS*. Addison-Wesley, 2001.
- E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. DOI 10.17487/RFC8446. Internet Request for Comments.
- E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, January 2012. DOI 10.17487/RFC6347. Internet Request for Comments.

- E. Rescorla, H. Tschofenig, and N. Modadugu. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. draft-ietf-tls-dtls13-38, May 2020. Internet-Draft (Work in Progress).
- J. Richer, M. Jones, J. Bradley, M. Machulak, and P. Hunt. OAuth 2.0 Dynamic Client Registration Protocol. RFC 7591, July 2015. DOI 10.17487/RFC7591. Internet Request for Comments.
- B. K. Rios. When IoT Attacks. Black Hat, 2017. URL <https://www.blackhat.com/docs/us-17/wednesday/us-17-Rios-When-IoT-Attacks-Understanding-The-Safety-Risks-Associated-With-Connected-Devices.pdf>. Online, accessed 2020-09-17.
- P. Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02*, pages 98–107, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581136129. DOI 10.1145/586110.586125.
- S. B. Roosa and S. Schultze. The "Certificate Authority" Trust Model for SSL: A Defective Foundation for Encrypted Web Traffic and a Legal Quagmire. *Intellectual Property & Technology Law Journal*, 22(11), Nov. 2010. URL [http://citpsite.s3.amazonaws.com/publications/Roosa\\_Schultze\\_CA\\_Trust\\_Model.pdf](http://citpsite.s3.amazonaws.com/publications/Roosa_Schultze_CA_Trust_Model.pdf). Online; accessed 2020-09-18.
- N. Sakimura, J. Bradley, M. B. Jones, B. de Medeiros, and C. Mortimore. OpenID Connect Core 1.0 incorporating errata set 1, Nov 2014. URL [http://openid.net/specs/openid-connect-core-1\\_0.html](http://openid.net/specs/openid-connect-core-1_0.html). Online; accessed 2020-09-18.
- J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sept 1975. ISSN 0018-9219. DOI 10.1109/PROC.1975.9939.
- R. S. Sandhu and P. Samarati. Access Control: Principle and Practice. *IEEE Communications Magazine*, 32(9):40–48, Sept 1994. ISSN 0163-6804. DOI 10.1109/35.312842.
- J. Schaad. CBOR Object Signing and Encryption (COSE). RFC 8152, July 2017. DOI 10.17487/RFC8152. Internet Request for Comments.

- Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch. The Rise of Certificate Transparency and Its Implications on the Internet Ecosystem. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, pages 343–349, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356190. DOI 10.1145/3278532.3278562.
- B. Schneier. *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1996.
- B. Schneier. Security and Privacy Guidelines for the Internet of Things. *Schneier on Security*, Feb 2017. URL [https://www.schneier.com/blog/archives/2017/02/security\\_and\\_pr.html](https://www.schneier.com/blog/archives/2017/02/security_and_pr.html). Online; accessed 2020-09-18.
- L. Seitz. Additional OAuth Parameters for Authorization in Constrained Environments (ACE). draft-ietf-ace-oauth-params-00, September 2018a. Internet-Draft (Work in Progress).
- L. Seitz. Additional OAuth Parameters for Authorization in Constrained Environments (ACE). draft-ietf-ace-oauth-params-01, Nov 2018b. Internet-Draft (Work in Progress).
- L. Seitz. Additional OAuth Parameters for Authorization in Constrained Environments (ACE). draft-ietf-ace-oauth-params-13, April 2020. Internet-Draft (Work in Progress).
- L. Seitz, S. Gerdes, G. Selander, M. Mani, and S. Kumar. Use Cases for Authentication and Authorization in Constrained Environments. RFC 7744, January 2016. DOI 10.17487/RFC7744. Internet Request for Comments.
- L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE). draft-ietf-ace-oauth-authz-06, Mar 2017. Internet-Draft (Work in Progress).
- L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE). draft-ietf-ace-oauth-authz-16, Oct 2018a. Internet-Draft (Work in Progress).

- 
- L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE). draft-ietf-ace-oauth-authz-17, Nov 2018b. Internet-Draft (Work in Progress).
- L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE). draft-ietf-ace-oauth-authz-18, Jan 2019a. Internet-Draft (Work in Progress).
- L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth). draft-ietf-ace-oauth-authz-21, Feb 2019b. Internet-Draft (Work in Progress).
- L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth). draft-ietf-ace-oauth-authz-35, June 2020. Internet-Draft (Work in Progress).
- G. Selander, J. Mattsson, F. Palombini, and L. Seitz. Object Security for Constrained RESTful Environments (OSCORE). RFC 8613, July 2019. DOI 10.17487/RFC8613. Internet Request for Comments.
- Z. Shelby. Constrained RESTful Environments (CoRE) Link Format. RFC 6690, August 2012. DOI 10.17487/RFC6690. Internet Request for Comments.
- Z. Shelby and C. Bormann. *6LoWPAN*. John Wiley & Sons, 2009.
- Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014. DOI 10.17487/RFC7252. Internet Request for Comments.
- Z. Shelby, M. Koster, C. Bormann, P. van der Stok, and C. Amsuess. CoRE Resource Directory. draft-ietf-core-resource-directory-25, July 2020. Internet-Draft (Work in Progress).
- R. Shirey. Internet Security Glossary, Version 2. RFC 4949, August 2007. DOI 10.17487/RFC4949. Internet Request for Comments.

- P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review*, 41(2):303–332, 1999. ISSN 0036-1445. DOI /10.1137/S0036144598347011.
- P. Stacewicz and A. Wlodarczyk. Modeling in the Context of Computer Science – A Methodological Approach. *Studies in Logic, Grammar and Rhetoric*, 20(33): 155–179, 2010.
- F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Proceedings of 3rd AT&T Software Symposium*, Middletown, NJ, 1999.
- U. Stern and D. L. Dill. Improved Probabilistic Verification by Hash Compaction. In P. E. Camurati and H. Eweking, editors, *Correct Hardware Design and Verification Methods*, pages 206–224. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-45516-5. DOI 10.1007/3-540-60385-9\_13.
- Tenable Research. Tenable Research Advisory: Peekaboo Critical Vulnerability in NUUO Network Video Recorder. Sep 2018. URL <https://www.tenable.com/blog/tenable-research-advisory-peekaboo-critical-vulnerability-in-nuuo-network-video-recorder>. Online; accessed 2020-09-28.
- H. Tschofenig and S. Farrell. Report from the Internet of Things Software Update (IoTSU) Workshop 2016. RFC 8240, September 2017. DOI 10.17487/RFC8240. Internet Request for Comments.
- J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Framework. RFC 2904, August 2000. DOI 10.17487/RFC2904. Internet Request for Comments.
- M. M. Waldrop. More than Moore. *Nature*, 530:144–147, 2 2016.
- K. Watsen, M. Richardson, M. Pritikin, and T. Eckert. A Voucher Artifact for Bootstrapping Protocols. RFC 8366, May 2018. DOI 10.17487/RFC8366. Internet Request for Comments.
- G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, Mar 2006.

- G. Williamson, D. Yip, and I. Sharoni. *Identity Management: A Primer*. Mc Press, 2009.
- J. Woetzel, J. Remes, B. Boland, K. Lv, S. Sinha, G. Strube, J. Means, J. Law, A. Cadena, and V. von der Tann. Smart Cities: Digital Solutions For a More Livable Future. 2018. [https://www.mckinsey.com/~media/McKinsey/Industries/Public and Social Sector/Our Insights/Smart cities Digital solutions for a more livable future/MGI-Smart-Cities-Full-Report.pdf](https://www.mckinsey.com/~media/McKinsey/Industries/Public%20and%20Social%20Sector/Our%20Insights/Smart%20cities%20Digital%20solutions%20for%20a%20more%20livable%20future/MGI-Smart-Cities-Full-Report.pdf). Online; accessed 2020-09-26.
- T. Y. Woo and S. S. Lam. Authorization in Distributed Systems: a Formal Approach. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 33–50, May 1992a. ISBN 0-8186-2825-1. DOI 10.1109/RISP.1992.213272.
- T. Y. Woo and S. S. Lam. Authentication for Distributed Systems. *Computer*, 25(1): 39–52, Jan 1992b. ISSN 0018-9162. DOI 10.1109/2.108052.
- P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen. Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7250, June 2014. DOI 10.17487/RFC7250. Internet Request for Comments.
- Xerox. Statements of Volatility for Xerox Products, 2020. URL <https://security.business.xerox.com/en-us/documents/statements-of-volatility/>. Online; accessed 2020-09-16.
- T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006. DOI 10.17487/RFC4251. Internet Request for Comments.
- C. Young. Tripwire Research: IoT Smart Lock Vulnerability Spotlights Bigger Issues. Aug 2020. URL <https://www.tripwire.com/state-of-security/featured/tripwire-research-iot-smart-lock-vulnerability/>. Online; accessed 2020-09-30.