Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Basic Resource Trading Mechanisms and Access Scopes

The symbIoTe Consortium

Intracom SA Telecom Solutions, ICOM, Greece
Sveučiliste u Zagrebu Fakultet elektrotehnike i računarstva, UNIZG-FER, Croatia
AIT Austrian Institute of Technology GmbH, AIT, Austria
Nextworks Srl, NXW, Italy
Consorzio Nazionale Interuniversitario per le Telecomunicazioni, CNIT, Italy
ATOS Spain SA, ATOS, Spain
University of Vienna, Faculty of Computer Science, UNIVIE, Austria
Unidata S.p.A., UNIDATA, Italy
Sensing & Control System S.L., S&C, Spain
Fraunhofer IOSB, IOSB, Germany
Ubiwhere, Lda, UW, Portugal
VIPnet, d.o.o, VIP, Croatia
Instytut Chemii Bioorganicznej Polskiej Akademii Nauk, PSNC, Poland
NA.VI.GO. SCARL, NAVIGO, Italy

© Copyright 2016, the Members of the symbIoTe Consortium

For more information on this document or the symbIoTe project, please contact:
Sergios Soursos, INTRACOM TELECOM, souse@intracom-telecom.com

Document Control

Number: D3.1

Title: Basic Resource Trading Mechanisms and Access Scopes

Type: Public

Editor(s): Peter Reichl, UNIVIE; Gerard Frankowski, PSNC

E-mail: peter.reichl@univie.ac.at; gerard.frankowski@man.poznan.pl

Author(s): Giuseppe Bianchi, Gennaro Boggia, Daniele Caldarola, Gerard Frankowski, João Garcia, Elena Garrido, Nemanja Ignjatov, Gabriel Kovacs, Michał Pilc, Giuseppe Piro, Peter Reichl, Savio Sciancalepore (in alphabetical order)

Doc ID: D3.1- Basic Resource Trading Mechanisms and Access Scopes

Amendment History

Version	Date	Author	Description/Comments
v0.1	Sep 25	P. Reichl, G. Frankowski	Initial ToC
V0.2	Oct 18	S.Sciancalepore, G.Piro, G.Boggia, G.Bianchi	Sections 4.2 and 4.3
V0.3	Oct 18	M. Pilc	Sections 4.1 and 4.3
V0.4	Oct. 21	S.Sciancalepore, D.Caldarola, G.Piro, G.Boggia, G.Bianchi	Sections 4.2, 4.3 and 4.4
V0.5	Oct 26	E. Garrido	Section 3.4
V0.6	Oct 26	P. Reichl, G. Kovacs	Sections 3.1, 3.2
V0.7	Oct 27	P. Reichl, G. Kovacs, N. Ignjatov	Section 3.3 – sequence diagrams
V0.8	Oct 28	M. Pilc	Section 4.1 and 4.2
V0.9	Nov. 3	S.Sciancalepore, D. Caldarola, G. Piro, G.Boggia, G.Bianchi	Sections 4.2, 4.3, 4.4 and 4.5
V0.10	Nov.4	S.Sciancalepore, D. Caldarola, G. Piro, G.Boggia, G.Bianchi	Sections 4.2, 4.3,4.4, 4.5 and 4.6
V0.11	Nov 7	P. Reichl, G. Kovacs, N. Ignjatov	Section 3.3 – missing diagrams
V0.12	Nov 8	João Garcia	Section 3.5
V0.13	Nov 8	P. Reichl	Update ToC + comments
V0.14	Nov 20	P. Reichl	Finishing Chapter 3 + reformatting document
V0.15	Nov 21	M. Pilc	Finishing Chapter 4 + references + Chapter 5
V0.16	Nov 21	João Garcia	Extended Section 3.5
V0.17	Nov 22	S. Sciancalepore, D. Caldarola, G. Piro, G.Boggia, G.Bianchi	Section 4, Next Steps in T3.2
V0.18	Nov 24	M. Pilc	Internal proofreading
V0.19	Nov 28	M. Pilc, E. Garrido, P. Reichl, G. Kovacs	Adressing comments from internal review
V0.20	Nov 29	E. Garrido	Adressing comments and corrections from conf call
V0.21	Nov 29	P. Reichl	Final integration
V1.0	Nov 30	P. Reichl and all authors	Final version ready for submission

© Copyright 2016, the Members of the symbIoTe Consortium

Legal Notices

The information in this document is subject to change without notice.

The Members of the symbIoTe Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the symbIoTe Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

© Copyright 2016, the Members of the symbIoTe Consortium

(This page is left blank intentionally.)

© Copyright 2016, the Members of the symbIoTe Consortium

Table of Contents

1 Executive Summary 7

2 Introduction 9

2.1 Purpose of the Document 9

2.2 Structure and Overview 9

3 Bartering and Trading 10

3.1 Overview of Work in T3.1 10

3.2 Fundamentals and Related Work 10

3.2.1 Basic Concepts 10

3.2.2 Related Work 12

3.3 Basic Models 13

3.3.1 Direct Buy – Payment to Core 13

3.3.2 Direct Buy – Payment to Platform 17

3.3.3 Forward Auction 22

3.3.4 Reverse Auction 28

3.3.5 Voucher-based Bartering 32

3.4 Payment Systems 36

3.5 Conclusions for symbIoTe Architecture 38

4 Security and Access Scopes 40

4.1 Overview of Work in T3.2 40

4.1.1 Security Requirements 40

4.1.2 Summary of Activities 41

4.2 Authentication and Authorization 42

4.2.1 Attribute-Based Access Control 42

4.2.2 Review of the Available Solutions 44

4.3 Security Architecture and Components 48

4.3.1 Main Security Rationale 49

4.3.2 Core Authentication and Authorization Manager 51

4.3.3 Platform Authentication and Authorization Manager 52

4.3.4 Security Handler 53

4.4 Token Formats and Comparison 53

4.4.1 Token Format and Security Requirements 53

4.4.2 Token Content and Format in symbIoTe 59

4.4.3 Evaluation of Macaroons and JSON Web Tokens 60

© Copyright 2016, the Members of the symbIoTe Consortium

4.5 Interfaces and Services for the Security Components 61

4.5.1 Core AAM: Interfaces and Services 62

4.5.2 Platform AAM: Interfaces and Services 64

4.5.3 Security Handler (SH): Interfaces and Services 66

4.6 Preliminary Implementation 68

4.6.1 Level-1 Sequence Diagram: Access to Resources (without reservation) 68

4.6.2 Details on the Preliminary Implementation 71

4.7 Other Security Related Issues 77

4.8 Summary 78

5 Next Steps for T3.1 and T3.2 79

6 References 80

7 List of Acronyms 82

© Copyright 2016, the Members of the symbIoTe Consortium

1 Executive Summary

This document reports on initial work in WP 3 “IoT Platform Federation” of the symbIoTe project and aims at providing an introduction into the bartering and trading of resources in a symbIoTe context, as well as a report on the discussion on security and access scopes.

After a brief introduction into the main purpose and the structure of the document, this deliverable includes two main parts, the first one focusing on bartering and trading issues in B2B use cases. Here, trading refers to scenarios where platforms use symbIoTe to offer their resources to other interested platforms or applications/enablers and are paid for it. Prices are either set explicitly by the producer (Direct Buy) or determined through suitable auction mechanisms.

In contrast, resource bartering refers to a matching process between the needs and offers of two platforms without monetary implications. In this case, it is important to recognize that the participants act as prosumers, i.e. as producers and consumers at the same time. The actual matching process is based on vouchers that include a Service Level Agreement (SLA) together with details on the requested service, the expected price and further parameters.

While there is plenty of related work on pricing and auction theory available, which is not repeated in this deliverable, a broad search for related bartering mechanisms reveals that there are neither open software platforms available, nor exist platforms for digital goods and/or B2B scenarios.

Next, as one of the main contributions of this deliverable, five key scenarios for trading and bartering are analysed in detail with the help of comprehensive message sequence charts: two versions of Direct Buy (depending on whether the payment is processed via the Core Bartering and Trading component, or directly between the participating platforms), two version of auctions (Forward vs Reverse Auctions), and the standard voucher-based bartering scenario.

Then, as an important side topic, the question of corresponding payment systems is addressed, concluding that an integrated payment system would be nice to have but is not considered to be in the primary scope of symbIoTe and hence will not be involved directly. The first part ends with a couple of conclusions for the symbIoTe architecture.

The second part of the deliverable is dedicated to security issues, which cover a crucial aspect of symbIoTe and must be addressed from an early stage of the project development life cycle. More specifically, the deliverable provides information on work and achievements obtained in the T3.2 task that concentrates on all security related aspects in symbIoTe. Following the security requirements established in D1.2, a basic security architecture and protocols have been designed that should be incorporated into the symbIoTe system to provide user, application, enabler (i.e. entity) as well as platform authentication and authorization.

Hence, security requirements are elaborated, which are a required basis for reliable determining the necessary set of solutions. The requirements are derived from a subset of IoT use cases of the existing IoT platforms and general security knowledge about distributed computing systems, together with the identification of threats. Thus, requirements for different use cases may also be different. A deep analysis of system requirements and definition of symbIoTe architecture serves as starting point for designing the security architecture.

© Copyright 2016, the Members of the symbIoTe Consortium

Therefore, a set of analyses and comparisons are performed in order to select the optimal security solutions in particular areas. As a result, the Attribute-Based Access Control (ABAC) method of authorization has been decided to be the most suitable one for symbIoTe, preferred over role-based access control (RBAC) and group-based access control (GBAC) mainly due to flexibility in the networked and federated IoT environment. ABAC can be implemented with one of the authorization tokens: Macaroons or JSON Web Tokens – an extensive analysis has shown the latter to be optimal for symbIoTe.

For documentary purposes and in order more clearly justify the decisions, also information on the analyzed solutions that finally have been considered not optimal (and thus not selected), is provided.

The resulting security architecture and components are described as well. The main security modules include the Core Authentication and Authorization Manager (Core AAM), the Security Handler (SH) and the components on the platform level: Authentication and Authorization Manager (AAM) and Resource Access Proxy (RAP). All the aforementioned modules are described in detail, explaining their tasks and architectural placement in the context of the derived requirements and particular scenarios. The detailed description of interfaces and services envisioned for each security components is provided as well.

Based on that, our preliminary implementation of a proof-of-concept is described, which has been developed to demonstrate the interactions between applications and components in the symbIoTe ecosystem. The implementation consists of eight steps, i.e. (1) requesting a core token, (2) creating a core token, (3) returning the core token to the application, (4) requesting a foreign token from the SH, (5) core token validation with the foreign AAM, (6) requesting the foreign token from the foreign AAM, (7) returning the foreign token to the application, and (8) accessing foreign resources with the foreign token.

The second part concludes with a discussion on further security issues relevant for the symbIoTe context. Anomaly detection is discussed in some detail, especially with respect to centralized vs decentralized approaches, and a preliminary implementation proposal is presented.

Finally, we depict an outlook on the future work for both involved task T3.1 and T3.2. Next steps in the field of bartering and trading will adress B2C scenarios as well as more complex bartering situations (e.g. circular bartering and voucher composition). Modelling utility functions and taking Quality of Service (QoS) and Quality of Experience (QoE) aspects into account will support resource access management, while also contributions to the detailed specifications of several symbIoTe modules are planned which will further facilitate the bartering and trading functionality.

One key security concept, which is not covered yet but considered significant for the further stages of symbIoTe, concerns the anomaly detection functionality that is mentioned in DoW for T3.2. The approach based on discovering anomalies, which are patterns of data that do not conform to a well-defined notion of behaviour, allows detecting unknown threats as well as flaws or failures not related to security. Based on our analysis, it is planned to propose suitable architecture adjustments to enrich it with anomaly detection capabilities, while avoiding the introduction of unacceptable processing overheads.

© Copyright 2016, the Members of the symbIoTe Consortium

2 Introduction

According to the overall work plan, the first phase of the symbIoTe project is focusing on WP1, including the detailed specification of use cases (see Deliverable D1.1) as well as a first version of the overall symbIoTe architecture (see Deliverable D1.2) and an initial analysis of the business ecosystem (see Deliverable D1.3, to be published in M12). Based on this work, WP3 together with WP2 and WP4 has initiated the second project phase, and deals with key aspects of IoT platform federation within the Cloud Domain, aiming at the development of mechanisms for interoperability, security (including anomaly detection), trading and optimization mechanisms concerning Quality of Experience, as well as cost and energy efficiency. The present deliverable reports on early work in WP3.

2.1 Purpose of the Document

The purpose of this deliverable is to provide an introduction into the bartering and trading of resources in a symbIoTe context, as well as a report on the discussion on security and access scopes. Hence, D3.1 will document the initial work of tasks T3.1 “Resource Trading and User-centricity” and T3.2 “Security and Access Scopes”, while both tasks will continue to run until month M26. More specifically, as pointed out in the DoW, this document will report on activities related to resource trading mechanisms and access scopes in the context of IoT platform federations.

2.2 Structure and Overview

Hence, the remainder of this deliverable is structured as follows:

Chapter 3 is devoted to work in the area of bartering and trading IoT resources. It starts with a basic overview about important concepts and related work in section 3.2, before analysing in detail five key scenarios in section 3.3 and providing the corresponding sequence diagrams. In this context, the question of underlying payment systems is decoupled from the symbIoTe scope and dealt with separately in section 3.4. Finally, section 3.5 emphasizes the link to Deliverable D1.2, providing conclusions of this work for the symbIoTe architecture.

Chapter 4 focuses on our work on security mechanisms, with an emphasis on the topic of access scopes. To this end, Attribute-Based Access Control (including available solutions) is presented as a key approach for authentication and authorization. Based on this, symbIoTe’s security reference architecture is presented, and three key components are described in detail, i.e. the Core Authentication and Authorization Manager, the Platform Authentication and Authorization Manager, and the Security Handler. As a next step – after having introduced the concept of tokens, including their requirements, content and format – the interfaces and services for these components are discussed. The chapter continues with a description of the initial implementation before discussing anomaly detection aspects and concluding with a brief summary.

Chapter 5 concludes the deliverable with an outlook on future work planned in both T3.1 and T3.2, which will contribute to the remaining deliverables in WP3, i.e. D3.2 “Resource Trading, Security and Federation Mechanisms” (due month M22) and D3.3 “Complete Federation Environment” (due month M30). References and an acronym list are included in Chapters 6 and 7, resp.

© Copyright 2016, the Members of the symbIoTe Consortium

3 Bartering and Trading

Bartering and trading of resources is one of the central benefits for the stakeholders of the symbIoTe ecosystem. Note that in this deliverable we mainly focus on the B2B case, which is considered to have higher relevance, while a detailed analysis of B2C scenarios is shifted towards future work.

3.1 Overview of Work in T3.1

In this chapter, we report on the current state of the activities concerning resource-trading mechanisms in the context of IoT platform federations. This is part of task T3.1 (running from M6 to M26) which, according to the DoW, investigates resource-trading aspects from an interdisciplinary perspective, with a specific focus on mechanism which seamlessly adapt to user needs, user utilities and user preferences (e.g., in terms of subjective Quality of Experience, QoE). While the present document mainly refers to the state of the art on this topic, the future roadmap foresees defining reasonable parameters for basic resource trading, as well as the detailed modelling of prosumers. More specifically, next steps will include modelling of utility functions which consider tradeoffs as a way to manage resource allocation internally, while comparing aspects related to the expected Quality of Service (QoS) (i.e. response time vs availability) and cost functions related to the internal cost of symbIoTe resources.

3.2 Fundamentals and Related Work

In this subsection, we outline the fundamental concepts of bartering and trading before summarizing related approaches.

3.2.1 Basic Concepts

The basic economic concept of *bartering* refers to a market situation where two or more market participants exchange their respective goods or services directly for other goods or services, without monetary implications. While the concept itself is a rather old one, it has been repeatedly criticized for its alleged inefficiency, for instance with respect to difficulties in matching suitable partners, issues with determining common value metrics, and problems arising from the fact that certain goods may be indivisible and hence impossible to precisely match in terms of their value. Eventually, the main justification for employing a bartering mechanism originates from the fact that it allows two parties achieving a joint win-win situation without the need of resorting to the explicit exchange of money.

In the context of an IoT middleware like symbIoTe, most of the aforementioned problems disappear by definition: matching suitable partners is relatively easy, as all platforms participating in symbIoTe are assumed to be *prosumers*, i.e. are interested in offering services to other platforms (as *producers*) and using services from other platforms (as *consumers*) at the same time. Hereby, a *service* typically consists of allowing or making use of access to IoT resources, e.g. sensors and their corresponding data, which circumvents the problem of indivisibility: we can easily define small *units of service* and thus provide a mutually acceptable metrical unit for comparing the value/worth of an offer or a request.

© Copyright 2016, the Members of the symbIoTe Consortium

From a more formal modelling perspective, bartering schemes are typically based on the concept of *indifference curves*, i.e. curves in a two-dimensional coordinate system that indicate combinations of services which are considered to be of equal value to the respective other party. Then, bartering can be performed along these curves, exchanging a certain amount *a* of service s_A against an amount *b* of service s_B, where both service quantities are considered of equal worth. In the case of two parties, this approach is well-known as “Edgeworth diagrams”, see for instance [1].

However, in order to increase the efficiency of the mechanism, we will not employ bartering in this purest form, but instead introduce commonly accepted *vouchers* as a means to subsume all important properties referring to a service offer or service request. Hence, a voucher typically includes

- access token
- Service Level Agreement (SLA)
- details on requested service (wanted)
- details on expected value (price)
- time constraints (e.g. timeout conditions)

Further details on voucher-based bartering schemes are explained in the Section 3.3.

Of course, symbIoTe will also offer a platform ways to access resources from other platforms without an immediate material counteroffer, i.e. by *trading*. Here, three basic scenarios have to be considered:

- *Direct Buy:* a platform sells access to own resources to an application/enabler or another platform for a fixed price;
- *Forward Trading:* a platform is offering access to own resources and asks for corresponding requests (bids) from other platforms;
- *Backward (reverse) Trading:* a platform is looking for access to resources offered by foreign platform(s).

Here, an agreement on monetary compensation is fundamental for closing a deal. In microeconomics, such situations are usually treated within the framework of auction theory, i.e. forward auctions (access to resources is offered, and requests are submitted in the form of bids) and reverse auctions (access to resources is requested, and corresponding offers including access conditions received by the platform). The symbIoTe approach is focusing on a suitable adaptation of Progressive Second-Price (PSP) auctions, or a more general Vickrey-Clarke-Groves (VCG) mechanism, which have been proven to be incentive compatible and thus force auction participants to be honest concerning their estimations about the value of the offered/requested resources.

Basically, second-price auctions are so-called closed envelope auctions, where all bidders submit their bid (desired quantity and offered price per unit quantity) individually and secretly to the auctioneer before the deadline. As soon as the deadline has passed, the auctioneer opens all bids, and determines the highest bid as winner of the auction. However, the price to be paid by the winner is not determined by his own bid, but by the bid of the highest-bidding loser. Thus, winning an auction is decoupled from the price to be paid by the winner, which yields some very desirable properties. This principle has been generalized by Lazar and Semret [2]. Here, the basic idea is to consider an auction of a

© Copyright 2016, the Members of the symbIoTe Consortium

divisible resource and with multiple participants who are bidding on potentially different amounts of the resource and to calculate the price to be paid by a winning bidder as the overall damage s/he is causing to participants who are losing due to the sheer existence of the winning bidder.

An additional characteristic of the auctions we are considering is the existence of a so-called reserve price, i.e. a minimal bid valid for all potential participants. In this way, we can map the “Direct buy” scenarios with a fixed price to the auction paradigm.

3.2.2 Related Work

Bartering and trading mechanism are widely used in today’s Internet, and hence a plethora of different websites which enable businesses as well as individuals to trade or barter their goods and services are available. Even if the number of platforms is high, their modus operandi is usually very similar: the registered user publishes her/his offer and what s/he wishes to get in return. As in most cases there is no automated matching algorithm, hence the users must actively engage in finding a trading/barter partner by means of a search engine available on the platform. Some platforms offer just bartering (e.g., service vs service; goods vs goods), while others also make it possible for users to buy goods/services using money or a virtual currency.

The service vs service (e.g., piano lessons vs gardening) use case is rather limited to smaller communities (e.g., residents of a city, like for instance with Local Exchange Trading Systems or LETS), while the goods vs goods use case can even stretch over country borders. In almost all cases the users have to deal with the shipping by themselves without any involvement of the platform.

However, we were not able to identify any Web-based platform that offers digital services or a remotely similar functionality to the one envisioned in symbIoTe. Nevertheless, we have taken a closer look to the following websites:

- http://www.tradeaway.com/: claims to be “the world’s largest bartering site”, aiming at a broad range of products and services.
- https://www.listia.com: mobile bartering app, employing some sort of credit scheme (including initial free credits)
- http://www.barterquest.com: specializing on luxury goods, offering a dedicated matching algorithm
- https://www.swap.com: mainly for women and baby clothing

Further similar web sites include:

- http://www.swapace.com/index.php
- http://gametz.com
- https://www.freecycle.org
- http://neighborgoods.net
- http://www.swapright.com
- http://www.u-exchange.com
- https://www.leaptrade.com

© Copyright 2016, the Members of the symbIoTe Consortium

From these examples, the following conclusions may be drawn:

- To the best of our knowledge, there are hardly any open software platforms for bartering available which might be reused in the symbIoTe framework.
- There is hardly any bartering platform for digital goods available. More specifically, we could not find any bartering platform specializing on IoT resources.
- There is hardly any bartering platform for B2B scenarios. Instead, typically they address end users only.
- Almost all current bartering platforms focus on bilateral consensus without automatic support. Eventually, the platform service usually boils down to enabling communication between two potentially interested parties and leave the rest to their direct discussion.
- Typically, current bartering platforms focus on local exchange of goods.

3.3 Basic Models

In this section, we focus on five key bartering and trading models considered to be relevant for symbIoTe:

- Direct Buy – Payment to Core
- Direct Buy – Payment to Platform
- Forward auction
- Reverse auction
- Voucher-based Bartering

Note that this section includes only a very simple usage of vouchers, while more complex voucher-based scenarios (e.g. voucher composition and circular bartering) will be left to further work and will be reported in D3.2.

For the following diagrams, please refer also to D1.2 [7] concerning the usage of home/ core/foreign tokens as well as details on the functionality of the different modules involved (e.g. Federation Manager, AAM, SH, Search Engine, Registry, etc.). Moreover, note that for reasons of completeness and consistency, the following sequence diagrams include <u>all</u> relevant messages (which means that some parts of the description might be considered redundant; for instance, messages 1–20 in Fig. 3.1 correspond precisely to the search functionality already described in D1.2 [7] but have been here included nevertheless, etc.).

3.3.1 Direct Buy – Payment to Core

Focusing on B2B scenarios, Fig. 3.1 depicts the most basic case of trading, where a symbIoTe-compliant application/enabler buys a service (*e.g.* resource access) from a symbIoTe-compliant platform for a fixed price, which is paid via the Core Bartering and Trading component. Note that, without loss of generality, the buying party can also be another symbIoTe-enabled platform instead of an application/enabler – this would have no significant impact on the message flow; therefore, in order to stay consistent with D1.2 [7], we have decided to stay with this notion.

© Copyright 2016, the Members of the symbIoTe Consortium

Summarizing this scenario briefly, suppose that a platform (producer) has registered, within the symbIoTe Core, a resource together with a fixed price for it. Assume further that the Core Search Engine provides result even if the seeker of the resource does not possess the necessary attributes. Finally, it is considered out of scope which payment system (out of several offered by the Core) will be chosen by the consumer if it comes to paying her bill.

Then, the message sequence diagram initially describes how the Application/Enabler (consumer) makes use of the Core Search Engine to find the desired resource. The Application choses a presented resource and signals its intent to buy it which gets forwarded to the Core Bartering and Trading component. The Core Bartering and Trading component initiates the payment and updates the status of the resource after the successful financial transaction. In the next step the Core Bartering and Trading informs (certificate) the producer (IoT platform) about the successful purchase of its resource. After the successful processing of the information within the producers IoT platform the buyer receives a confirmation for the purchase.

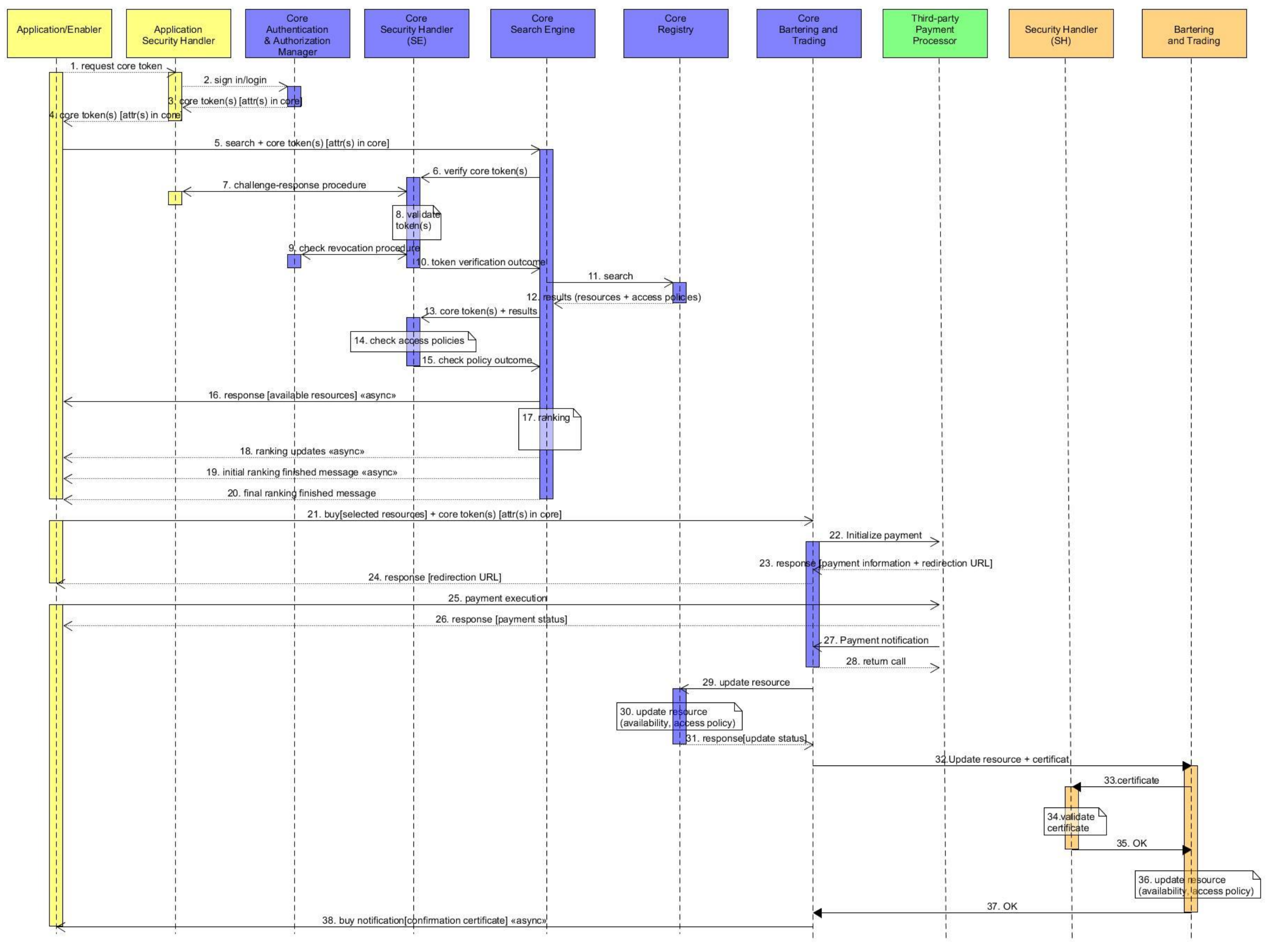

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Security Handler (SE)
Core Search Engine
Core Registry
Core Bartering and Trading
Third-party Payment Processor
Security Handler (SH)
Bartering and Trading
1. request core token
2. sign in/login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
5. search + core token(s) [attr(s) in core]
6. verify core token(s)
7. challenge-response procedure
8. validate token(s)
9. check revocation procedure
10. token verification outcome
11. search
12. results (resources + access policies)
13. core token(s) + results
14. check access policies
15. check policy outcome
16. response [available resources] «async»
17. ranking
18. ranking updates «async»
19. initial ranking finished message «async»
20. final ranking finished message
21. buy[selected resources] + core token(s) [attr(s) in core]
22. Initialize payment
23. response [payment information + redirection URL]
24. response [redirection URL]
25. payment execution
26. response [payment status]
27. Payment notification
28. return call
29. update resource
30. update resource (availability, access policy)
31. response[update status]
32.Update resource + certificat
33.certificate
34.validate certificate
35. OK
36. update resource (availability, access policy)
37. OK
38. buy notification[confirmation certificate] «async»

Figure 3.1: Direct Buy – Payment to Core

© Copyright 2016, the Members of the symbIoTe Consortium

Note that the upper part of this sequence (messages 1 – 20) diagram is directly taken from the symbIoTe search functionality as described in D1.2 [7] and therefore is not directly related to buying a service, but rather to the common procedure of finding a service in the SymbIoTe system.

Altogether, the diagram comprises the following 38 messages and/or procedures:

Message 1 (optional): generated by the Application/Enabler and sent to the Application Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.

Message 2 (optional) (AppAAInterface): generated by the Application Security Handler and sent to the Core AAM in which the Application/Enabler is registered. It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.

Message 3 (optional): generated by the Core AAM in the IoT platform and sent to the Application Security Handler. It is used to provide the core token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.

Message 4 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It is used to deliver the core token(s).

Message 5 (SearchInterface): generated by the Application/Enabler and sent to the Search Engine. It sends a search query and the core token(s) to the Search Engine.

Message 6: generated by the Search Engine and sent to the Core Security Handler. It is used to ask the security handler to verify the complete validity of the token.

Procedure 7 (AppSecurityInterface): procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s).

Procedure 8: verification of the time validity, authenticity and integrity of the provided token(s).

Procedure 9: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).

Message 10: generated by the Core Security Handler and sent to the Search Engine. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.

Message 11: generated by the Search Engine and sent to the Registry. It is used to search available resources.

Message 12: generated by the Registry and sent to the Search Engine. It is used to return the result of the search operation, containing resources and associated access policies.

Message 13: generated by the Search Engine and sent to the Core Security Handler. It is used to deliver the core token(s) previously verified and the results of the search operation.

Procedure 14: procedure that checks, for each resource, if the attributes contained in the core token(s) satisfy the access policy associated to that resource.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 15: generated by the Core Security Handler and sent to the Search Engine. It is used to deliver the result of the previous procedure.

Message 16: generated by the Search Engine and sent to the Application/Enabler asynchronously. It is used to deliver the result of the search operation (available resources).

Procedure 17: executes ranking of resources.

Message 18 (ApplicationInterface): asynchronously sends ranking updates to Application/ Enabler.

Message 19 (ApplicationInterface): asynchronously sends message about the end of initial ranking.

Message 20 (ApplicationInterface): synchronously sends message of the end of final ranking.

Message 21 (optional): generated by the Application/Enabler and sent to the Core Bartering and Trading. It is used to purchase a selected resource.

Message 22 (mandatory): generated by the Core Bartering and Trading and sent to the Payment Processor. It is used to initialize the payment of the previous selected resource.

Message 23 (mandatory): generated by the Payment Processor and sent to the Core Bartering and Trading. It is used provide the transaction data and the redirection URL.

Message 24 (mandatory): generated by the Core Bartering and Trading and sent to the Application/Enabler. It is used to send the redirection URL for the payment process.

Message 25 (mandatory): generated by the Application/Enabler and sent to the Payment Processor. It is used to execute the payment.

Message 26 (mandatory): generated by the Payment Processor and sent to the Application/Enabler. It is used to inform the Application/Enabler about the payment transaction outcome.

Message 27 (mandatory): generated by the Payment Processor and sent to the Core Bartering and Trading. It is used to inform the Core Bartering and Trading about the successful payment execution.

Message 28 (mandatory): generated by the Core Bartering and Trading and sent to the Payment Processor. It is used to acknowledge the previous received message.

Message 29 (mandatory): generated by the Core Bartering and Trading and sent to the Core Registry. It is used to send a request for resource status update.

Procedure 30 (mandatory): the Core Registry updates the resource status and access policy.

Message 31 (mandatory): generated by the Core Registry and sent to the Core Bartering and Trading. It is used to communicate the outcome of the resource status update.

Message 32 (mandatory): generated by the Core Bartering and Trading and sent to the Bartering and Trading of the offering platform. It is used to send a request for resource status update and the certificate which confirms the buying transaction.

Message 33 (mandatory): generated by the Bartering and Trading and sent to the Security Handler. It is used to send the certificate for validation.

© Copyright 2016, the Members of the symbIoTe Consortium

Procedure 34 (mandatory): the Security Handler validates the previously obtained certificate.

Message 35 (mandatory): generated by the Security Handler and sent to the Bartering and Trading. It is used to communicate the outcome of the certificate validation.

Procedure 36 (mandatory): the Bartering and Trading updates the resource availability and access policy.

Message 37 (mandatory): generated by the bartering and Trading and sent to the Core Bartering and Trading. It is used to acknowledge the buying transaction.

Message 38 (mandatory): generated by the Core Bartering and Trading and sent to the Application/Enabler. It is used to confirm the buying transaction.

3.3.2 Direct Buy – Payment to Platform

Another relevant B2B scenario depicts a similar case of one symbIoTe-compliant platform buying a service from another symbIoTe-compliant platform for a fixed price, however payment is done now via the Cloud Bartering and Trading component located in the serving platform.

Note that, again, the buying party can also be another symbIoTe-enabled platform instead of an application/enabler, with essentially the same message sequence diagram.

For this scenario, the following prerequisites are assumed: A platform (producer) has already registered, within the symbIoTe Core, a resource and also appended a fixed price to it. The Core Search Engine provides result even if the seeker of the resource does not possess the necessary attributes.

The Application/Enabler (consumer) makes use of the Core Search Engine to find the desired resource. After identifying the desired resource, the Application acquires a foreign token which will enable it to identify itself within the IoT platform (producer) that offers the desired resource. After the successful authentication of the Application within the producer's platform the payment intention is signaled to the Bartering and Trading component of the producer. Upon a successful payment transaction, the producer will inform the Core Registry about the purchase of the resource, which will update the availability of the given resource. Finally, after the Core Registry acknowledges the successful update, the Application will receive a confirmation for the purchase.

The corresponding message sequence diagram is depicted in Figure 3.2.

Note that, like with Figure 3.1, the upper part of Figure 3.2, comprising messages 1 to 20, is taken directly from the symbIoTe search functionality as described in D1.2 [7] and has been included for reasons of completeness and consistency.

© Copyright 2016, the Members of the symbIoTe Consortium

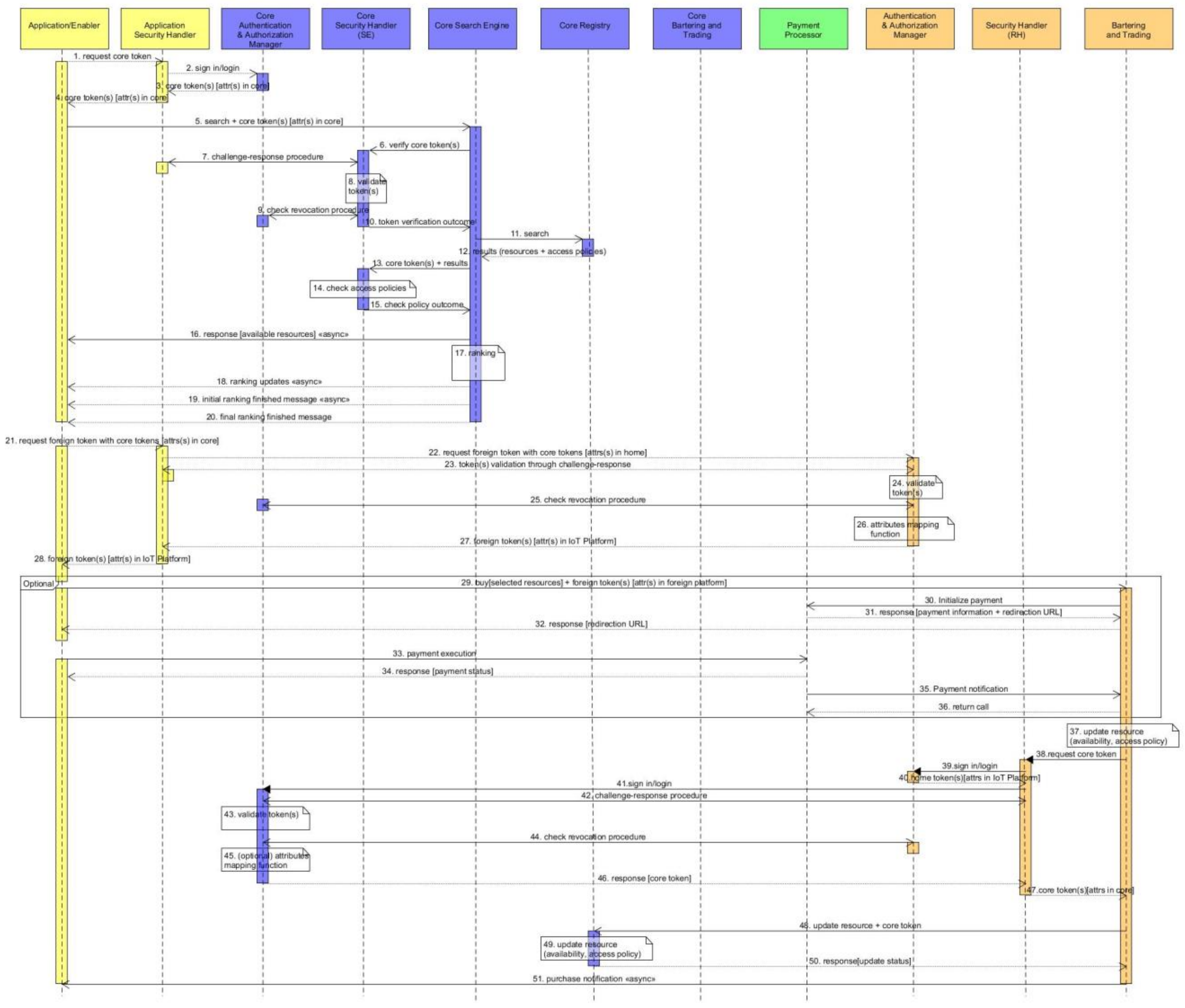

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Security Handler (SE)
Core Search Engine
Core Registry
Core Bartering and Trading
Payment Processor
Authentication & Authorization Manager
Security Handler (RH)
Bartering and Trading
1. request core token
2. sign in/login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
5. search + core token(s) [attr(s) in core]
6. verify core token(s)
7. challenge-response procedure
8. validate token(s)
9. check revocation procedure
10. token verification outcome
11. search
12. results (resources + access policies)
13. core token(s) + results
14. check access policies
15. check policy outcome
16. response [available resources] «async»
17. ranking
18. ranking updates «async»
19. initial ranking finished message «async»
20. final ranking finished message
21. request foreign token with core tokens [attrs(s) in core]
22. request foreign token with core tokens [attrs(s) in home]
23. token(s) validation through challenge-response
24. validate token(s)
25. check revocation procedure
26. attributes mapping function
27. foreign token(s) [attr(s) in IoT Platform]
28. foreign token(s) [attr(s) in IoT Platform]
Optional
29. buy[selected resources] + foreign token(s) [attr(s) in foreign platform]
30. Initialize payment
31. response [payment information + redirection URL]
32. response [redirection URL]
33. payment execution
34. response [payment status]
35. Payment notification
36. return call
37. update resource (availability, access policy)
38.request core token
39.sign in/login
40.home token(s)[attrs in IoT Platform]
41.sign in/login
42. challenge-response procedure
43. validate token(s)
44. check revocation procedure
45. (optional) attributes mapping function
46. response [core token]
47.core token(s)[attrs in core]
48. update resource + core token
49. update resource (availability, access policy)
50. response[update status]
51. purchase notification «async»

Figure 3.2: Direct Buy – Payment to Platform

Hence, Figure 3.2 comprises the following 51 messages and/or procedures:

<u>Message 1 (optional):</u> generated by the Application/Enabler and sent to the Application Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.

<u>Message 2 (optional) (AppAAInterface):</u> generated by the Application Security Handler and sent to the Core AAM in which the Application/Enabler is registered. It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.

<u>Message 3 (optional):</u> generated by the Core AAM in the IoT platform and sent to the Application Security Handler. It is used to provide the core token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.

<u>Message 4 (optional):</u> generated by the Application Security Handler and sent to the Application/Enabler. It is used to deliver the core token(s).

© Copyright 2016, the Members of the symbIoTe Consortium

Message 5 (SearchInterface): generated by the Application/Enabler and sent to the Search Engine. It sends a search query and the core token(s) to the Search Engine.

Message 6: generated by the Search Engine and sent to the Core Security Handler. It is used to ask the security handler to verify the complete validity of the token.

Procedure 7: (AppSecurityInterface): procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s).

Procedure 8: verification of the time validity, authenticity and integrity of the provided token(s).

Procedure 9: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).

Message 10: generated by the Core Security Handler and sent to the Search Engine. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.

Message 11: generated by the Search Engine and sent to the Registry. It is used to search available resources.

Message 12: generated by the Registry and sent to the Search Engine. It is used to return the result of the search operation, containing resources and associated access policies.

Message 13: generated by the Search Engine and sent to the Core Security Handler. It is used to deliver the core token(s) previously verified and the results of the search operation.

Procedure 14: procedure that checks, for each resource, if the attributes contained in the core token(s) satisfy the access policy associated to that resource.

Message 15: generated by the Core Security Handler and sent to the Search Engine. It is used to deliver the result of the previous procedure.

Message 16: generated by the Search Engine and sent to the Application/Enabler asynchronously. It is used to deliver the result of the search operation (available resources).

Procedure 17: executes ranking of resources.

Message 18 (ApplicationInterface): asynchronously sends ranking update to Application/ Enabler.

Message 19 (ApplicationInterface): asynchronously sends message about the end of initial ranking.

Message 20 (ApplicationInterface): synchronously sends message of the end of final ranking.

Message 21 (optional): generated by the Application/Enabler and sent to Application Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from the IoT platform. If the Application/Enabler already has valid foreign token(s), it is not necessary.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 22 (optional) (AAInterface): generated by the Application Security Handler and sent to the foreign AAM in IoT platform. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 23 (optional) (AppSecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 24 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 25 (optional) (PlatformAAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the core AAM before the expiration time indicated within the token itself). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 26 (optional): procedure that, in case it is needed, translates attributes that the Application/Enabler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Application/Enabler already has valid foreign token(s), it is not necessary.

Message 27 (optional): generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Message 28 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It is used to forward the foreign token generated at the previous step.

Message 29 (mandatory): generated by the Application/Enabler and sent to the Bartering and Trading of the foreign platform. It is used to initiate the payment for the previous attained resource.

Message 30 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Payment Processor. It is used to initialize the payment procedure.

Message 31 (mandatory): generated by the Payment Processor and sent to the Bartering and Trading of the foreign platform. It is used provide the transaction data and the redirection URL.

Message 32 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Application/Enabler. It is used to send the redirection URL for the payment process.

Message 33 (mandatory): generated by the Application/Enabler and sent to the Payment Processor. It is used to execute the payment.

Message 34 (mandatory): generated by the Payment Processor and sent to the Application/Enabler. It is used to inform the Application/Enabler about the payment transaction outcome.

Message 35 (mandatory): generated by the Payment Processor and sent to the Bartering and Trading of the foreign platform. It is used to inform the Bartering and Trading about the successful payment execution.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 36 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Payment Processor. It is used to acknowledge the previous received message.

Procedure 37 (mandatory): the Bartering and Trading of the foreign platform updates the resource status and policy.

Message 38 (optional): generated by Bartering and Trading and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Bartering and Trading is already logged in, it is not necessary.

Message 39 (optional): generated by the Security Handler and sent to the home (platform) AAM in which the Bartering and Trading is registered. It is used to authenticate the Bartering and Trading. If the Bartering and Trading is already logged in, it is not necessary.

Message 40 (optional): generated by the home (platform) AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Bartering and Trading is already logged in, it is not necessary.

Message 41 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 42 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Bartering and Trading to demonstrate that it is the real owner of the token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 43 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 44 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 45 (optional): procedure that, in case it is needed, translates attributes that the Bartering and Trading has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Bartering and Trading already has valid core token(s), it is not necessary.

Message 46 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Message 47 (optional): generated by the Security Handler and sent to the Bartering and Trading. It is used to forward the core token generated at the previous step.

Message 48 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Registry in the Core. It is used to synchronize the resource status.

Procedure 49 (mandatory): the Registry of the Core updates the resource status and policy.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 50 (mandatory): generated by the Registry of the Core and sent to the Bartering and Trading of the foreign platform. It is used to inform the Bartering and Trading of the foreign platform regarding the previous transaction outcome.

Message 51 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Application/Enabler. It is used to confirm the successful purchase of the resource.

3.3.3 Forward Auction

Figure 3.3 presents the sequence diagram for forward auctioning. As with the aforementioned Direct Buy scenarios, the payment processing again depends on employing third-party payment systems (like Paypal or Stripe). For a more detailed discussion of this issue please refer to Section 3.4 of this document.

Before describing this scenario in more detail, remember that auctions are generally characterized by a bidding process where interested bidders submit their bids either openly (open-outcry auction) or secretly (sealed bid auction). With open-outcry auctions, two main types have to be distinguished, i.e. ascending (English) auctions where the price increases (usually step-wise) vs. descending (Dutch) auctions with a decreasing price. Similarly, there are two types of sealed bid auctions to be distinguished, i.e. first-price auctions (where the winning bidder actually pays her bid) vs. second-price (Vickrey) auctions where the winning bidder pays the bid of the highest-bidding loser.

While these four auction types seem to be rather different from each other, it is highly interesting to note that – according to the so-called “Revenue Equivalence Theorem” [1] under rather mild conditions the expected revenue achieved does not vary between these different types. Hence, the actual choice of an auction mechanism may depend also on second-order properties like incentive compatibility or revenue variance. This explains why Vickrey auctions are rather typical in the field of network economics as they have the intrinsic property of forcing auction participants to be honest about their true evaluation of the value of the good, because this can be proven to provide them an optimal strategy (cf. for instance the bidding process followed in ebay that follows the same idea).

Having said that, the corresponding message sequence chart depicted in Figure 3.3 starts from the assumption that a platform (producer) has already registered within the symbIoTe Core the resource to be auctioned, and furthermore that the Core Search Engine provides result even if the seeker of the resource does not possess the necessary attributes.

Then, the producer starts by launching a forward auction for one of its resources which will be hosted by the symbIoTe Core. The Application/Enabler (consumer) makes use of the Core Search Engine to find the desired resource. The desired resource is found to be up for sale by means of a forward auction. All interested parties can place a bid with the aim to acquire the resource. After the time, designated for the auction, expires the Core Bartering and Trading will determine the winner. The producer which started the auction will receive a detailed report regarding the auction outcome. The auction initiator will inform the winning party that it accepts its bid and that the payment can be performed. A successful payment transaction will lead to the producer and the symbIoTe Core updating the status and availability of the sold resource. Eventually, the winner will receive a confirmation for the purchase.

© Copyright 2016, the Members of the symbIoTe Consortium

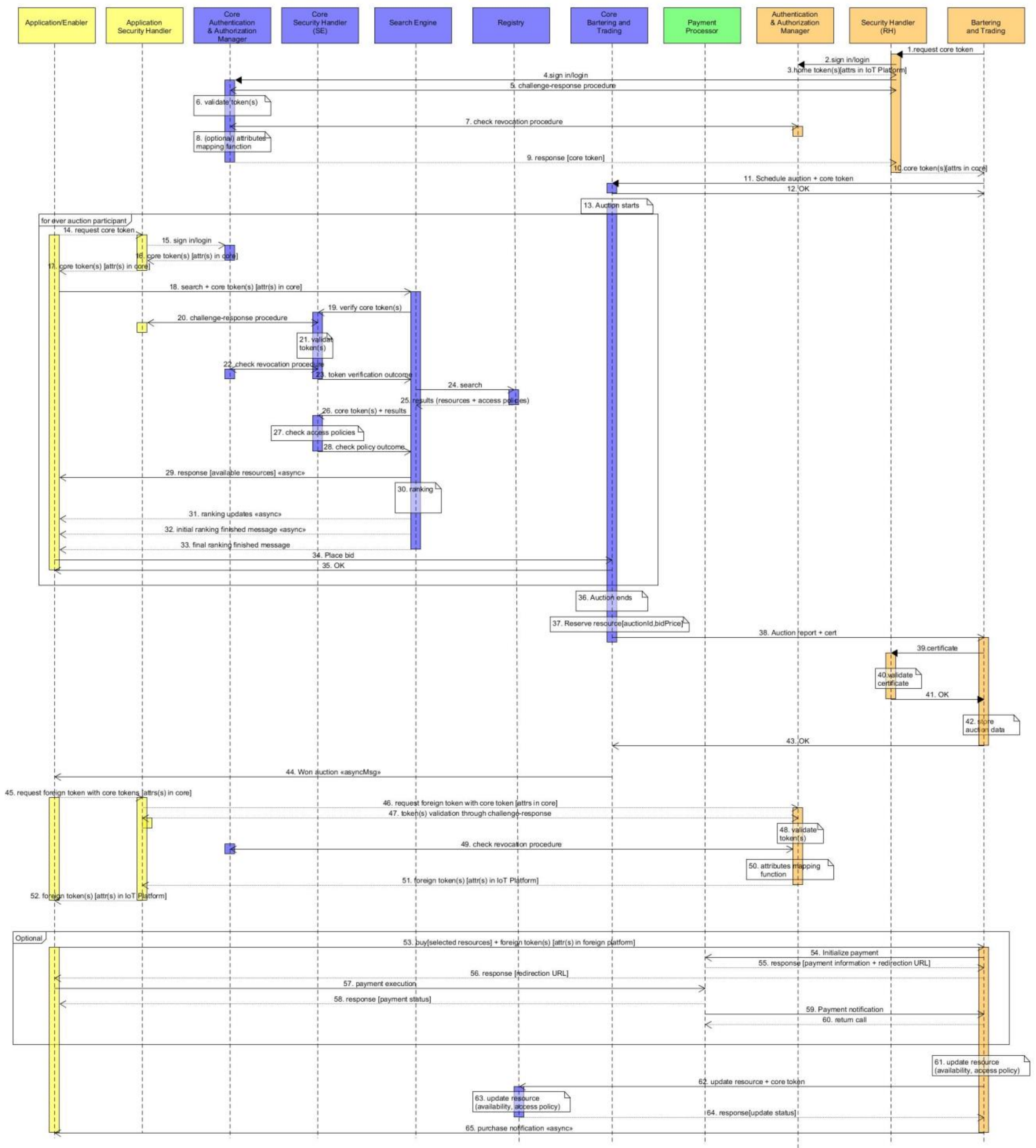

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Security Handler (SE)
Search Engine
Registry
Core Bartering and Trading
Payment Processor
Authentication & Authorization Manager
Security Handler (RH)
Bartering and Trading
1.request core token
2.sign in/login
3.home token(s)[attrs in IoT Platform]
4.sign in/login
5. challenge-response procedure
6. validate token(s)
7. check revocation procedure
8. (optional) attributes mapping function
9. response [core token]
10.core token(s)[attrs in core]
11. Schedule auction + core token
12. OK
13. Auction starts
for ever auction participant
14. request core token
15. sign in/login
16. core token(s) [attr(s) in core]
17. core token(s) [attr(s) in core]
18. search + core token(s) [attr(s) in core]
19. verify core token(s)
20. challenge-response procedure
21. validat token(s)
22. check revocation procedure
23. token verification outcome
24. search
25. results (resources + access policies)
26. core token(s) + results
27. check access policies
28. check policy outcome
29. response [available resources] «async»
30. ranking
31. ranking updates «async»
32. initial ranking finished message «async»
33. final ranking finished message
34. Place bid
35. OK
36. Auction ends
37. Reserve resource[auctionId,bidPrice]
38. Auction report + cert
39.certificate
40.validate certificate
41. OK
42. store auction data
43. OK
44. Won auction «asyncMsg»
45. request foreign token with core tokens [attrs(s) in core]
46. request foreign token with core token [attrs in core]
47. token(s) validation through challenge-response
48. validate token(s)
49. check revocation procedure
50. attributes mapping function
51. foreign token(s) [attr(s) in IoT Platform]
52. foreign token(s) [attr(s) in IoT Platform]
Optional
53. buy[selected resources] + foreign token(s) [attr(s) in foreign platform]
54. Initialize payment
55. response [payment information + redirection URL]
56. response [redirection URL]
57. payment execution
58. response [payment status]
59. Payment notification
60. return call
61. update resource (availability, access policy)
62. update resource + core token
63. update resource (availability, access policy)
64. response[update status]
65. purchase notification «async»

Figure 3.3: Forward Auction

In total, Figure 3.3 comprises the following 65 messages and/or procedures:

Message 1 (optional): generated by Bartering and Trading and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Bartering and Trading is already logged in, it is not necessary.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 2 (optional): generated by the Security Handler and sent to the home (platform) AAM in which the Bartering and Trading is registered. It is used to authenticate the Bartering and Trading. If the Bartering and Trading is already logged in, it is not necessary.

Message 3 (optional): generated by the home (platform) AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Bartering and Trading is already logged in, it is not necessary.

Message 4 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 5 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Bartering and Trading to demonstrate that it is the real owner of the token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 6 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 7 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 8 (optional): procedure that, in case it is needed, translates attributes that the Bartering and Trading has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Bartering and Trading already has valid core token(s), it is not necessary.

Message 9 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Message 10 (optional): generated by the Security Handler and sent to the Bartering and Trading. It is used to forward the core token generated at the previous step.

Message 11 (mandatory): generated by the Bartering and Trading and sent to the Core Bartering and Trading. It is used to initiate and schedule a forward auction.

Message 12 (mandatory): generated by the Core Bartering and Trading and sent to the Bartering and Trading. It is used to acknowledge the previous scheduled forward auction.

Procedure 13 (mandatory): the Core Bartering and Trading starts the forward auction.

Message 14 (optional): generated by the Application/Enabler and sent to the Application Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.

Message 15 (optional) (AppAAInterface): generated by the Application Security Handler and sent to the Core AAM in which the Application/Enabler is registered. It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 16 (optional): generated by the Core AAM in the IoT platform and sent to the Application Security Handler. It is used to provide the home token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.

Message 17 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It is used to deliver the core token(s).

Message 18 (SearchInterface): generated by the Application/Enabler and sent to the Search Engine. It sends a search query and the core token(s) to the Search Engine.

Message 19: generated by the Search Engine and sent to the Core Security Handler. It is used to ask the security handler to verify the complete validity of the token.

Procedure 20: *(AppSecurityInterface):* procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s).

Procedure 21: verification of the time validity, authenticity and integrity of the provided token(s).

Procedure 22: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).

Message 23: generated by the Core Security Handler and sent to the Search Engine. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.

Message 24: generated by the Search Engine and sent to the Registry. It is used to search available resources.

Message 25: generated by the Registry and sent to the Search Engine. It is used to return the result of the search operation, containing resources and associated access policies.

Message 26: generated by the Search Engine and sent to the Core Security Handler. It is used to deliver the core token(s) previously verified and the results of the search operation.

Procedure 27: procedure that checks, for each resource, if the attributes contained in the core token(s) satisfy the access policy associated to that resource.

Message 28: generated by the Core Security Handler and sent to the Search Engine. It is used to deliver the result of the previous procedure.

Message 29: generated by the Search Engine and sent to the Application/Enabler asynchronously. It is used to deliver the result of the search operation (available resources).

Procedure 30: executes ranking of resources.

Message 31 (ApplicationInterface): asynchronously sends ranking update to Application/ Enabler.

Message 32 (ApplicationInterface): asynchronously sends message about the end of initial ranking.

Message 33 (ApplicationInterface): synchronously sends message of the end of final ranking.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 34 (optional): generated by the Application/Enabler and sent to the Core Bartering and Trading. It is used the place a bid.

Message 35 (mandatory): generated by the Core Bartering and Trading and sent to the Application/Enabler. It is used to acknowledge the previously placed bid.

Procedure 36 (mandatory): the Core Bartering and Trading closes the auction.

Procedure 37 (mandatory): the Core Bartering and Trading reserves the resource and calculates the price.

Message 38 (mandatory): generated by Core Bartering and Trading and sent to the Bartering and Trading. It is used to provide information about the auction outcome and the corresponding certificate.

Message 39 (mandatory): generated by the Bartering and Trading and sent to the Security Handler. It is used to verify the validity of the previously received certificate.

Procedure 40 (mandatory): the Security Handler validates the certificate.

Message 41 (mandatory): generated by the Security handler and sent to the Bartering and Trading. It is used to report the outcome of the certificate validation.

Procedure 42 (mandatory): the Bartering and Trading stores the data regarding the auction.
Message 43 (mandatory): generated by the Bartering and Trading and sent to the Core Bartering and Trading. It is used to acknowledge the outcome of the auction.

Message 44 (mandatory): generated by the Core Bartering and Trading and sent to the Application/Enabler. It is used to inform the winner of the auction that the won.

Message 45 (optional): generated by the Application/Enabler and sent to Application Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from IoT platform. If the Application/Enabler already has valid foreign token(s), it is not necessary.

Message 46 (optional) (AAInterface): generated by the Application Security Handler and sent to the foreign AAM in IoT platform. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 47 (optional) (AppSecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 48 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 49 (optional) (PlatformAAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the core AAM before the expiration time indicated within the token itself). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Procedure 50 (optional): procedure that, in case it is needed, translates attributes that the Application/Enabler has in the home IoT platform in a new set of attributes that it has in the

© Copyright 2016, the Members of the symbIoTe Consortium

core layer. If attributes are the same or the Application/Enabler already has valid foreign token(s), it is not necessary.

Message 51 (optional): generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.

Message 52 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It is used to forward the foreign token generated at the previous step.

Message 53 (mandatory): generated by the Application/Enabler and sent to the Bartering and Trading of the foreign platform. It is used to initiate the payment for the previous attained resource.

Message 54 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Payment Processor. It is used to initialize the payment procedure.

Message 55 (mandatory): generated by the Payment Processor and sent to the Bartering and Trading of the foreign platform. It is used provide the transaction data and the redirection URL.

Message 56 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Application/Enabler. It is used to send the redirection URL for the payment process.

Message 57 (mandatory): generated by the Application/Enabler and sent to the Payment Processor. It is used to execute the payment.

Message 58 (mandatory): generated by the Payment Processor and sent to the Application/Enabler. It is used to inform the Application/Enabler about the payment transaction outcome.

Message 59 (mandatory): generated by the Payment Processor and sent to the Bartering and Trading of the foreign platform. It is used to inform the Bartering and Trading about the successful payment execution.

Message 60 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Payment Processor. It is used to acknowledge the previous received message.

Procedure 61 (mandatory): the Bartering and Trading of the foreign platform updates the resource status and policy.

Message 62 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Registry in the Core. It is used to synchronize the resource status.

Procedure 63 (mandatory): the Registry of the Core updates the resource status and policy.

Message 64 (mandatory): generated by the Registry of the Core and sent to the Bartering and Trading of the foreign platform. It is used to inform the Bartering and Trading of the foreign platform regarding the previous transaction outcome.

Message 65 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Application/Enabler. It is used to confirm the successful purchase of the resource.

© Copyright 2016, the Members of the symbIoTe Consortium

3.3.4 Reverse Auction

Reverse auctions differ from forward auctions by the fact that there is no good to be auctioned between different interested consumers, but rather the other way round, a consumer publishes a request for a certain resource, and different producers provide corresponding offers between which the consumer has to decide.

The corresponding message sequence diagram is depicted in Figure 3.4 and starts from the prerequisites that one or more platforms (producers) have already registered, within the symbIoTe Core, their resources, and that the Core Search Engine provides result only to producers which have already registered within symbIoTe the desired resources.

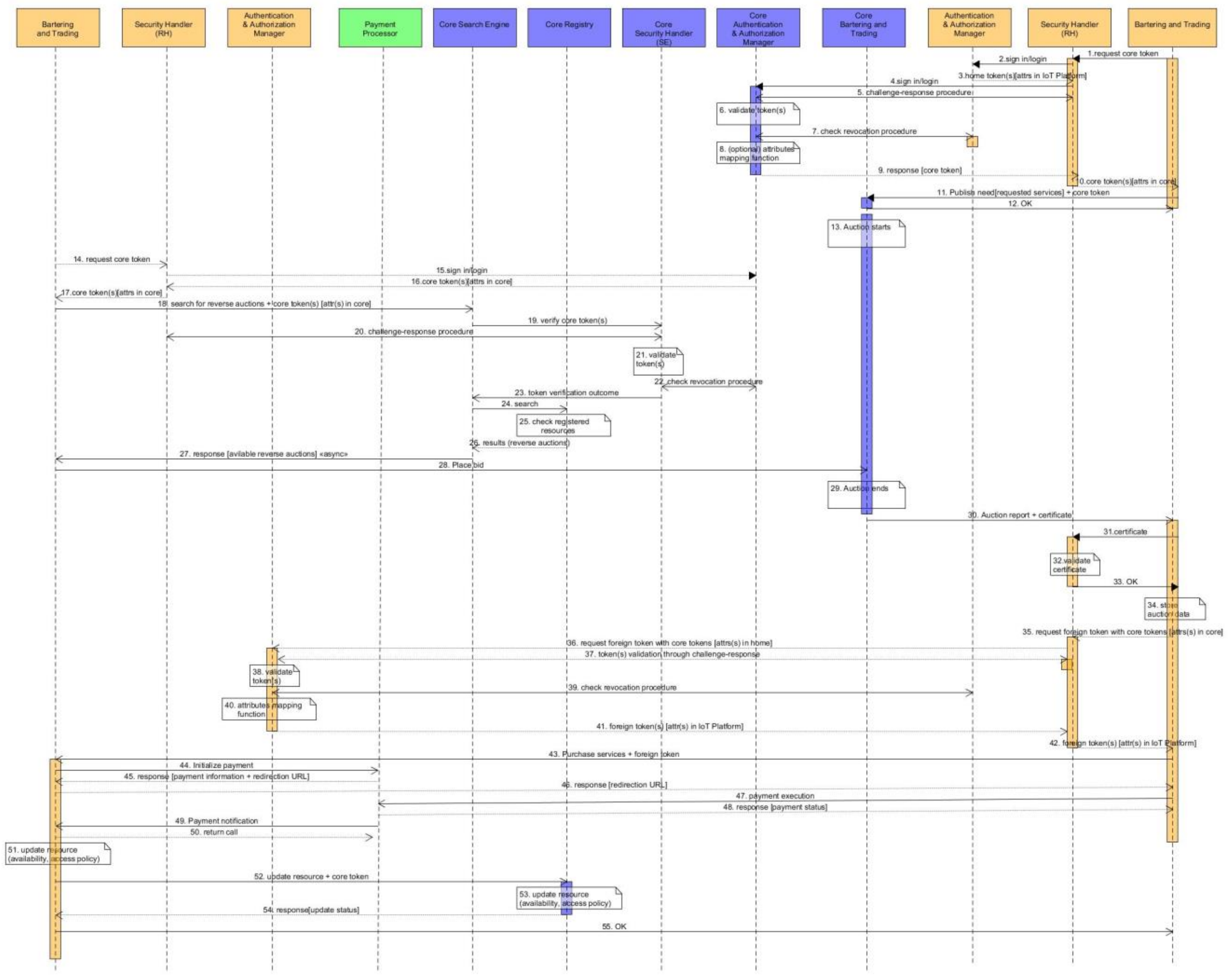

Bartering and Trading
Security Handler (RH)
Authentication & Authorization Manager
Payment Processor
Core Search Engine
Core Registry
Core Security Handler (SE)
Core Authentication & Authorization Manager
Core Bartering and Trading
Authentication & Authorization Manager
Security Handler (RH)
Bartering and Trading
1.request core token
2.sign in/login
3.home token(s)[attrs in IoT Platform]
4.sign in/login
5. challenge-response procedure
6. validate token(s)
7. check revocation procedure
8. (optional) attributes mapping function
9. response [core token]
10.core token(s)[attrs in core]
11. Publish need[requested services] + core token
12. OK
13. Auction starts
14. request core token
15.sign in/login
16.core token(s)[attrs in core]
17.core token(s)[attrs in core]
18. search for reverse auctions + core token(s) [attr(s) in core]
19. verify core token(s)
20. challenge-response procedure
21. validate token(s)
22. check revocation procedure
23. token verification outcome
24. search
25. check registered resources
26. results (reverse auctions)
27. response [avilable reverse auctions] «async»
28. Place bid
29. Auction ends
30. Auction report + certificate
31.certificate
32.validate certificate
33. OK
34. store auction data
35. request foreign token with core tokens [attrs(s) in core]
36. request foreign token with core tokens [attrs(s) in home]
37. token(s) validation through challenge-response
38. validate token(s)
39. check revocation procedure
40. attributes mapping function
41. foreign token(s) [attr(s) in IoT Platform]
42. foreign token(s) [attr(s) in IoT Platform]
43. Purchase services + foreign token
44. Initialize payment
45. response [payment information + redirection URL]
46. response [redirection URL]
47. payment execution
48. response [payment status]
49. Payment notification
50. return call
51. update resource (availability, access policy)
52. update resource + core token
53. update resource (availability, access policy)
54. response[update status]
55. OK

Figure 3.4: Reverse Auction

Then, a consumer launches a reverse auction by defining a request (i.e. a resource s/he wishes to buy) which will be hosted by the symbIoTe Core. A producer makes use of the Core Search Engine to find potential buyers. Every producer who can fulfill the request may submit a bid. After the auction deadline has expired, the Core Bartering and Trading will determine the winner. The consumer which started the auction will receive a detailed

© Copyright 2016, the Members of the symbIoTe Consortium

report regarding the auction outcome. The auction initiator will authenticate itself within winner's IoT platform and signal its intent to buy the desired resource. A successful payment transaction will lead to the producer and the symbIoTe Core updating the status and availability of the sold resource. Eventually, the winning producer will receive a confirmation for the purchase.

Hence, Figure 3.4 includes the following 55 messages and/or procedures:

Message 1 (optional): generated by Bartering and Trading and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Bartering and Trading is already logged in, it is not necessary.

Message 2 (optional): generated by the Security Handler and sent to the home (platform) AAM in which the Bartering and Trading is registered. It is used to authenticate the Bartering and Trading. If the Bartering and Trading is already logged in, it is not necessary.

Message 3 (optional): generated by the home (platform) AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Bartering and Trading is already logged in, it is not necessary.

Message 4 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 5 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Bartering and Trading to demonstrate that it is the real owner of the token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 6 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 7 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 8 (optional): procedure that, in case it is needed, translates attributes that the Bartering and Trading has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Bartering and Trading already has valid core token(s), it is not necessary.

Message 9 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Message 10 (optional): generated by the Security Handler and sent to the Bartering and Trading. It is used to forward the core token generated at the previous step.

Message 11 (mandatory): generated by the Bartering and Trading and sent to the Core Bartering and Trading. It is used to initiate and schedule a reverse auction.

Message 12 (mandatory): generated by the Core Bartering and Trading and sent to the Bartering and Trading. It is used to acknowledge the previous scheduled reverse auction.

© Copyright 2016, the Members of the symbIoTe Consortium

Procedure 13 (mandatory): the Core Bartering and Trading starts the reverse auction.

Message 14 (optional): generated by the Bartering and Trading and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Bartering and Trading is already logged in, it is not necessary.

Message 15 (optional): generated by the Security Handler and sent to the Core AAM in which the platform is registered. It is used to authenticate the Bartering and Trading. If the Bartering and Trading is already logged in, it is not necessary.

Message 16 (optional): generated by the Core AAM and sent to the Security Handler. It is used to provide the core token(s) with attributes included. If the Bartering and Trading is already logged in, it is not necessary.

Message 17 (optional): generated by the Security Handler and sent to the Bartering and Trading. It is used to deliver the core token(s).

Message 18 (SearchInterface): generated by the Bartering and Trading and sent to the Search Engine. It sends a search query and the core token(s) to the Search Engine.

Message 19: generated by the Search Engine and sent to the Core Security Handler. It is used to ask to the security handler to verify the complete validity of the token.

Procedure 20: procedure that allows the Security Handler that is acting on behalf of the Bartering and Trading to demonstrate that it is the real owner of the token(s).

Procedure 21: verification of the time validity, authenticity and integrity of the provided token(s).

Procedure 22: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).

Message 23: generated by the Core Security Handler and sent to the Search Engine. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.

Message 24: generated by the Search Engine and sent to the Registry. It is used to search available reverse auctions.

Procedure 25: the Core Registry compares the resources registered by the querying platform with resource needs present in the ongoing or scheduled reverse auctions.

Message 26: generated by the Core Registry and sent to the Core Search Engine. It is used to return the result of the search operation, containing reverse auctions which fulfil the check in procedure 25.

Message 27: generated by the Search Engine and sent to the Bartering and Trading asynchronously. It is used to deliver the result of the search operation (reverse auctions).

Message 28: generated by the Bartering and Trading and sent to the Core Bartering and Trading. It is used to place a bid for a reverse auction.

Procedure 29: the Core Bartering and Trading closes the reverse auction.

Message 30: generated by the Core Bartering and Trading and sent to the Bartering and trading which initiated the reverse auction. It is used to report the outcome of the reverse auction and the confirmation certificate.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 31: generated by the Bartering and Trading and sent to the Security Handler. It is used to send the previously obtained certificate for validation.

Procedure 32: the Security Handler validates the previously obtained certificate.

Message 33: generated by the Security Handler and sent to the Bartering and Trading. It is used to report on the certificate validation outcome.

Procedure 34: the Bartering and Trading stores the reverse auction data.

Message 35 (optional): generated by Bartering and Trading and sent to the Security Handler. It is used to trigger the recovery of the foreign token(s). If the Bartering and Trading is already logged in, it is not necessary.

Message 36 (optional): generated by the Security Handler and sent to the foreign (platform) AAM which has won the reverse auction. It is used to authenticate the Bartering and Trading. If the Bartering and Trading is already logged in, it is not necessary.

Procedure 37 (optional): procedure that allows the Security Handler that is acting on behalf of the Bartering and Trading to demonstrate that it is the real owner of the token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 38 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 39 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Bartering and Trading already has valid core token(s), it is not necessary.

Procedure 40 (optional): procedure that, in case it is needed, translates attributes that the Bartering and Trading has in the home IoT platform in a new set of attributes that it has in the foreign platform. If attributes are the same or the Bartering and Trading already has valid foreign token(s), it is not necessary.

Message 41 (optional): generated by the foreign platform AAM and sent to the Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Bartering and Trading already has valid foreign token(s), it is not necessary.

Message 42 (optional): generated by the Security Handler and sent to the Bartering and Trading. It is used to forward the foreign token generated in the previous step.

Message 43 (mandatory): generated by the Bartering and Trading and sent to the Bartering and Trading of the foreign platform. It is used to initiate the payment for the previous requested (needed) resource.

Message 44 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Payment Processor. It is used to initialize the payment procedure.

Message 45 (mandatory): generated by the Payment Processor and sent to the Bartering and Trading of the foreign platform. It is used provide the transaction data and the redirection URL.

Message 46 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Bartering and Trading which initiated the reverse auction. It is used to send the redirection URL for the payment process.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 47 (mandatory): generated by the Bartering and Trading and sent to the Payment Processor. It is used to execute the payment.

Message 48 (mandatory): generated by the Payment Processor and sent to the Bartering and Trading. It is used to inform the Bartering and Trading about the payment transaction outcome.

Message 49 (mandatory): generated by the Payment Processor and sent to the Bartering and Trading of the foreign platform. It is used to inform the Bartering and Trading about the successful payment execution.

Message 50 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Payment Processor. It is used to acknowledge the previous received message.

Procedure 51 (mandatory): the Bartering and Trading of the foreign platform updates the resource status and policy.

Message 52 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Registry in the Core. It is used to synchronize the resource status.

Procedure 53 (mandatory): the Registry of the Core updates the resource status and policy.

Message 54 (mandatory): generated by the Core Registry and sent to the Bartering and Trading of the foreign platform. It is used to inform the Bartering and Trading of the foreign platform regarding the previous transaction outcome.

Message 55 (mandatory): generated by the Bartering and Trading of the foreign platform and sent to the Bartering and Trading which initiated the reverse auction. It is used to confirm the successful purchase of the resource.

3.3.5 Voucher-based Bartering

Finally, Figure 3.5 depicted below represents the basic bartering scenario between two distinct platforms. Suppose again that a platform (prosumer) has already registered, within the symbIoTe Core, the resources s/he is willing to trade, using a voucher description. When defining the desired resources on the voucher, the the prosumer can determine suitable intervals for the parameters describing the resource.

To start with, two distinct prosumers (IoT platforms) send to the Core Bartering and Trading a description of the resources which they are willing to barter (own ones) and the resources (needed ones) they are looking for. Upon receiving the information from the platforms the Core Bartering and Trading can start matching the desired resources with the offered resources. As perfectly matching resource descriptions might be relatively rare occasions, various way of relaxing the matching criterion can be implemented, for instance a matching ratio of higher than 90% with respect to the resource SLAs. If such a ratio can be determined, the Core Bartering and Trading automatically sends the corresponding vouchers, which hence correspond mostly to the needs of the platforms, to the platforms. If, however, the match is below or equal to 90%, the Core Bartering and Trading will only inform the participating platforms about the potential bartering opportunity, and only after both parties have explicitly stated their interest, the Core Bartering and Trading will send out the corresponding vouchers. The last step after a successful exchange is the update of the corresponding availability and policy of the resources.

© Copyright 2016, the Members of the symbIoTe Consortium

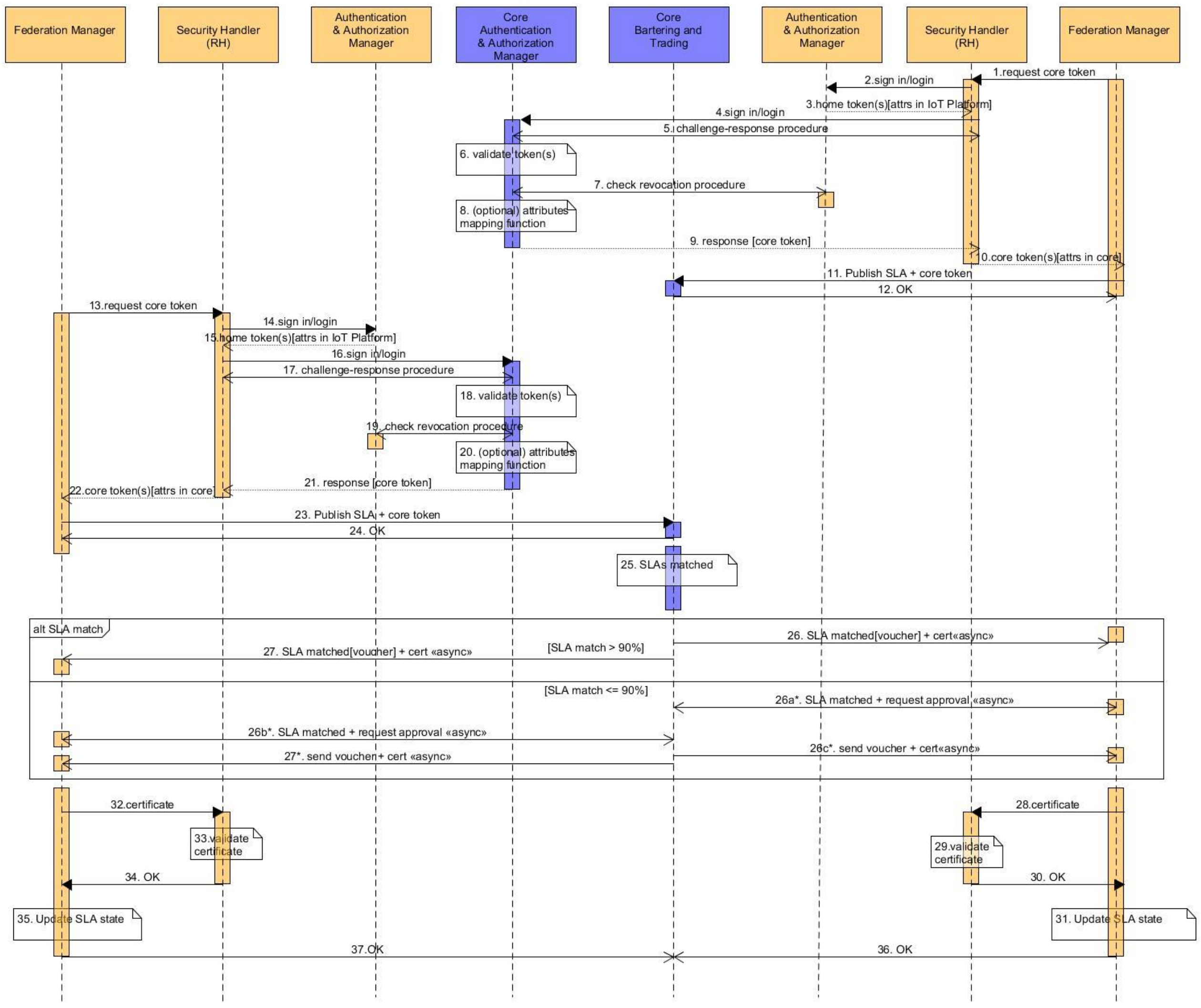

Federation Manager
Security Handler (RH)
Authentication & Authorization Manager
Core Authentication & Authorization Manager
Core Bartering and Trading
Authentication & Authorization Manager
Security Handler (RH)
Federation Manager
1.request core token
2.sign in/login
3.home token(s)[attrs in IoT Platform]
4.sign in/login
5. challenge-response procedure
6. validate token(s)
7. check revocation procedure
8. (optional) attributes mapping function
9. response [core token]
10.core token(s)[attrs in core]
11. Publish SLA + core token
12. OK
13.request core token
14.sign in/login
15.home token(s)[attrs in IoT Platform]
16.sign in/login
17. challenge-response procedure
18. validate token(s)
19. check revocation procedure
20. (optional) attributes mapping function
21. response [core token]
22.core token(s)[attrs in core]
23. Publish SLA + core token
24. OK
25. SLAs matched
alt SLA match
26. SLA matched[voucher] + cert«async»
27. SLA matched[voucher] + cert «async»
[SLA match > 90%]
[SLA match <= 90%]
26a*. SLA matched + request approval «async»
26b*. SLA matched + request approval «async»
26c*. send voucher + cert«async»
27*. send voucher + cert «async»
32.certificate
33.validate certificate
34. OK
35. Update SLA state
28.certificate
29.validate certificate
30. OK
31. Update SLA state
37.OK
36. OK

Figure 3.5: Voucher-based Bartering between two Platforms A (left) and B (right)

Figure 3.5 includes a total of 37 (alternative path: 39) messages and/or procedures. Note that this scenario depicts two alternative paths: while messages 26 and 27 describe the case of automatic bartering due to an SLA match above 90%, messages 26a*, 26b*, 26c* and 27* refer to the case that, before vouchers are sent out, both participants have to explicitly agree as the matching ratio between their SLAs is below 90%. Hence, the alternative messages with an asterisk * are supposed to be exchanged between platforms and Core services if SLAs are accepted by both Federation Managers involved in the bartering process, and in this sense are characterized as not mandatory but rather optional messages.

<u>*Message 1 (optional):*</u> generated by the Federation Manager and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, it is not necessary.

<u>*Message 2 (optional):*</u> generated by the Security Handler and sent to the home (platform) AAM in which the Federation Manager is registered. It is used to authenticate the Federation Manager. If the Federation Manager is already logged in, it is not necessary.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 3 (optional): generated by the home (platform) AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Federation Manager is already logged in, it is not necessary.

Message 4 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Federation Manager already has valid core token(s), it is not necessary.

Procedure 5 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Federation Manager to demonstrate that it is the real owner of the token(s). If the Federation Manager already has valid core token(s), it is not necessary.

Procedure 6 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Federation Manager already has valid core token(s), it is not necessary.

Procedure 7 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Federation Manager already has valid core token(s), it is not necessary.

Procedure 8 (optional): procedure that, in case it is needed, translates attributes that the Federation Manager has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Federation Manager already has valid core token(s), it is not necessary.

Message 9 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Federation Manager already has valid core token(s), it is not necessary.

Message 10 (optional): generated by the Security Handler and sent to the Federation Manager. It is used to forward the core token generated at the previous step.

Message 11 (mandatory): generated by the Federation Manager and sent to the CoreBartering and Trading. It is used to deliver the core token(s) with the new attribute(s), the offered SLA and the desired SLA.

Message 12 (mandatory): generated by the Core Bartering and Trading and sent to the Federation Manager. It is an acknowledgement of the received SLAs.

Message 13 (optional): generated by the Federation Manager and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, it is not necessary.

Message 14 (optional): generated by the Security Handler and sent to the home (platform) AAM in which the Federation Manager is registered. It is used to authenticate the Federation Manager. If the Federation Manager is already logged in, it is not necessary.

Message 15 (optional): generated by the home (platform) AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Federation Manager is already logged in, it is not necessary.

Message 16 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Federation Manager already has valid core token(s), it is not necessary.

© Copyright 2016, the Members of the symbIoTe Consortium

Procedure 17 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Federation Manager to demonstrate that it is the real owner of the token(s). If the Federation Manager already has valid core token(s), it is not necessary.

Procedure 18 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Federation Manager already has valid core token(s), it is not necessary.

Procedure 19 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Federation Manager already has valid core token(s), it is not necessary.

Procedure 20 (optional): procedure that, in case it is needed, translates attributes that the Federation Manager has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Federation Manager already has valid core token(s), it is not necessary.

Message 21 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Federation Manager already has valid core token(s), it is not necessary.

Message 22 (optional): generated by the Security Handler and sent to the Federation Manager. It is used to forward the core token generated at the previous step.

Message 23 (mandatory): generated by the Federation Manager and sent to the CoreBartering and Trading. It is used to deliver the core token(s) with the new attribute(s), the offered SLA and the desired SLA.

Message 24 (mandatory): generated by the Core Bartering and Trading and sent to the Federation Manager. It is an acknowledgement of the received SLAs.

Message 25 (mandatory): the Core Bartering and Trading tries to find the best match between desired and offered SLA.

Message 26 (mandatory): generated by the Core Bartering and Trading and sent to the Federation Manager of platform B. It is used to send the matching desired SLA and the certificate (confirmation).

Message 27 (mandatory): generated by the Core Bartering and Trading and sent to the Federation Manager of platform A. It is used to send the matching desired SLA and the certificate (confirmation).

Message 26a (mandatory):* generated by the Core Bartering and Trading and sent to the Federation Manager of platform B. It is used to send the best matching desired SLA and the request for confirmation, and needs explicit approval by platform B.

Message 26b (mandatory):* generated by the Core Bartering and Trading and sent to the Federation Manager of platform A. It is used to send the best matching desired SLA and the request for confirmation, and needs explicit approval by platform A.

Message 26c (optional):* generated by the Core Bartering and Trading and sent to the Federation Manager of platform B. It is used to send the matching desired SLA and the certificate (confirmation). If the other party does not accept the swapping, this message will not be sent.

© Copyright 2016, the Members of the symbIoTe Consortium

Message 27 (optional):* generated by the Core Bartering and Trading and sent to the Federation Manager of platform A. It is used to send the matching desired SLA and the certificate (confirmation). If the other party does not accept the swapping, this message will not be sent.

Message 28 (mandatory): generated by the Federation Manager and sent to the Security Handler. It is used to send the obtained certificate (confirmation) which validates the bartering transaction.

Message 29 (mandatory): the Security Handler validates the received certificate.

Message 30 (mandatory): generated by the Security Handler and sent to the Federation Manager. It is used to provide a response regarding the validity check of the certificate.

Message 31 (mandatory): the Federation Manager updates the quota regarding the involved SLA in the bartering transaction.

Message 32 (mandatory): generated by the Federation Manager and sent to the Security Handler. It is used to send the obtained certificate (confirmation) which validates the bartering transaction.

Message 33 (mandatory): the Security Handler validates the received certificate.

Message 34 (mandatory): generated by the Security Handler and sent to the Federation Manager. It is used to provide a response regarding the validity check of the certificate.

Message 35 (mandatory): the Federation Manager updates the quota regarding the involved SLA in the bartering transaction.

Message 36 (mandatory): generated by the Federation Manager and sent to the Core Bartering and Trading. It is used to provide an acknowledgment regarding the bartering transaction.

Message 37 (mandatory): generated by the Federation Manager and sent to the Core Bartering and Trading. It is used to provide an acknowledgment regarding the bartering transaction.

3.4 Payment Systems

A payment system is any system used to settle financial transactions through the transfer of monetary value, and includes the institutions, instruments, people, rules, procedures, standards, and technologies that make such an exchange possible [10][11].

The main characteristic of a payment system is that they use cash-substitutes. No bank notes are involved in the transactions. Some payment systems might include credit mechanisms. Payment systems are used in lieu of tendering cash in domestic and international transactions and consist of a major service provided by banks and other financial institutions.

An e-commerce payment system facilitates the acceptance of electronic payment for online transactions. They have become increasingly popular due to the widespread use of the internet-based shopping and banking. A large number of alternative electronic payment systems have emerged and more will be available soon. It has been found out that the more types of payment systems are used, the higher percentage of selling a website will have [15], so it makes sense to integrate more than one payment system for

© Copyright 2016, the Members of the symbIoTe Consortium

monetary transactions. Below we comment some details about the most important existing payment systems.

Credit cards have become one of the most common and known form of payment for e-commerce transactions. In North America almost 90% of online retail transactions were made with this payment type [12]. Online merchants have to comply with stringent rules stipulated by the credit and debit card issuers (Visa and MasterCard) this means that merchants must have security protocol and procedures in place to ensure transactions are more secure. This can also include having a certificate from an authorized certification authority (CA) who provides PKI(Public-Key infrastructure) for securing credit and debit card transactions [13].

PayPal: fees are charged to sellers/merchants for each transaction by the PayPal [16]. PayPal payment system can be integrated with other shopping cart systems, which enables individual websites, retail and online shopping centers/markets or point of sales, to accept payments on their own. Any individual sending/receiving personal payments can also use PayPal. Within PayPal, a user is able to configure how the funds have to be charged for a transaction: credit card, debit card or bank account. PayPal does offer its own Buyer Protection program, which safeguards users in case that something goes wrong, for example if an item doesn't arrive or doesn't match its description [17].

Bitcoin Payment System: Bitcoin is a digital cryptocurrency created in 2009 by an unknown person using the alias Satoshi Nakamoto. The system is peer-to-peer and transactions take place between users directly, without the presence of Trusted Third Parties (e.g. banks). Transactions are verified by network nodes and recorded in a public distributed ledger called *blockchain.* It used to have no transaction fees, but right now it changed to “Cost very little”. Bitcoins are not tied to any country and they are not subject to regulation. There is even no need for the Bitcoin holders to provide their real names. People compete to “mine” bitcoins using computers to solve complex math puzzles. Bitcoins are stored in a “digital wallet,” which exists either in the cloud or on a user’s computer. The wallet is a kind of virtual bank account that allows users to send or receive bitcoins, pay for goods or save their money [18][19].

Google Pay (aka, Google Wallet [20], upgraded into Android Pay [21]): is used by Google based services, products, and supported by many retail shopping centers/markets and point of sales. Google Wallet allows users to send/receive money to/from other users via their Android or iOS based phone. Android Pay is used by buyers for purchasing, and is available mostly on Google Android OS based phones. Google charges sellers/merchants around 30% fee for each payment that a buyer pays [22]. It started running in the U.K. on 18th May 2016, when Google’s mobile payment service officially went online. Android phones with version 4.4 (KitKat) or newer include an NFC chip inside are compatible with Android Pay, provided that banks and card providers support the service. The system uses tokenization, which processes transactions via individual random account numbers, rather than the actual credit or debit card account number [23].

Apple Pay: is used in iTunes store, in Apple Stores, Apple based services and products, and it is supported by many retail shopping centers/markets. Apple charges around 15% + $0.15 for each payment transaction, before giving the rest of the amount to seller. Banks take-away around 2% fee from Apple's portion for each transaction. It is possible to use an iPhone, iPad or Apple Watch within apps when you see Apple Pay as a payment option. Online shopping support was launched in September 2016. It runs in a customer’s Safari web browser, like a regular online payment page [22][24].

© Copyright 2016, the Members of the symbIoTe Consortium

Samsung Pay: is used by Samsung-based services, products, and is supported by many retail shopping centers/markets. When Samsung Pay is used via Samsung smartphone devices, then neither the buyer nor the seller are charged. Also, it uses tokenization for individual transactions. Online payments were already possible through Samsung Pay, but only in South Korea, the company's native country. In early 2017, Samsung Pay users will be able to make online payments from their phones, tablets, or computers, and use express checkouts that automatically fill in payment data based on information saved in Samsung Pay wallets [25].

Some e-commerce solutions exist for small retailers in order to go online to sell their products. The most prominent ones are:

- WooCommerce: WordPress users can create their own e-commerce shop
- 2CheckOut: payment processor which is a combination of a merchant account and payment gateway. By registering and verifying the account it is possible to start accepting credit card and PayPal payments
- Stripe: mainly used by developers due to the robust API
- ACH Payments: one of the top payment system primarily used for person-to-person
- Authorize.Net: exists since 1996, most widely used payment gateway on internet
- Amazon Payments: safe and easy method to receive money [14].

It is important to note that, within symbIoTe, no payment system will be explicitly involved. This decision has been taken based on information requested from platform owners and use cases and a subsequent long discussion. Here, while an interest has been identified to use a payment system and it would be a nice feature to have, especially from a use cases perspective, however no payment systems were identified in the existing platform within symbIoTe. symbIoTe has to provide interoperability between platforms. In any case, in [26] a protocol is presented for payments across payment systems. It enables secure transfers between ledgers and allows anyone with an account on two ledgers to create a connection between them. In [27], the authors comment that there is no need to create a protocolfor several ledger systems, since one unified ledger for the whole world like Bitcoin already exists. If there is some need in the future of a real implementation supporting payment systems interoperability, both solutions should be studied carefully and pros and cons must be taken into account, in relevance with the requirements of symbIoTe.

3.5 Conclusions for symbIoTe Architecture

symbIoTe's architecture has to be sufficiently versatile to be able to support different platforms, domains and technologies. As such, a system that can cope with such a rich environment will be complex by nature. As a module of the symbIoTe architecture, Bartering and Trading might not necessarily be a core module needed to make the system function, but it is very important for whoever will use the system in the future. It allows users to buy, sell and trade access to their resources, making the system very dynamic. This is of great interest to prosumers, which are envisioned to be symbIoTe's main users.

The Bartering and Trading Manager (B&T Manager) module will make three different systems available: vouchers, direct payments and auctions. To make them work correctly,

© Copyright 2016, the Members of the symbIoTe Consortium

the Bartering & Trading Manager requires the cooperation of several other symbIoTe modules. One of these is the Registration Handler, which stores rules regarding bids and usage of a resource, configured through B&T Manager. The Resource Access Proxy (RAP) checks with the B&T Manager if a user has quota and can pay for the access for a given resource. If these conditions are not satisfied, the RAP will not grant access. The Federation Manager monitors SLA agreements and checks if any violations occur. If that happens, the B&T Manager can take certain actions (reward/punish the user/platform/ application) to compensate. The B&T Manager can also make use of third party payment processing systems to handle monetary payments between users. Additionally, the B&T Manager needs to validate incoming certificates with the Security Handler. Only after the successful completion of this process, can the B&T Manager allow access to certain resources.

There are certain procedures, such as third party payments and forward auctions, that need to be handled centrally. As such, they will be processed by the Core Bartering & Trading Manager. After these procedures are executed, the Core B&T Manager must communicate with the B&T Manager, stating any update to the resources, so that these can be updated in the Registry.

The B&T module might not be of interest to every use case, but it is very important in the commercial context of symbIoTe. As such, it is important that its implementation is intelligent and contained, since its performance can affect the users’ satisfaction within symbIoTe’s ecosystem.

© Copyright 2016, the Members of the symbIoTe Consortium

4 Security and Access Scopes

In the following chapter, we discuss access scopes to resources in symbIoTe with a strong focus on security. Section 4.1 summarises the efforts that were made in task T3.2. Section 4.2 is devoted to authentication and authorization in symbIoTe. Subsection 4.2.1 presents attribute-based access control (ABAC) as a method of authorization in a network of federated IoT platforms. Subsection 4.2.2 shows a review of the available implementations of ABAC, namely OAuth2.0 and two solutions that rely on authorization tokens, one that relies on Macaroons and another that is based on JSON Web Tokens (JWT). In Section 4.3, we present the proposed security architecture with sequence diagrams showing the steps of obtaining access to resources in different scenarios. In Section 4.4, two types of authorization tokens, i.e. Macaroons and JSON Web Tokens are compared in terms of security related requirements that were defined in Deliverable D1.2 and difficulties in implementation. Section 4.5 describes interfaces and services for security components, which are Core Authentication and Authorization Manager (Core AAM), Platform Authentication and Authorization Manager (Platform AAM) and Security Handler (SH). Section 4.6 describes a preliminary proof-of-concept that has been developed to show interactions between the components described in 4.2 and 4.3. Finally, in Section 4.7 we describe other security related issues that might be helpful for further development of system architecture. Threat analysis is carried out and anomaly detection methods that can be applied to overcome security threats which are described.

4.1 Overview of Work in T3.2

Task T3.2 concentrates on all security related aspects in symbIoTe. Its objective is to implement authentication and authorization mechanisms, provide privacy and data anonymization and finally – to propose anomaly detection methods. Following the security requirements defined in D1.2, we designed security architecture and protocols that should be incorporated into the symbIoTe system to provide user, application, enabler (i.e. entity) and platform authentication and authorization.

First, attribute-based access control method of authorization was decided to be the most suitable for symbIoTe, because it originates from distributed computing systems. Other authorization methods like role-based access control (RBAC) or group-based access control (GBAC) are not flexible enough to be applied in a network of federated IoT platforms. The details of access control methods are given in Section 4.2.1.

Second, the details of operations that ensure mutual authentication (an entity vs. a platform) and authorization are presented with the aid of sequence diagrams. A related work prior to the agreement of sequence diagrams was the final choice of symbIoTe requirements that refer to security (within T1.3), the final decision about the location of architecture components related to security and their functionality (within T1.4). These are listed below.

4.1.1 Security Requirements

After the symbIoTe project was launched, the definition of system requirements was started, containing a subset of security requirements. The discussion was handled between April 2016 and September 2016, when Deliverable D1.2 was issued [7]. The requirements were derived from a subset of IoT use cases of the existing IoT platforms

© Copyright 2016, the Members of the symbIoTe Consortium

and general security knowledge about distributed computing systems, together with the identification of threats. Taking over the control of sensors and actuators in a network of federated IoT platforms can cause several unexpected operations like opening doors in a smart home environment or launching the evacuation mechanism in smart stadium. Some of the security requirements were defined to prevent such threats.

First of all, the system must provide authentication and authorization (granting access rights) of its entities, which are enablers, platforms, applications and users. Not only applications must be authenticated by IoT platforms, but also IoT platforms must be authenticated by applications to avoid impersonating the whole platforms. Secondly, a multifactor authentication must be supported that refers to two independent characteristics that identify the user (e.g. RF fingerprint indicating the location and password indicating the users' knowledge). Moreover, validation of input data based on sanitization mechanisms, which remove potentially harmful input data is a necessity. Another important security mechanism that must be implemented is the access control defined through access policies. Referring to security requirements, access control must be handled through Attribute-Based Access Control (ABAC) schemes. The latter is discussed in Section 4.2.

It is important that user authentication is carried out independently of authorization. In other words, verification of a user's identity by a foreign IoT platform should be performed regardless of granting or not granting him access rights to the resources stored within this IoT platform. Especially desirable is the capability of getting access to resources in one IoT platform while being a registered user in another IoT platform. To overcome this problem the system must offer entity identification mechanisms e.g. by adoption of tokens. From a practical point of view it is necessary to let users, applications and enablers decide how their data is processed and whom they can delegate their authorities by attenuating them. For example, an owner of a Smart Home platform can let his guests open the entrance gate to his residence between 7 p.m. and 9 p.m. in a particular day. All requirements are listed in Table 2 of Deliverable D1.2, which is recollected in Section 4.4 [7].

4.1.2 Summary of Activities

A deep analysis of system requirements and definition of symbIoTe architecture were the starting point for designing the security architecture. Combining this with our knowledge on sensor networks, distributed and cloud computing helped us define functional units and their responsibilities in Application Domain and Cloud Domain. A detailed analysis resulted in the security architecture concept that is presented in Section 4.3, with Level1 sequence diagrams showing the exchange of messages between security components. Moreover, a Demo that is a proof-of-concept of our design was built by us and presented in the Technical Review of symbIoTe project that took place in Vienna in October 2016. In the meantime, we tried to identify a method of authentication and authorization that would match symbIoTe's requirements and architecture.

Initially, the Open Authorization Framework (OAuth 2.0), defined in RFC 6749 was considered for authorization [3]. OAuth 2.0 is an authorization framework for web applications, desktop applications, mobile phones, and living room devices. In the introduction of RFC 6749, it was mentioned: "It enables a third party application to obtain limited access to an HTTP service either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service or by allowing

© Copyright 2016, the Members of the symbIoTe Consortium

the third-party application to obtain access on its own behalf." The token format is out-of-scope of OAuth 2.0, however some implementations use JSON Web Tokens. OAuth 2.0 can also be implemented with Proof-of-Possession (PoP) schemes and with Macaroon tokens.

However, the conclusion drawn from several discussions was that OAuth 2.0 does not correspond to the security architecture and requirements of symbIoTe. Hence, we adopted some ideas from distributed computing systems and cloud computing. One idea that was seriously considered was the adoption of Macaroon tokens that are successfully applied for rights delegation and attenuation in cloud computing systems [5]. Macaroons eliminate complexity in the authorization code of an application in distributed computing systems. Moreover, they support decoupled authorization logic by separating the policy of user application (who can access what and when) from the mechanism (i.e. the source code) that upholds this policy. The concept of a security architecture that adopted Macaroon tokens for contextual delegation of authorities was presented in the 2nd plenary meeting in Vienna in July 2016. A detailed description of macaroon tokens is given in Section 4.4. Because a network of federated IoT platforms like symbIoTe resembles cloud computing, we claimed that the adoption of macaroons will cover the needs of that system. However, after a thorough investigation of requirements it turned out that original macaroons developed by Birgisson et al in 2014 do not support offline authentication in Smart Space domain, which is a must requirement in Smart Yachting and other use case scenarios [8]. Moreover, a privacy problem was identified, since all the attributes are shared with the platform. Other drawbacks of macaroons are the lack of a clear way to revoke access rights for specific users and the absence of search functionality. Finally, implementation of Macaroons in symbIoTe would require solving problems with managing session keys. A competitive solution that is based on JSON Web Tokens (JWT) was perceived as a promising candidate to be applied for storing and transferring security attributes. JSON Web tokens were adopted from online purchasing and they are based on claims. It turned out that JWTs match all security requirements defined in D1.2 and they are relatively easy to implement. There are libraries in Java, where JSON Web Tokens are implemented. This implementation is easy to be integrated into the symbIoTe core, which is implemented in Java and follows the microservices architectural approach. For this reason in M11 the members of symbIoTe Consortium decided to choose JSON Web Tokens for authorization.

4.2 Authentication and Authorization

Authentication and authorization in a network of federated IoT platforms can be performed by applying Attribute-Based Access Control (ABAC), which is described in Section 4.2.1. ABAC can be implemented with one of the authorization tokens: Macaroons or JSON Web Tokens, discussed in Section 4.2.2.

4.2.1 Attribute-Based Access Control

Provision of data and system security in distributed, hierarchical systems like symbIoTe requires sophisticated mechanisms of authentication and authorization for users, applications, enablers and platforms. Security requirements previously described stem from the main use case when smart devices connect to applications and different IoT platforms. Attribute-based access control (ABAC) fulfills these requirements, unlike role-

© Copyright 2016, the Members of the symbIoTe Consortium

based access control (RBAC). The latter method of authorization known from local computer networks, which assigns each user a role like 'administrator' or 'normal user', is impractical in distributed IoT environments [5]. Moreover, the fixed roles classification provided by the RBAC does not allow considering also environmental aspects when establishing rules for accessing the resources and requires a preliminary phase of role assignment, that can be unfeasible in modern IoT networks.

Security in a symbIoTe network of IoT platforms is achieved more effectively with ABAC, whose paradigm falls within a wide set of logical access control schemes. Their goal is the protection of sensitive data or services from unauthorized operations like discovering, reading, writing, creating files and so on.

ABAC is based on the assignment of 'attributes' to each client application and entity in the system. An 'attribute' is defined as a property, role or permission associated to an entity in the system, assigned after an authentication procedure by the system administrator.

In ABAC, by contrast to other access control methods, the access to resources is controlled through Access Control Policies. An access policy defined as a specific combination of attributes needed to grant access to a resource is assigned to by the owner of that resource.

Therefore, a client application, enabler or user may be granted access to a resource only if it possesses a set of attributes that match the predefined access policy. In symbIoTe, this policy can contain at the same time attributes assigned to users and objects, and environment conditions connected to the request [6].

In contrast to RBAC, ABAC is more general and allows the consideration of different heterogeneous aspects (properties of both users, applications and objects, environmental conditions, date considerations) when establishing access control policies.

The prevalence of ABAC over traditional access control schemes like Identity-Based Access Control (IBAC) or Group-Based Access Control (GBAC) is represented by the efficiency, simplicity and flexibility of the access rules. In fact, complex policies can be created and managed, without directly referencing to potentially numerous users, applications and objects. Moreover, the structure of the policy can be independent from the number of users within the system, with an enhanced flexibility especially in distributed environments, where the specific domains can avoid any form of synchronization to create consistent access control policies.

For instance, regarding the access policy depicted in Figure 4.1, an application may access the resource if and only if:

- the list of its attributes contains at least Attribute 1;
- the list of its attributes contains at least Attribute 2 and Attribute 3.

© Copyright 2016, the Members of the symbIoTe Consortium

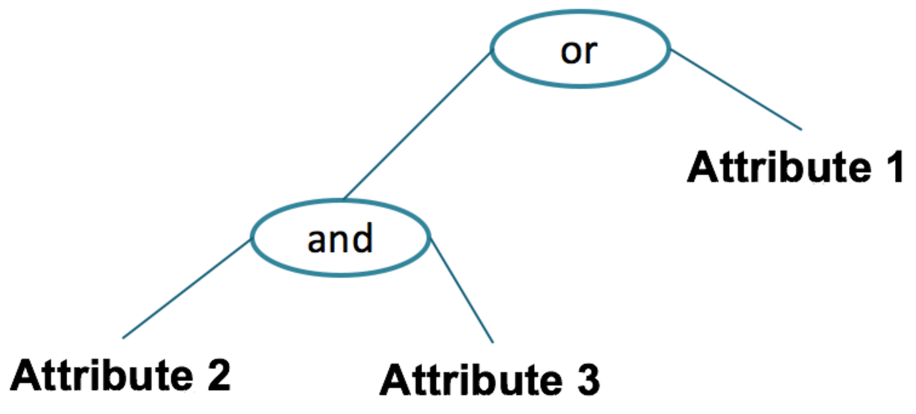


Figure 4.1: Example of Policy for the ABAC approach

4.2.2 Review of the Available Solutions

In this section, we present some possible technical solutions for the authentication and authorization architecture (OAuth2.0) and the token format (i.e. Macaroons and JSON Web Tokens), along with an initial comparison between their application in the symbIoTe context. The final comparison of both token format implementations in symbIoTe that considers security requirements and architecture is presented in Section 4.4.

4.2.2.1 OAuth 2.0

The OAuth 2.0 authorization framework, introduced in 2012, is one of the most adopted and trusty solution for managing access control in Internet. It was designed to allow a third-party application to get access to protected resources possessed by a user, without requiring the previous sharing of user's credentials [3].

OAuth 2.0 framework describes four different actors:

- **Resource Owner (RO):** the owner of a resource, able to grant or deny access to protected resources.
- **Resource Server (RS):** a dedicated server machine on which resources are stored.
- **Client**: an application that makes requests for accessing resources on behalf of the RO.
- **Authorization Server (AS)**: a dedicated server machine which issues access tokens, after authenticating with the RO.

To sum up, OAuth 2.0 allows a third-party application to obtain access to a HTTP service on behalf of a resource owner,. This is obtained through an approval interaction between the resource owner and the HTTP service. Access grants are released on a token-based approach. The definition of tokens is out-of-scope for the reference specification.

This interaction between the actors is depicted in Figure 4.2.

© Copyright 2016, the Members of the symbIoTe Consortium

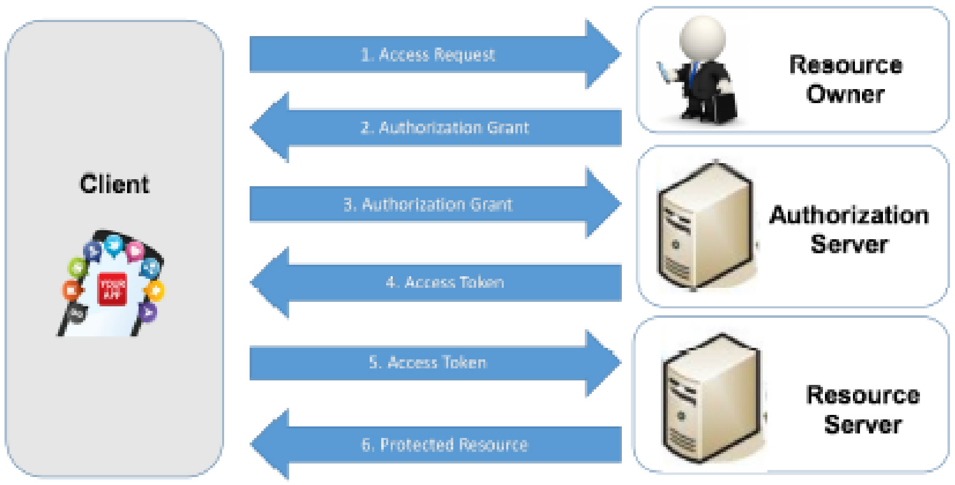


Figure 4.2: Protocol flow for the OAuth 2.0 access control framework

It is worth noting that all the messages described within the OAuth 2.0 specification are secured by the TLS protocol.

OAuth 2.0 defines different kind of tokens. An access token encodes the authorization issued to a client, and it is used to access resources on the RS. No specific details are provided regarding the format and the content of the token, but a common rule is to explicitly indicate the time-validity of this token in a dedicated entry within the token itself.

On the other hand, refresh tokens are used to obtain new access tokens. Specific implementations of the OAuth 2.0 authorization framework can customize their issuing at the AS; however, they are used only between the client and the AS.

Details about tokens are not provided by the main OAuth 2.0 authorization framework. Therefore, there are many technical documents and RFCs that specifically addressed this issue. Examples are [4] and [9].

OAuth 2.0 offers interesting functionalities, such as a general framework for managing authorization and the decoupling between evaluation of access rights and access to resources. Unfortunately, it cannot be used in symbIoTe as it is. In fact, the system architecture depicted by OAuth 2.0 is not aligned with the components identified in symbIoTe, as well as interactions envisaged between them. This is because OAuth 2.0 supposes that the control over resources is maintained by a single Resource Owner, while in symbIoTe we have different platforms (i.e., different Resource Owners) that share their resources. However, it can still be used as a useful starting point for the design of the provided security architecture.

4.2.2.2 Macaroons

Macaroons were developed by Birgisson et al. from Google in 2014 to overcome the problem of delegating access rights in public cloud computing systems like Google documents [8]. The idea is based on nested structure of HMAC. As bearer's credentials Macaroons offer similar function as cookies in World Wide Web, but they are more flexible and provide better security. Their construction relies on nested, chained HMAC (Hash-based Message Authentication Code (HMAC)) structure in a highly efficient, easy to

© Copyright 2016, the Members of the symbIoTe Consortium

deploy and widely applicable manner. Each field embedded within macaroons' structure, i.e. the *caveat*, restricts both the macaroons' authority and the context in which it can be used (e.g. by limiting the permitted operations and requiring its bearer to connect from a particular IP address to present additional evidence such as a third-party signature). These caveats are readable plain texts. A Macaroon also contains a list of AND conditions.

nonce
access_all
sign_root_macaroon

Figure 4.3: A generic structure of a root macaroon

The *main* (*root*) *macaroon*, whose generic structure is presented in Figure 4.3, gets successfully attenuated with those conditions. Each of them is signed with an HMAC function.

In the initial stage of symbIoTe project we studied the option of using macaroons for granting access to resources and delegating authorities. A network of federated IoT platforms like symbIoTe resembles public cloud computing systems like Google, Dropbox etc. in several aspects. Therefore, we claimed that Macaroon tokens designed for the purpose of fine-grained authorization in public cloud systems will fill the needs of symbIoTe. In M7 we gave a proposal of exchanging security messages between the system components according to the scheme presented in Figure 4.4.

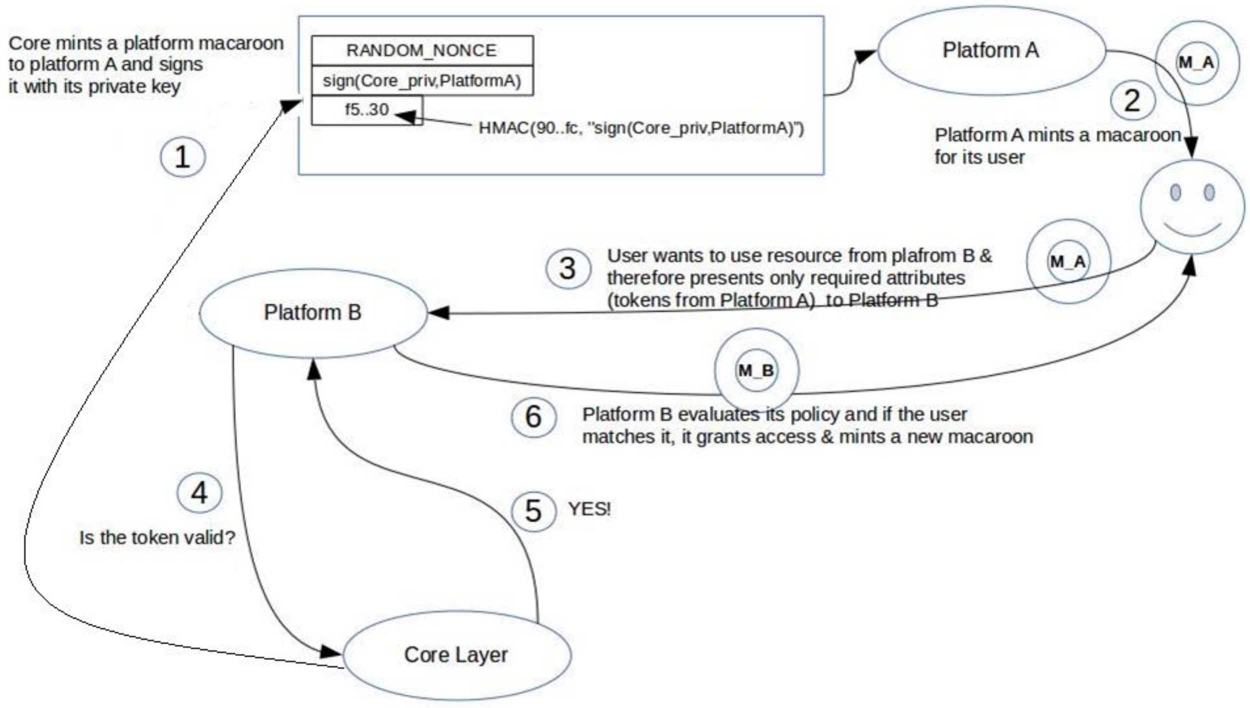

Core mints a platform macaroon to platform A and signs it with its private key
RANDOM_NONCE
sign(Core_priv,PlatformA)
f5..30
HMAC(90..fc, "sign(Core_priv,PlatformA)")
Platform A
M_A
2
Platform A mints a macaroon for its user
1
3
User wants to use resource from plafrom B & therefore presents only required attributes (tokens from Platform A) to Platform B
M_A
Platform B
M_B
6
Platform B evaluates its policy and if the user matches it, it grants access & mints a new macaroon
4
Is the token valid?
5
YES!
Core Layer

Figure 4.4: An initial proposal of Level 2 sequence diagrams that describe how macaroons are applied for granting access to resources from Platform B

© Copyright 2016, the Members of the symbIoTe Consortium

We assume that Platform B has registered its resources in core layer and a user of Platform A can find these resources through its search engine. The reference architecture for this solution was presented in M4. In accordance with it, three types of tokens can be used: the *root macaroon*, *platform macaroons* and *user/application macaroons*. The mediator, that is the symbIoTe core creates a *root macaroon* by calculating the HMAC function of a random nonce and its secret key. However, the output of the HMAC function must be shared among AAM (Authentication and Authorization Manager) of federated platforms for verifying the authenticity of tokens received during the resource access procedure. Starting from the *root macaroon* the mediator also generates *platform macaroons*. The *platform macaroons* are signed by the HMAC function with the key being the previously calculated HMAC value. Then, each platform can autonomously generate application macaroons by following the same process. However, further discussion on security architecture excluded the possibility of sequence diagrams presented in Figure 4.4. By the date of issuing this deliverable the work on Level-2 seuqence diagrams is still in progress. After obtaining a macaroon from the AAM component of its IoT platform a user can further attenuate it and delegate their authorities to other entities. For example, the owner of Smart Home platform can restrict the time his guests are allowed to open the gate to his garage and handle the appropriate token to them. Besides, the owner of the macaroon can specify whether it is possible to further delegate their authorities and restrict the potential group of interest. An example of a platform macaroon in the context of security architecture is given in Section 4.4.

4.2.2.3 JSON Web Tokens (*JWT*)

JWT is an open industry standard widely used in today's Internet to deal with authentication and authorization issues [4].

It contains a set of *claims*. A claim is a specific certified statement related both to the token itself or to the entity that is using it. Typically, these claims are encoded in the JSON format, thus easily allowing system interoperability.

A claim is identified with a specific name: it is possible to distinguish between *Registered Claim Names*, that are names defined and standardized in the reference document and *Private Claim Names*, that represent extensions that a developer could choose for his/her own system.

The cryptographic force of the JWT resides in the sign field, stored at the end of the token. It can be generated through symmetric or asymmetric cryptography techniques and allows verifying the authenticity of the token, i.e. generation by a trusted entity, as well as integrity, in the sense that no one could modify its content without invalidating it.

Each JWT contains a header that provides information about the type of the token and the algorithm used to build the sign of the token. It contains also a body, encoding a set of claims for this token, and finally - a sign containing the cryptographic validation of the token and generated as stated in the header. In Table 4.1 depicted below we can observe Registered Claim Names carried in the body of the JWT.

© Copyright 2016, the Members of the symbIoTe Consortium

Table 4.1: Registered Claim Names of the JWT and related description

Registered Claim Name	**Description**
iss	It uniquely identifies the entity that issued the token
sub	It uniquely identifies the entity for which this token has been released (it is a key field when a token needs to be used also for authentication purposes)
exp	It indicates the expiration time, after which this token should not be used and processed by any entity in the system
nbf	It identifies the time in which this token becomes effectively valid and can be processed by any entity in the system
iat	It uniquely identifies the time in which this token has been created
jti	It is the unique identifier of the token

Important features useful in symbIoTe, such as the support for an expiration date, are integrated in JWT thanks to the definition of the *exp* claim.

Also, each token can be easily associated with a given entity in the system through the *sub* claim. Specifically, the public key of the owner of the token can be embedded in this claim. This can be used in the challenge-response procedure to prove the possession of the respective private key and verify that the application using the token is effectively the entity for which the token has been generated. This procedure avoids replay attacks.

Finally, the JWT can be easily extended to support the carrying of attributes associated with the ABAC logic, thanks to the possibility to integrate customized Private Claims.

4.3 Security Architecture and Components

The reference architecture considered in this contribution is depicted in Figure 4.5.

It integrates many independent IoT platforms exposing heterogeneous resources. Each IoT platform (thus, each available resource) is registered with a trusted mediator (*i.e.* the symbIoTe core), which offers advanced mechanisms for enabling platform interoperability and distributed resource access. Moreover, there are applications willing to access the available resources.

To maximize interoperability among platforms the developed security framework has to deal with different scenarios: applications can be registered only with the symbIoTe core, only with one IoT platform, or with two (or more) IoT platforms federated with symbIoTe. Therefore, the following target scenarios can be identified:

- **Scenario #1:** an application is registered with an IoT platform and it would like to access resources exposed by the IoT platform where it is registered with. This is the case of a typical closed system, where applications, services and resources are controlled by the same administrator, without the need to interface with other platforms.

© Copyright 2016, the Members of the symbIoTe Consortium

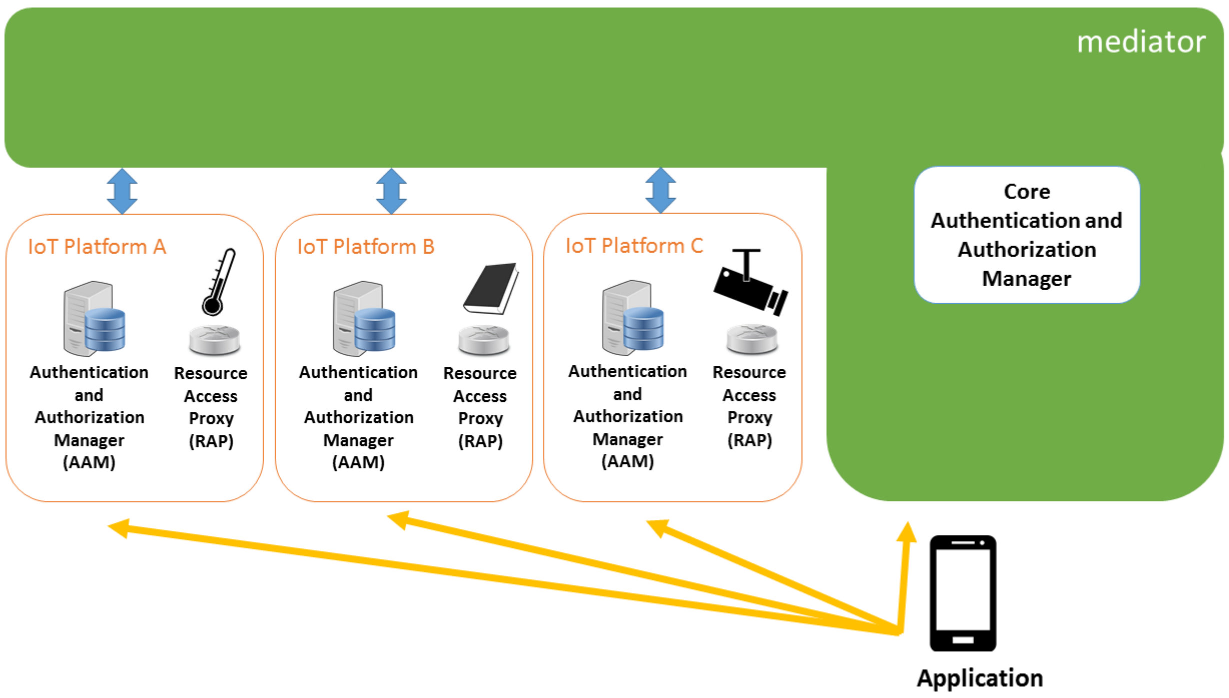

mediator
Core Authentication and Authorization Manager
IoT Platform A
Authentication and Authorization Manager (AAM)
Resource Access Proxy (RAP)
IoT Platform B
Authentication and Authorization Manager (AAM)
Resource Access Proxy (RAP)
IoT Platform C
Authentication and Authorization Manager (AAM)
Resource Access Proxy (RAP)
Application

Figure 4.5: symbIoTe System Architecture

- **Scenario #2:** an application is registered with the trusted mediator and it would like to access resources exposed by a federated IoT platform. This is the case of a third-party application developer, which implements special applications to access services and resources exposed by a given IoT platform controlled by a different administrator.
- **Scenario #3:** an application is registered with one or more IoT platforms federated with the mediator and it would like to access resources exposed elsewhere in the considered architecture. This is the case of a current sensor in a private smart home. To access the data, the system could require an application to register both with Smart-Home and the Service Provider IoT platforms. We refer to this scenario as the *multi-domain access rights composition* paradigm.

4.3.1 Main Security Rationale

The solution described hereby offers the decoupling between authentication and authorization processes. This means that authentication and authorization involve different components and are independently executed at different times. An application uses the authentication procedure to authenticate itself within a given domain (like the symbIoTe mediator or an IoT platform federated with the mediator). In case of a successful authentication it obtains a set of tokens storing its own attributes. Then, the collected attributes can be used during the authorization procedure to obtain access to resources only if the provided attributes satisfy the access control policy associated to the requested resources. A big picture of the proposed architecture is illustrated in Figure 4.6. As a general remark, the resource access is handled through the ABAC logic, described in Section 4.2.1.

© Copyright 2016, the Members of the symbIoTe Consortium

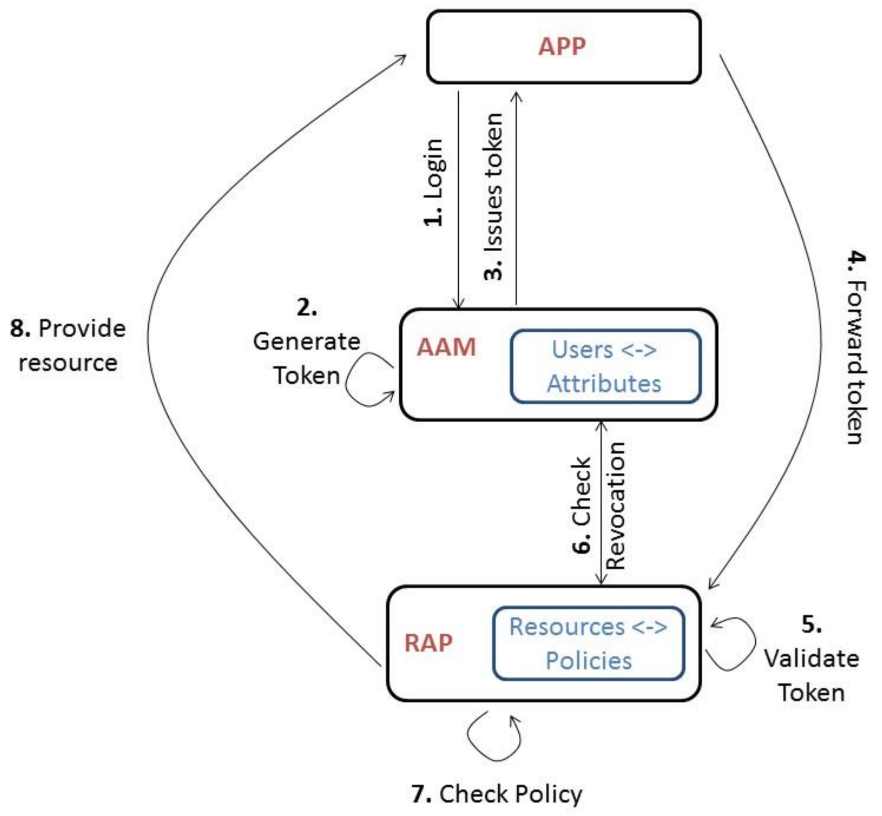


Figure 4.6: Main Security Rationale

Each platform has its own registered users/applications. Applications, as well as attributes related to each of them, are stored in the Authentication and Authorization Manager (AAM), which is the Authority for this platform. An attribute encodes a specific property, role or permission assigned to an application or component in the symbIoTe ecosystem. Attributes are stored within a digital object, namely a token, that certifies the authenticity of both the issuer (i.e., a dedicated component of the mediator or the IoT platform) and the owner (i.e., the application or the component), additionally to its time validity. Both symmetric or asymmetric cryptography techniques can be used to ensure authenticity and integrity of those tokens. To provide few examples, Macaroons, described in Section 4.2.2, use symmetric keys to generate a Hash-based Message Authentication Code (HMAC) that assures integrity and authenticity of the token, while JWTs can be created both by using symmetric or asymmetric keys. The initial configuration of users and related attributes is performed offline, by the system administrator or by the platform owner. Following a successful authentication procedure performed by an application, the AAM releases tokens, each of them encoding attributes assigned to the user. Therefore, the token represents a key element in the resource access mechanism. From the security perspective, it is generated during the authentication procedure, and inspected and validated during the authorization procedure.

Resources, as well as the access control policies, are stored on the Resource Access Proxy component, which represents the Policy Enforcement Point. An access policy enables a fine-grained access control mechanism. In fact, it describes the combination of attributes needed to obtain the access to a given resource. For each resource, a dedicated

© Copyright 2016, the Members of the symbIoTe Consortium

access policy can be defined. An application in possession of tokens storing a set of attributes matching the access policy can successfully obtain the access to the resource. Otherwise, its access request will be denied.

With reference to Figure 4.6, an application forwards the tokens to the RAP. The RAP validates the token, verifies that it has been issued for this application and if it is still valid. If the token is successfully validated, the RAP checks the policy against the set of attributes provided by the application. If the policy is satisfied, the RAP grants the access to the resource. Otherwise, the access is denied.

Since an application should not perform the whole authentication process for each resource access, the designed approach allows also for enhanced flexibility and scalability benefits for the whole system.

Note that when an application or component registered in a given IoT platform or in the symbIoTe mediator would like to access resources exposed elsewhere, it could be possible that the attributes that are assigned to it are not valid in the new domain. Therefore, an Attributes Mapping Function is needed to manage the translation between attributes in different platforms.

Thanks to the described functionalities, at the same time the interoperability framework works on top and extends the existing security architecture of a given IoT platform, providing procedures for allowing secure communications with foreign IoT platforms and third-party applications.

In what follows we provide a thorough description of functionalities and the scope of the components in the symbIoTe architecture specifically devoted to the management of security procedures. These are:

- Core Authentication and Authorization Manager (Core AAM)
- Platform Authentication and Authorization Manager (Platform AAM)
- Security Handler (SH)

4.3.2 Core Authentication and Authorization Manager

The Core Authentication and Authorization Manager (Core AAM) handles the authentication procedure only for third-party applications, i.e., applications registered with the symbIoTe core. Therefore, as a result of a successful authentication procedure, it releases *core* tokens, storing attributes that describe properties, roles and/or permissions assigned to the application at the mediator side, according to the ABAC logic described in Section 4.2.1.

Moreover, it also manages a Token Revocation List (TRL), storing the list of tokens that have been revoked before their nominal expiration (i.e., the expiration date reported in the token itself). For this reason, it may be contacted by any component in the architecture during a dedicated procedure, namely the *check revocation procedure*, to check if a given token is still valid.

When the Core AAM is contacted by a platform component or application, i.e., a component or an application that is registered within a platform federated with symbIoTe, it performs many operations to verify that the component/application is authentic, the tokens that it is providing are valid and they have not been revoked asynchronously, before the nominal expiration date reported in each of the tokens.

© Copyright 2016, the Members of the symbIoTe Consortium

The *Challenge-Response procedure* is initiated by the Core AAM to verify that the component/application that is using a token is effectively the entity for which this token has been issued. The procedure leverages public keys included in the token, and requires the component/application to demonstrate the possession of the respective private key corresponding to the public key reported in the token. During the challenge-response procedure, also the Core AAM authenticates towards the component/application, thus realizing the mutual authentication, as required by the Security Requirements no. 20.

Following a successful outcome by the previous procedure, the Core AAM executes the *token validation procedure*, aiming to verify the cryptographic validity of each of the presented tokens. Specifically, the Core AAM checks that the expiration date indicated within the token does not exceed the actual date, and that the final part of the token is cryptographically valid (it can be an HMAC in case Macaroons are used or a digital sign in case JWTs are adopted).

Then, the check revocation procedure previously described is applied.

If all the previous steps end successfully, the Core AAM executes the Attribute Mapping Function, in which it translates each of the attributes included in each of the token in a new attribute, valid within the symbIoTe core layer. Thus, a new set of tokens is issued and delivered to the component/application. These tokens will be used to access services and resources in the symbIoTe core, according the ABAC paradigm.

4.3.3 Platform Authentication and Authorization Manager

The Platform Authentication and Authorization Manager (Platform AAM) handles the authentication procedure for platform components or applications, i.e. components or applications registered in a particular IoT platform federated with symbIoTe.

As a result of a successful authentication, it releases *home* tokens storing attributes that describe properties, roles and/or permissions assigned to the component or application within the platform where it is registered in, i.e., its Home Platform. Moreover, similarly to the Core AAM, it manages the TRL and can be contacted by any component in the architecture during the *check revocation procedure*.

The Platform AAM can also be contacted by third-party components/applications or foreign components/applications, i.e. components/applications that are registered within other IoT platforms federated with symbIoTe. These situations occur when these components/applications want to access to resources hosted in the IoT platform in which the Platform AAM is located (see, e.g., the Sequence Diagram reported in Section 4.4).

As described for the Core AAM, when the Platform AAM is contacted, it performs many operations to verify that the component/application is authentic, the tokens that it is providing are valid and they have not been revoked asynchroneously, before the nominal expiration date reported in each of the tokens.

If all the previous steps end successfully, the Platform AAM executes the Attribute Mapping Function, in which it translates each of the attributes included in each of the home tokens in a new attribute, valid within the IoT platform in which it is registered. As a result, a new set of tokens is issued, namely Foreign Tokens, and they are delivered to the component/application. These foreign tokens will be used to access services and resources in the IoT platform, according the ABAC paradigm.

© Copyright 2016, the Members of the symbIoTe Consortium

4.3.4 Security Handler

The Security Handler (SH) component is a library, that can be plugged within any component in the symbIoTe ecosystem, to provide it witha wide set of security procedures and functionalities.

The installation of the Security Handler is a cornerstone operation for components that want to interact either with the core layer or with other IoT platforms.

First of all, the Security Handler allows the host component to authenticate with its reference AAM (it can be the Core AAM in case it is a third-party application or a Platform AAM if the host application is a platform application). Secondly, the Security Handler authenticates the reference AAM with which is communicating, thus providing mutual authentication.

Moreover, the SH includes the procedures to retrieve, store and manage tokens. These procedures are executed when the reference component tries to log-in in another platform or in the core layer. In this case, the SH performs also the challenge-response procedure on behalf of the reference component, by using the private/public key pair that has been assigned to it.

Finally, the SH includes also functionalities for verifying X.509 certificates delivered by foreign components to authenticate with the host component.

4.4 Token Formats and Comparison

The implementation of the security architecture described in Section 4.3 requires the selection of a suitable token format.

The following discussion does not provide only a comparison between the two approaches, but it illustrates also how Macaroons and JWTs can be modified in order to fit within the proposed architecture.

4.4.1 Token Format and Security Requirements

We made a thorough investigation of Macaroon tokens and JSON Web Tokens to identify whether symbIoTe security requirements are fulfilled by them. In Table 4.2 we summarise the results of our work. System security requirements are copied from Deliverable D1.2. Each row indicates whether the corresponding requirement is fulfilled or not. For clarity reasons we put a short message ‘YES” if the corresponding requirement is fulfilled by the authorization token that we refer to in the respective column. Otherwise, we put ‘NO’. The last column consists of comments or remarks that explain special cases. The numbers included in the “Use Cases” column denote the particular symbIoTe Use Cases which are:

1 – Smart Residence

2 – Edu Campus

3 – Smart Stadium

4 – Smart Mobility & Ecological Routing

5 – Smart Yachting

© Copyright 2016, the Members of the symbIoTe Consortium

Table 4.2: Security system requirements and their fulfillment by Macaroon tokens and JSON Web Tokens (JWT)

Index	**Domain**	**Description**	**Use Cases**	**Macaroons**	**JSON Web Tokens**	**Comments**
1	All	The system MUST offer mechanisms for the **authentication** of symbIoTe entities/actors i.e., users/application developers, IoT platforms, developed applications and clients.	1,2,3,4,5	YES	YES	
2	All	The system MUST offer mechanisms for the **authorization** of symbIoTe entities/actors i.e., users/application developers, IoT platforms, developed applications and clients.	1,2,3,4,5	YES	YES	
3	Application	The SymbIoTe ecosystem must offer mechanisms to establish trust relationships - and thus implicitly trust levels - prior to applying security mechanisms for the first time. This information must be stored in a secure datastore. e.g. by PKI infrastructure.	2	not refer to	not refer to	
4	Smart Space	The authentication to a smart space SHOULD work even if the smart space is disconnected from the Internet.	1,2,3,5	YES/NO	YES	If a copy of core AAM is included in each platform within symbIoTe then authentication with macaroons is possible. More details about this in section 4.4.3
5	Application / Cloud	The system MUST support the revocation of access rights to users/application developers, IoT platforms. *(Comment: Although in the Yachting use case it might only be revoked when the system comes online again.)*	1,2,3,4	YES	YES	This requirement can be fulfilled by allowing AAMs to asynchronously revoke tokens that they created. Therefore, it is more related to the architecture than to token format. This is supported in the architecture through the "check revocation procedure"
6	Application /Cloud	The system MUST explicitly support access rights expiration.	1,2,3,4	YES	YES	This requirement can be fulfilled by just including in the token an

© Copyright 2016, the Members of the symbIoTe Consortium

						expiration time. Both Macaroons and JWT support this feature.
7	Cloud	The authentication mechanisms of the system MUST support identity federation (e.g. single sign-on).	1,2,3,4	YES	YES	Attribute Mapping Function is responsible for translating the attributes from one platform to another. Both Macaroons and JWTs support this feature.
8	Application / Cloud	The system MUST preserve end-user privacy. (E.g. locations of end users / sent sensor data and their identity, e.g. via data anonymization)	1,2,3,4,5	not refer to	not refer to	Data anonymization can be done via generalization or indexing the sensor, This is a requirement related to the security architecture
9	Smart Space	The system MUST support encrypted data communication between all involved entities on level 1 and 2 (e.g. the SymbIoTe core, platforms, etc.).	1,2,3,4,5	not refer to	not refer to	Encryption depends on applying ciphering protocols like SSL.
10	Smart Space	The system MUST ensure privacy protection on each layer, do not publicly expose e.g., devices information or application used by applications.	1,2,3	not refer to	not refer to	
11	Application / Cloud	The system MUST support fine-grained and standardised access rights to registered IoT resources, including also aggregated resources e.g., resources provided by enablers. E.g. it must be possible to identify individual sensors (which also allows tracking their wearers) for the layer which interpolates the air quality from individual sensors. This functionality is done on a domain specific layer. The output of this will not give data from sensors away but for other entities (like street segments). E.g. In the smart stadium use case, the "normal" user should not be allowed to see locations of individuals. Certain personal security might need access to this information.	1,2,3,4	YES	YES	This feature is related more to the policy than to the token format.
12	Application / Cloud	The system MAY provide best practices guide for applications to set-up end-user security in order to function in a secure and privacy-preserving way.	2,3	not refer to	not refer to	

© Copyright 2016, the Members of the symbIoTe Consortium

13	Application / Cloud	The system MUST provide the possibility to let users / entities choose where (enablers/IoT platforms) their data is being used and processed. The users/entities MUST be able to modify the privacy parameters regarding their data.	1,2,3,4	YES	YES	With macaroons the entities can confine authorities however it is arguable if users can reject their previous operations of authority confinement; macaroon tokens cannot be modified ad hoc. However a new macaroon can be issued for revoking the previously given rights.
14	Application /Cloud	symbIoTe SHOULD detect and propagate any security error notifications through the system to application/enablers/end user.	1,2,3,4,5	not refer to	not refer to	
15	Application	The terminology used to describe the system status must not be overly technical so that users can understand without having a technical background. symbIote MAY create best practices with unified terminology for developers of applications and enablers.		not refer to	not refer to	
16	All layers	Access rules MUST be defined as an access policy.		not refer to	not refer to	Access policy is based on the attributes and both Macaroons and JWT address the problem of authorization.
17	All layers	The system MUST allow entities to delegate access to specific resources to other entities (e.g. by the usage of bearer access tokens)	1,2,3,4	YES	YES	
18	Application /Cloud	The system MUST support the authentication of the user without implying authorization for a specific resource. (E.g. it must be possible for platform B to have a user of platform A authenticated (by platform A) in a secure way while roaming in platform B)	2.4	YES	YES	If the token (either a Macaroon or a JWT) does not include any information about accessible resources, but only attributes, the same token can be used to try to access to different resources, until its validity expires. Tokens protect data integrity and identity of entities. Thus, this requirement is related both to token format and

© Copyright 2016, the Members of the symbIoTe Consortium

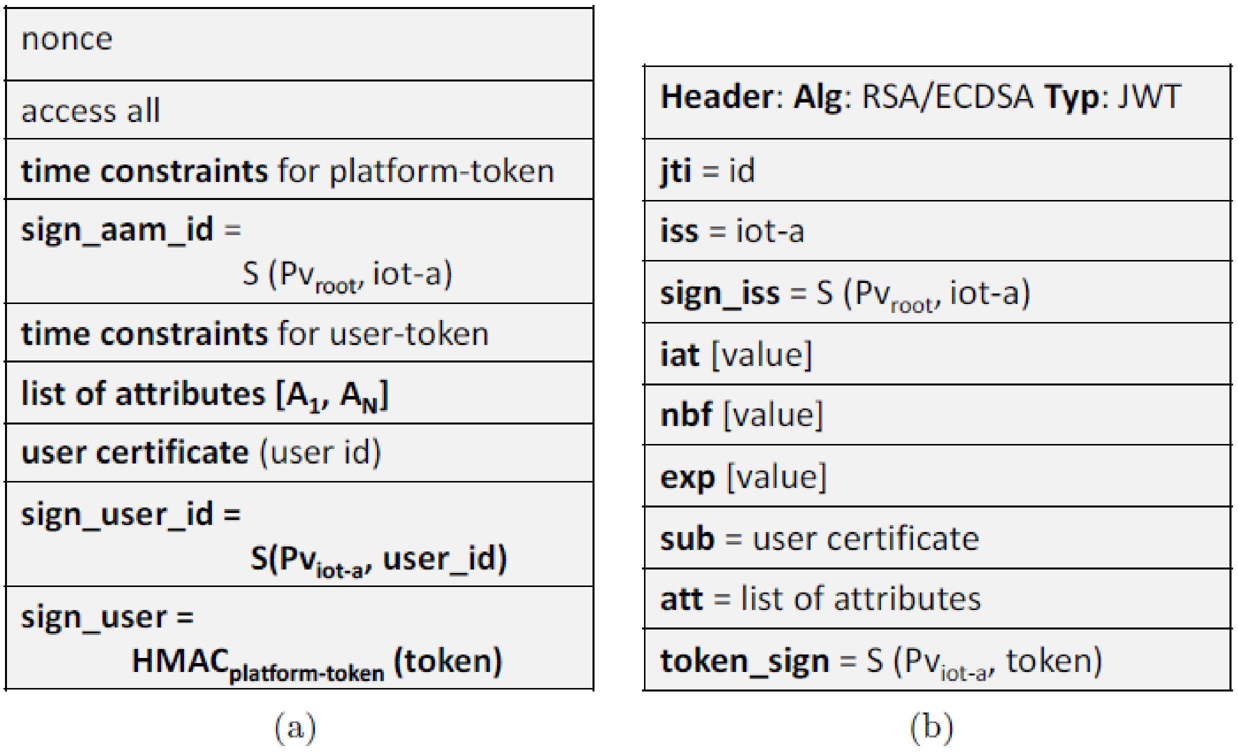
						architecture.
19	Application	Symbiote MAY support Multi-Factor Authentication if the underlying platform supports it. (e.g. Authentication using password and PIN)		not refer to	not refer to	
20	Application	Mutual authentication must be supported by all security mechanisms. (i.e. NOT only the user/application/software/... must be authenticated against the platform but also vice versa in order to facilitate malicious platform detection)	4	YES	YES	This requirement depends strongly on the implementation of the challenge-response procedure. If it is implemented correctly, both Macaroons and JWT fulfill it.
21	Application /Smart space	The access to a resource MUST be handled through 'Attribute-Based Access Control (ABAC)' schemes. An 'attribute' refers to a generic property/role/permission that the application grants during the authentication phases.	1 (MAY)	YES	YES	ABAC can be implemented both with Macaroons and with JWTs.
22	Smart Space	Within smart spaces it must be possible to run a local symbIoTe core instance for privacy and security reasons (e.g. a symbIoTe core instance installed in a smart residence). This instance might not be connected to the Internet (but could). If connected to the Internet it might expose sensors to another remote symbIoTe core instance.	2,3 (SHOULD)	YES	YES	Provided that a local instance contains a complete copy of the symbIoTe core that supports all its mechanisms and functionalities. However, if not connected to the Internet then it can work only locally.

© Copyright 2016, the Members of the symbIoTe Consortium

Observe that, according to Table 4.2, the performance of both Macaroons and JSON Web Tokens is similar for almost all security requirements. Only for requirement 4 Macaroons fail to support offline authentication unlike JWTs. This is because a root macaroon must be issued by symbIoTe core.

4.4.2 Token Content and Format in symbIoTe

In the following Figure, we compare the format of tokens described in Section 4.2.2. Even if the complexity of their structure is similar, the difficulty in implementation of security operations with each token format is an open issue.



(a)

nonce
access all
time constraints for platform-token
sign_aam_id = S (Pv$_{root}$, iot-a)
time constraints for user-token
list of attributes [A$_1$, A$_N$]
user certificate (user id)
sign_user_id = S(Pv$_{iot-a}$, user_id)
sign_user = HMAC$_{platform-token}$ (token)

(b)

Header: **Alg**: RSA/ECDSA **Typ**: JWT
jti = id
iss = iot-a
sign_iss = S (Pv$_{root}$, iot-a)
iat [value]
nbf [value]
exp [value]
sub = user certificate
att = list of attributes
token_sign = S (Pv$_{iot-a}$, token)

Figure 4.7: Authorization tokens: (a) a Macaroon, (b) a JSON Web Token

The *application macaroon* issued by a platform AAM to an application registered in the same IoT platform is showed in 4.7(a). It has a hierarchical structure. The nonce is used as a unique identifier of the token. The second, third and fourth caveats are related to the AAM of the platform in which the application is registered with. The ID of the AAM is signed through the private key of the mediator entity (PV$_{root}$). The remaining lines are dedicated to the application. They contain the list of attributes assigned to the application, its public key certificate and, finally, the sign on the user ID by the AAM that issued the token, through its private key (PV$_{iot-a}$). The last line is needed to assure that the AAM is the unique component able to sign the token. Finally, the last caveat is the chained HMAC of all the caveats in the token, performed starting from the output of the HMAC function of the root macaroon.

© Copyright 2016, the Members of the symbIoTe Consortium

The application JWT issued by a platform AAM to an application registered in the same IoT platform is showed in Fig. 4.7(b). Also, in this case we can identify the part related to the issuer of the token, certified through the sign with the private key (PV_{root}), and the part related to the owner of the token.
The private claim "*att*" is introduced to encode the information about the list of attributes possessed by the application. Finally, the sign of the whole token is performed through the private key of the issuer (PV_{iot-a}) without the need of a symmetric shared secret.

Note that both token types have a limited time validity. After the expiration of that date, the token must be renewed through a new authentication procedure.

4.4.3 Evaluation of Macaroons and JSON Web Tokens

In the 3rd Plenary Meeting that took place in Rome in November 2016 we compared the security functionalities of Macaroons and JSON Web Tokens in terms of potential use in the reference security architecture of symbIoTe that is described in Section 4.3. Three basic evaluation criteria of the considered authorization tokens were:

a) the fulfillment of security requirements
b) match between token formats and the reference architecture
c) difficulties in implementation

In the discussion, the members of symbIoTe consortium assessed that both Macaroons and JSON Web Tokens can be applied as authorization tokens in a way that fulfills the security requirements and matches the reference architecture. The performance of both tokens was investigated to be very similar in most cases. Both tokens support ABAC, delegation of rights and their contextual confinement, even though these features are implemented differently. As listed in Table 4.2 both Macaroons and JSON Web Tokens support mutual authentication, which protects against logging in malicious platforms, expiration of access rights ('time constraints' field in a platform macaroon, 'exp' filed in body of a corresponding JWT) and Attributes Mapping Function.

However, there are slight differences in performance of both tokens that have to be noted. One difference refers to the case of offline authentication. The correct performance of offline authentication while implementing Macaroons requires that an instance of symbIoTe core is launched in the considered platform. In that case, a platform AAM issues a root macaroon on behalf of the Core AAM. Besides, the access to symbIoTe core must be periodically refreshed. Otherwise, the tokens (regardless of their format) will expire, and consequently access to the resources will be prevented. In case of JWTs, offline authentication in Platform side works irrespectively of the connection to symbIoTe core or its instance. This is because JWTs can leverage asymmetric cryptography techniques. Thus, the knowledge of only the public key of the creator of the token is enough for verifying it. Another important difference between both token formats is represented by key management. We claim that the implementation of Macaroon tokens would require solving key management and distribution concerns. This problem originates from the construction of Macaroons, i.e. the issuer of the session key and its verifier (according to the original concept presented in [8]) is the same entity. Although this concept works well with public cloud computing systems, it can cause security threats in symbIoTe because the symmetric key used to generate the sign of the token would be shared with each AAM and

© Copyright 2016, the Members of the symbIoTe Consortium

RAP entity in the system. This problem can be overcome by integrating Macaroons with a key management system. In case of JSON Web Tokens, we do not observe this problem due to different construction (token_sign field that includes the private key of the issuer without the need of a symmetric shared secret).

Finally, we tried to identify possible difficulties in implementation. It turned out that while there is a publically available 'libmacaroon' library including the implementation of Macaroons in C, there is no similar library in Java [28]. In case of JSON Web Tokens, we have jjwt library that is publically available [29]. We have ready implementations of JWTs in Java and Spring, which will simplify their integration with the remaining parts of the architecture that are implemented in Java and follow the microservices architectural approach.

Taking into account security concerns and the much simplier implementation with JWTs, the symbIoTe consortium decided to choose JSON Web Tokens as authorization tokens for security operations in symbIoTe. However, we concluded that Macaroon tokens could be applied as well but with additional burden of work; that is the implementation/integration of a key management system and libraries in Java that would cooperate with rest of the software.

4.5 Interfaces and Services for the Security Components

A detailed description of interfaces and services envisioned for each security components will be provided in this section. Section 4.5.1 focuses on the Core AAM, Section 4.4.2 describes the Platform AAM, while Section 4.5.3 highlights interfaces and services for the SH. For each of the components, a table is provided highlighting the sequence diagrams in which those interfaces are used. The sequence diagrams have been described in D1.2 and they are:

- Resource Registration (S1)
- Resource Unregistration (S2)
- Resource Update (S3)
- Monitoring of Resource Availability (S4)
- Search for Resources (S5)
- Access to Resources (S6)
- Monitoring (S7)

© Copyright 2016, the Members of the symbIoTe Consortium

4.5.1 Core AAM: Interfaces and Services

Two interfaces are envisioned for the implementation of the Core AAM. They are the AppAAInterface and the PlatformAAInterface.

The AppAAInterface is the interface that groups services implemented on the Core AAM and dedicated to the communication between the Core AAM and third-party applications/components in the symbIoTe application domain. At the time of this writing, the only service included in this interface is the SignIn service, used by third-party applications/components to authenticate with the Core AAM. The inputs provided to this service are the username and the password of the application/component. The output provided by the service is a set of tokens, storing trusted attributes that are assigned to the application/component in the symbIoTe core layer.

The PlatformAAInterface is the interface that groups services implemented on the Core AAM and dedicated to the communication between the Core AAM and platform-side applications/components. The PlatformAAInterface includes two services, that are the RequestCoreToken and the CheckCoreTokenRevocation services. The RequestCoreToken service is used by foreign components/application to obtain the desired set of core tokens. The input that must be provided to this service is constituted by a list of tokens (the application/components home tokens), while the output is a list of core tokens.The CheckCoreTokenRevocation service is used by platform AAMs to verify the validity of a given token list, used by applications registered in the symbIoTe core layer when they try to access resources outside the symbIoTe core. The input to this procedure is a list of core tokens, and the output is a list of statuses, indicated if each of the input tokens is valid or not.

The following table (Table 4.3) highlights the main features of the above-described interfaces of the Core AAM.

© Copyright 2016, the Members of the symbIoTe Consortium

Table 4.3: Core AAM Interfaces and Services

Interface	*Service*	*Functionality*	*Seq. diagrams*	*Involved components*	*Input*	*Output*
AppAAInterface	SignIn	Authentication of components and applications registered in the core	S4 S5 S6	Application/ Enabler	username, password	Core token
PlatformAA Interface	Request CoreToken	Authentication of components and applications registered in the IoT platforms	S1 S2 S3 S4 S7	Registration Handler, Monitoring	Home tokens	Core tokens
	Check CoreTokenRevocation	check actual validity of home tokens	S4 S5 S6	Platform AAM, RAP	Core token	Status (YES/NO)

© Copyright 2016, the Members of the symbIoTe Consortium

4.5.2 Platform AAM: Interfaces and Services

Two interfaces are envisioned for the implementation of the Platform AAM. They are the HomeAAInterface and the ForeignAAInterface.

The HomeAAInterface is the interface that groups services implemented on the Platform AAM and dedicated to the communication between the Platform AAM and home applications/components, i.e., applications/components registered in the same IoT platform. At the time of this writing, the only service included in this interface is the SignIn service, used by third-party applications/components to authenticate with the Platform AAM. The inputs provided to this service are the username and the password of the application/component. The output provided by the service is a set of tokens, storing trusted attributes that are assigned to the application/component in the reference IoT platform.

The ForeignAAInterface is the interface that groups services implemented on the Platform AAM and dedicated to the communication between the Platform AAM and foreign applications/components, i.e. applications/components registered in the symbIoTe core layer or in another IoT platform federated with symbIoTe. The ForeignAAInterface includes two services, that are the RequestForeignToken and CheckHomeTokenRevocation services. The RequestForeignToken service is used by foreign components/applications to obtain the desired set of foreign tokens. The input that must be provided to this service is constituted by a list of tokens (the application/component home tokens), while the output is a list of foreign tokens. The CheckHomeTokenRevocation service is used by other platform or Core AAMs to verify the validity of a given token list, used by applications registered in the reference IoT platform when they try to access resources outside the reference IoT platform. The input to this procedure is a list of home tokens, and the output is a list of statuses, indicated if each of the input tokens is valid or not.

The following table (Table 4.4) highlights the main features of the above-described interfaces of the Platform AAM.

© Copyright 2016, the Members of the symbIoTe Consortium

Table 4.4: Platform AAM *Interfaces and Services*

Interface	*Service*	*Functionality*	*Seq. diagrams*	*Involved components*	*Input*	*Output*
HomeAA Interface	SignIn	Authentication of components and applications registered in the home IoT platform	S1 S2 S3 S4	Registration Handler, Monitoring	username, password	Home token
ForeignAA Interface	Request Foreign Token	Authentication of components and applications registered in the foreign IoT platforms	S4 S6	Application/Enabler	Home tokens	Foreign tokens
	CheckHomeTokenRevocation	check actual validity of home tokens	S1 S2 S3 S4 S6 S7	Core AAM, RAP	Home token	Status (YES/NO)

© Copyright 2016, the Members of the symbIoTe Consortium

4.5.3 Security Handler (SH): Interfaces and Services

A single interface is envisioned for the implementation of the SH, i.e. the SecurityInterface.

The SecurityInterface is the interface that groups services implemented on the SH and dedicated to the communication between the SH and any component in the symbIoTe ecosystem. The SecurityInterface includes three services, i.e. the ChallengeResponse, the VerifyCertificate and the VerifyToken services.

The ChallengeResponse service is used to execute operations connected with the Challenge-Response procedure. Its input is a challenge, while its output is a status that identifies if the target application/component is authentic or not.

The VerifyCertificate service, instead, is used to verify the validity of X.509 certificates supplied to the host component/application. Its input is a X.509 certificate, while its output is a status indicating if the certificate is valid or not.

Finally, the VerifyToken service is used by any component in the symbIoTe ecosystem to verify the full validity of a token. This includes the performing of the challenge-response procedure, the cryptography validation of the token and the check of the asynchronous revocation of the token.

The following table (Table 4.5) highlights the main features of the above-described interfaces of the SH.

© Copyright 2016, the Members of the symbIoTe Consortium

Table 4.5: *SH Interfaces and Services*

Interface	**Service**	**Functionality**	**Seq. diagrams**	**Involved components**	**Input**	**Output**
SecurityInterface	Challenge Response	Management of the challenge-response procedure	S1,S2, S3 S4,S5, S6 S7	Registration Handler, RAP, Application/ Enabler, Monitoring	challenge	Status (YES/NO)
	Verify Certificate	Validation of X.509 certificates	S1 S2 S3	Registration Handler, RAP	X.509 certificate	Status (YES/NO)
	Verify Token	Home/Foreign Token validation	S1, S2, S3, S4, S5, S6, S7	Registry, Resource Monitor, Search Engine, RAP	Home or Foreign Token	Status (YES/NO)

© Copyright 2016, the Members of the symbIoTe Consortium

4.6 *Preliminary Implementation*

In this section, we provide a preliminary implementation of a proof-of-concept, developed to show the interactions between applications and components in the symbIoTe ecosystem.

The proof-of-concept makes explicit reference to the sequence diagram describing the access to resources exposed in an IoT platform federated with symbIoTe by a third-party application, without reservation mechanisms, which has been thoroughly described in D2.1.

Therefore, Section 4.6.1 recaps the main operations included in this sequence diagram, while Section 4.6.2 provides more technical details about the preliminary implementation.

4.6.1 Level-1 Sequence Diagram: Access to Resources (without reservation)

The "Resource Access" procedure allows an application registered in the symbIoTe core layer to access resources in an IoT platform, which these resources are produced in and exposed. This procedure can happen both with and without resources reservation. Figure 4.8 depicts the corresponding message sequence diagram.

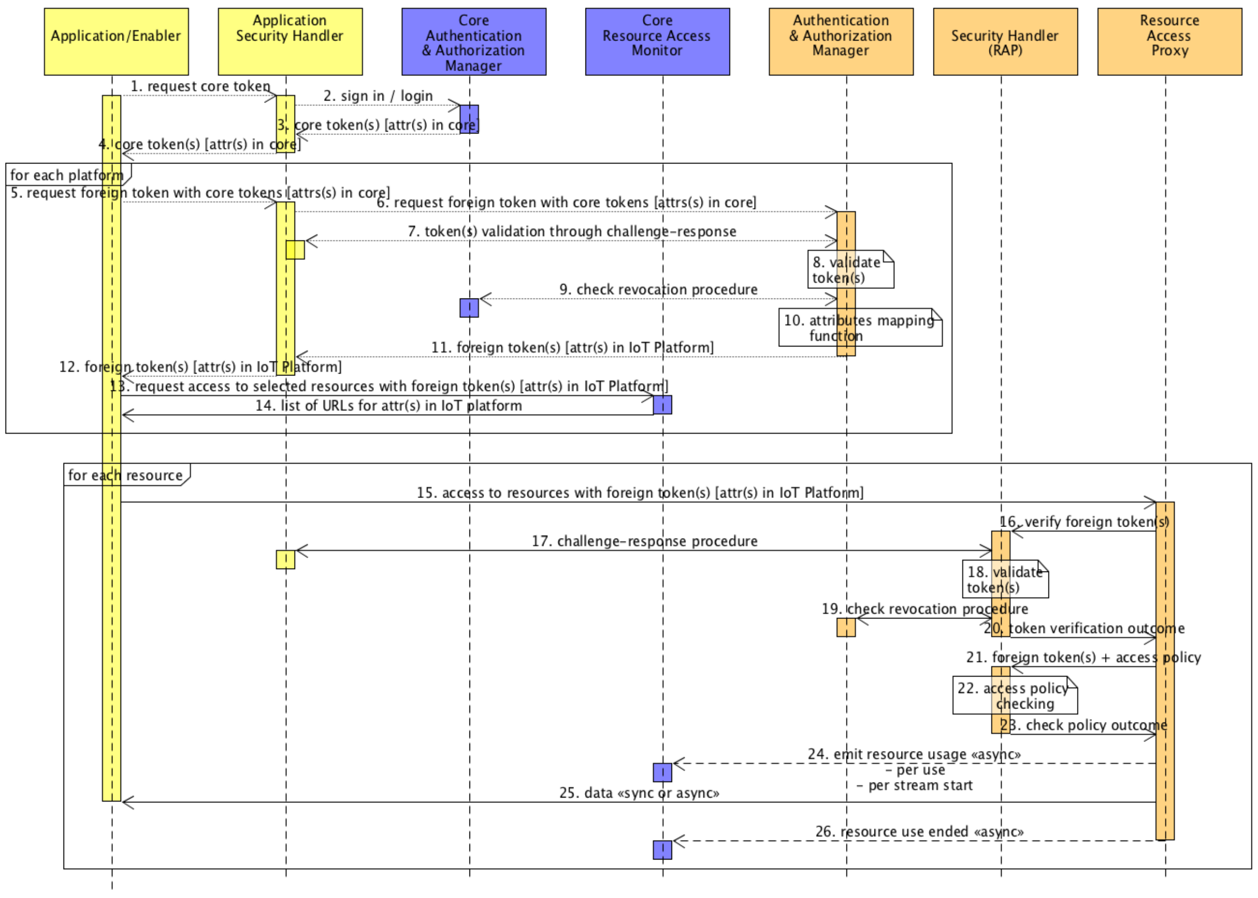

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Resource Access Monitor
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping function
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
13. request access to selected resources with foreign token(s) [attr(s) in IoT Platform]
14. list of URLs for attr(s) in IoT platform
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate token(s)
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy checking
23. check policy outcome
24. emit resource usage «async»
- per use
- per stream start
25. data «sync or async»
26. resource use ended «async»

Figure 4.8: Sequence Diagram for Level-1 Resource Access Without Reservation

© Copyright 2016, the Members of the symbIoTe Consortium

The procedure involves the following steps:

- *Message 1 (optional):* generated by the Application/Enabler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.
- *Message 2 (optional):* generated by the Security Handler and sent to the core AAM in which the Application/Enabler is registered. It is used to authenticate the Application/ Enabler. If the Application/Enabler is already logged in, it is not necessary.
- *Message 3 (optional):* generated by the core AAM in the IoT platform and sent to the Security Handler. It is used to provide the core token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary. If the Application/Enabler is not registered in the core AAM or username and/or password are wrong, the core token is not provided and a “401 Unauthorized” error code is returned to the Security Handler.
- *Message 4 (optional):* generated by the Security Handler and sent to the Application/ Enabler. It is used to deliver the core token(s).
- *Message 5 (optional):* generated by the Application/Enabler and sent to Application Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from IoT platform. If the Application/Enabler already has valid foreign token(s), it is not necessary.
- *Message 6 (optional):* generated by the Application Security Handler and sent to the foreign AAM in IoT platform. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- *Procedure 7 (optional):* procedure that allows the Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary. If an error occurs during the challenge-response procedure, a “401 Unauthorized” error code is returned to the application and no foreign token is provided.
- *Procedure 8 (optional):* verification of the time validity, authenticity and integrity of the provided token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary. If an error occurs during the token validation procedure, a “403 Forbidden” error code is returned to the application and no foreign token is provided.
- *Procedure 9 (optional):* verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Application/Enabler already has valid foreign token(s), it is not necessary. If an error occurs during the check revocation procedure, a “403 Forbidden” error code is returned to the application and no foreign token is provided.
- *Procedure 10 (optional):* procedure that, in case it is needed, translates attributes that the Application/Enabler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Application/Enabler already has valid foreign token(s), it is not necessary.

© Copyright 2016, the Members of the symbIoTe Consortium

- *Message 11 (optional):* generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- *Message 12 (optional):* generated by the Application Security Handler and sent to the Application/Enabler. It is used to forward the foreign token generated at the previous step.
- *Message 13 (mandatory):* Application/Enabler sends request access to selected resources to Core Resource Access Monitor. Message includes foreign tokens obtained in previous message
- *Message 14 (mandatory):* Core Resource Access Monitor returns list of URLs for selected resources in IoT platform.
- *Message 15 (mandatory):* generated by the Application/Enabler and sent to the Resource Access Proxy in the foreign IoT platform. It is used to access resources, while providing the foreign token previously obtained.
- *Message 16 (mandatory):* generated by the Resource Access Proxy and sent to the Security Handler in the foreign IoT platform. It is used to ask to the security handler to verify the complete validity of the token.
- *Procedure 17 (mandatory):* procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s). If an error occurs during the challenge-response procedure, a “401 Unauthorized” error code is returned to the RAP and no resource is provided by the RAP to the Application/Enabler.
- *Procedure 18 (mandatory):* verification of the time validity, authenticity and integrity of the provided token(s). If an error occurs during the token validation procedure, a “403 Forbidden” error code is returned to the RAP and no resource is provided by the RAP to the Application/Enabler.
- *Procedure 19 (mandatory):* verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the output is NO, this means that the token has been revoked. Thus, a “403 Forbidden” error code is delivered to the RAP and no resource is provided by the RAP to the Application/Enabler.
- *Message 20 (mandatory):* generated by the Security Handler in the foreign IoT platform and sent to the Resource Access Proxy. It is used to communicate the outcome of the token validation procedures performed by the Foreign Security Handler. If the output is NO, no resource is provided by the RAP to the Application/Enabler and one of the previously indicated error codes (401 or 403) is returned to the Application/Enabler.
- *Message 21 (mandatory):* generated by the Resource Access Proxy and sent to the Security Handler. It is used to deliver the foreign token(s) previously verified and the access policy of the requested resource to the Security Handler.
- *Procedure 22 (mandatory):* it is used to check if the attributes included in the foreign token(s) satisfy the access policy associated to the requested resource. If the access policy is not satisfied, a NO status is returned to the RAP. Thus, no resource is

© Copyright 2016, the Members of the symbIoTe Consortium

provided and a “403 Forbidden” error code is returned by the RAP to the Application/Enabler.

- *Message 23 (mandatory):* generated by the Security Handler and sent to the Resource Access Proxy. It is used to deliver the result of the operation executed at the previous step.
- *Message 24 (optional):* asynchronously emit resource usage per use/per stream start
- *Message 25 (mandatory):* this message can be synchronous, then Resource Access Proxy returns data. If it is asynchronously then it can emit asynchronous messages for some time.
- *Message 26 (optional):* if previous message is asynchronous then this message informs Core Resource Access Monitor when the stream is ended

4.6.2 Details on the Preliminary Implementation

A preliminary proof-of-concept has been developed to show the interactions between components described in Section 4.3 of this document.

The proof-of-concept refers to the sequence diagram described in Section 4.6.1, with explicit reference to the case of resource access without reservation, depicted in Figure 4.8.

The perspective through which the diagram has been analyzed is from the Application/Enabler’s point of view. Therefore, the preliminary implementation has been divided into many steps, assembling together messages that are related to each other with a request/response relationship.

Moreover, some figures include both the part of the sequence diagram involved in the step and a terminal, showing the details on the message sent and received by the application when interacting with another component in the symbIoTe ecosystem.

Each step has been thoroughly described in the following subsections.

4.6.2.1 Step 1: Request Core Token

The Application/Enabler sends a request to obtain the core token(s) from the Core AAM to the Security Handler.

© Copyright 2016, the Members of the symbIoTe Consortium

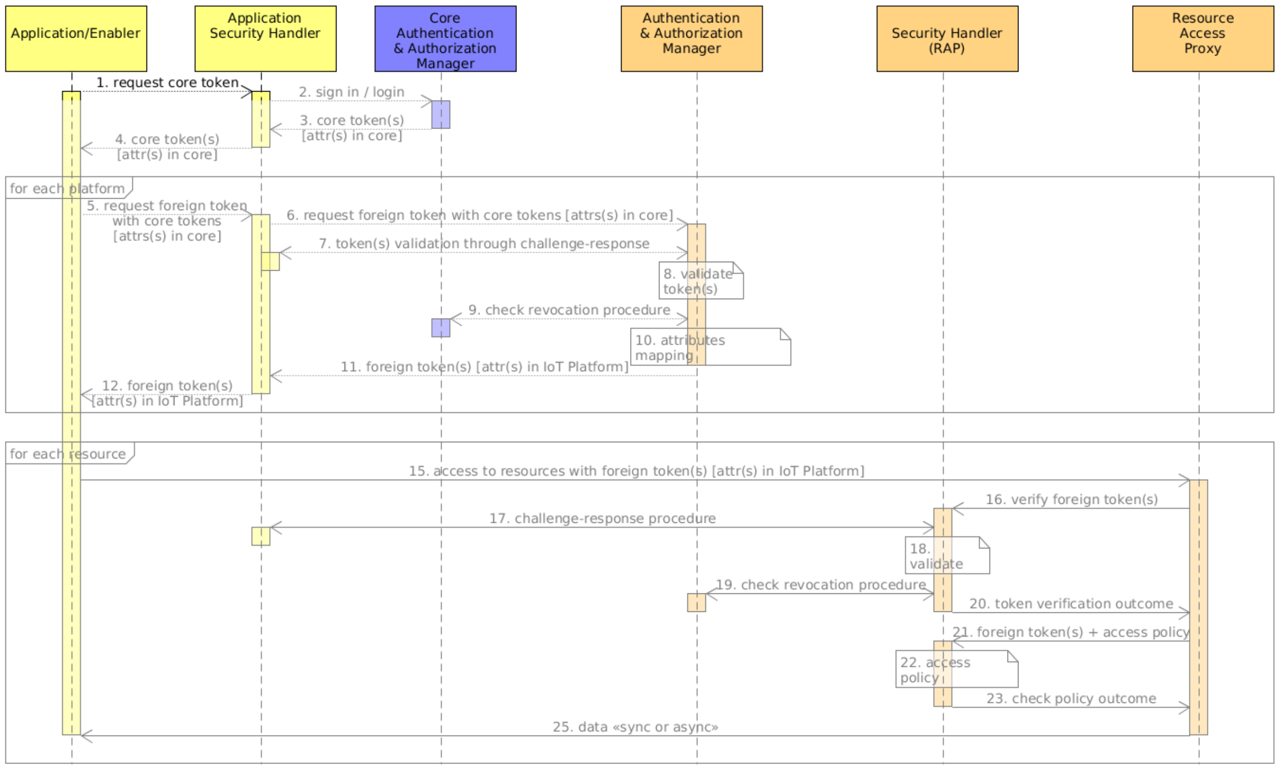

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»

Figure 4.9: Step 1 of the Proof-of-Concept implementation

Interface: AppAAInterface

Service: AppAAInterface_RequestCoreTokens

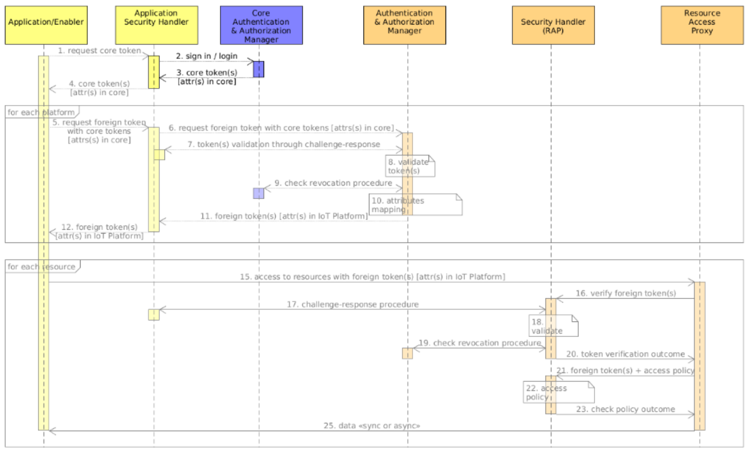
input: username, password

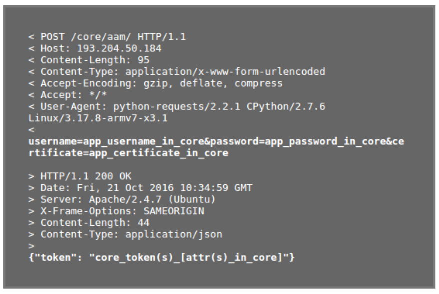
output: core token(s)

4.6.2.2 Step 2: Core Token creation

The Security Handler requests the list of the core tokens available for this application from the Core AAM.

© Copyright 2016, the Members of the symbIoTe Consortium


Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»



```
< POST /core/aam/ HTTP/1.1
< Host: 193.204.50.184
< Content-Length: 95
< Content-Type: application/x-www-form-urlencoded
< Accept-Encoding: gzip, deflate, compress
< Accept: */*
< User-Agent: python-requests/2.2.1 CPython/2.7.6
Linux/3.17.8-armv7-x3.1
<
username=app_username_in_core&password=app_password_in_core&ce
rtificate=app_certificate_in_core

> HTTP/1.1 200 OK
> Date: Fri, 21 Oct 2016 10:34:59 GMT
> Server: Apache/2.4.7 (Ubuntu)
> X-Frame-Options: SAMEORIGIN
> Content-Length: 44
> Content-Type: application/json
>
{"token": "core_token(s)_[attr(s)_in_core]"}
```

Figure 4.10: Step 2 of the Proof-of-Concept implementation

Interface: AppAAInterface

Input: username of the application, password of the application, IP address of Core AAM

Output: list of core tokens

4.6.2.3 Step 3: Return core token to the Application

The Security Handler returns the Core token previously requested in Step 1 to the Application.

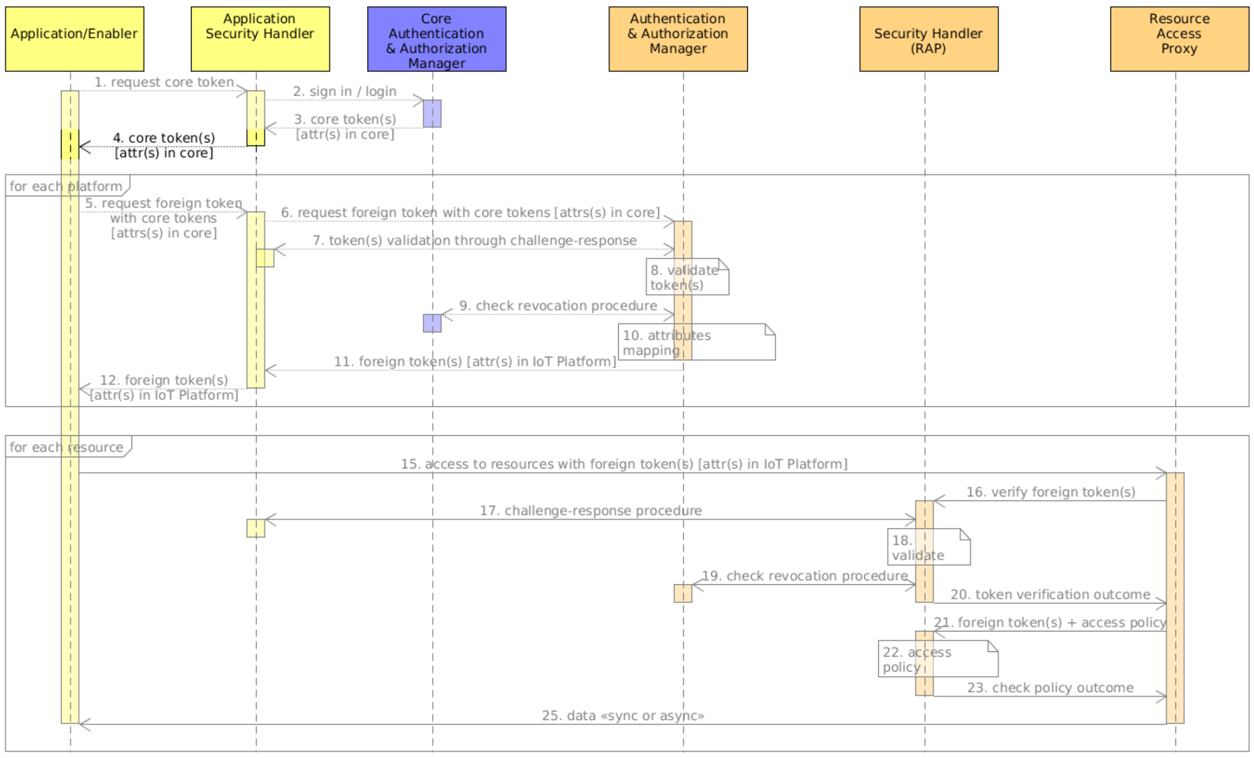

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»

Figure 4.11: Step 3 of the Proof-of-Concept implementation

© Copyright 2016, the Members of the symbIoTe Consortium

As this is the return call to the function initiated in Step 1, the Interfaces and services involved are the same.

Interface: AppAAInterface

Service: AppAAInterface_RequestCoreTokens()

input: username of the application, password of the application, IP adress of the Core AAM

output: list of core token(s)

4.6.2.4 Step 4: Request Foreign Token

The Application/Enabler sends a request to obtain the foreign token(s) from the Foreign AAM to the Security Handler.

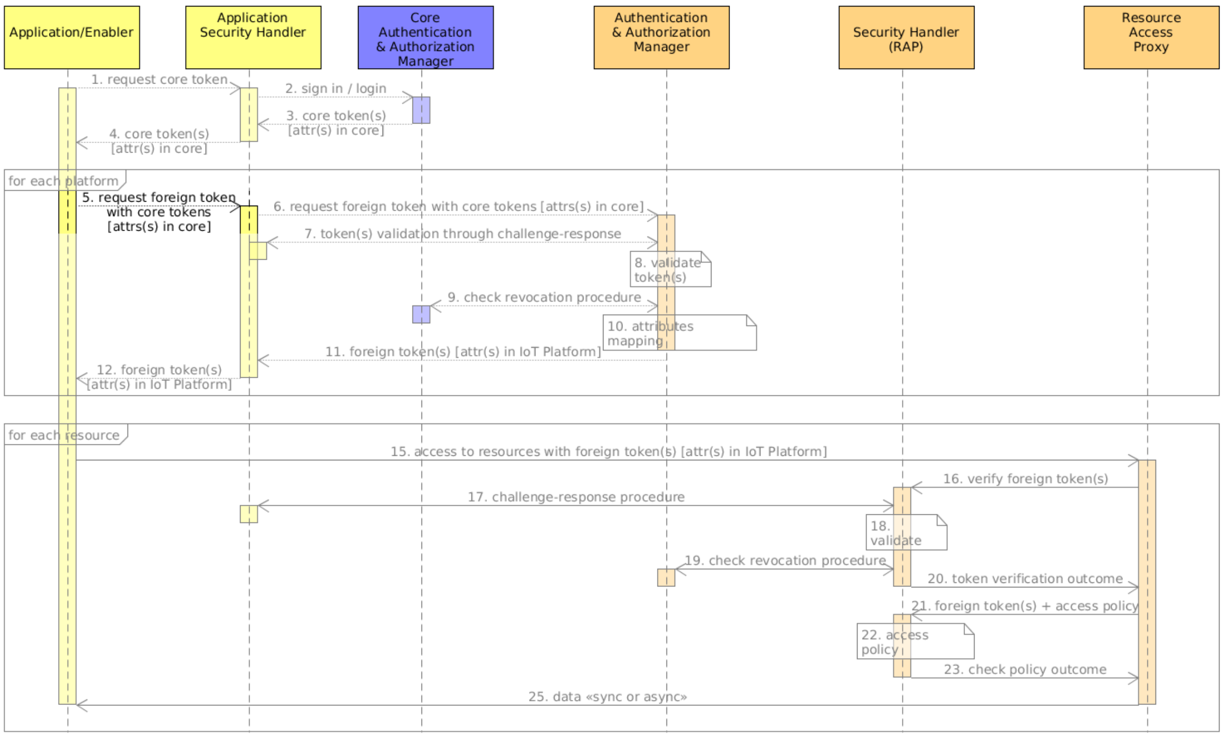

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»

Figure 4.12: Step 4 of the Proof-of-Concept implementation

Interface: ForeignAAInterface

Service: ForeignAAInterface_RequestForeignTokens();

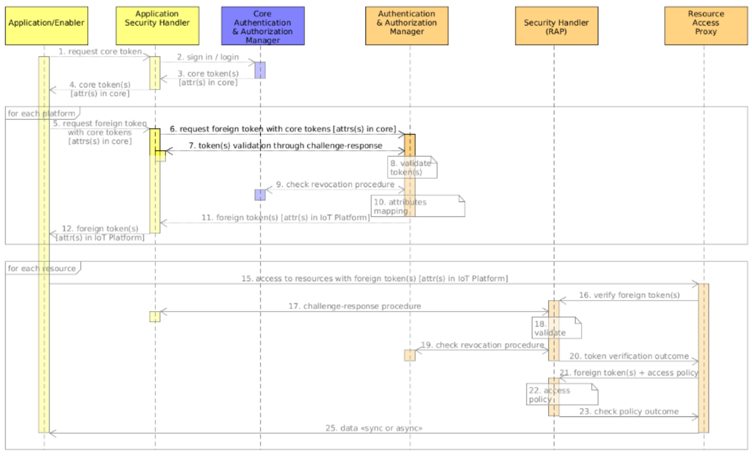
Input: List of Core Tokens, IP address of the Foreign AAM

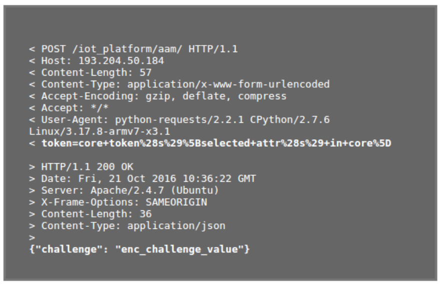
Output: List of Foreign Tokens

4.6.2.5 Step 5: Application Authentication to the Foreign AAM

The Security Handler running on the Application/Enabler forwards the core token to the Foreign AAM for validation.

© Copyright 2016, the Members of the symbIoTe Consortium


Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»


```
< POST /iot_platform/aam/ HTTP/1.1
< Host: 193.204.50.184
< Content-Length: 57
< Content-Type: application/x-www-form-urlencoded
< Accept-Encoding: gzip, deflate, compress
< Accept: */*
< User-Agent: python-requests/2.2.1 CPython/2.7.6
Linux/3.17.8-armv7-x3.1
< token=core+token%28s%29%5Bselected+attr%28s%29+in+core%5D

> HTTP/1.1 200 OK
> Date: Fri, 21 Oct 2016 10:36:22 GMT
> Server: Apache/2.4.7 (Ubuntu)
> X-Frame-Options: SAMEORIGIN
> Content-Length: 36
> Content-Type: application/json
>
{"challenge": "enc_challenge_value"}
```

Figure 4.13: Step 5 of the Proof-of-Concept implementation

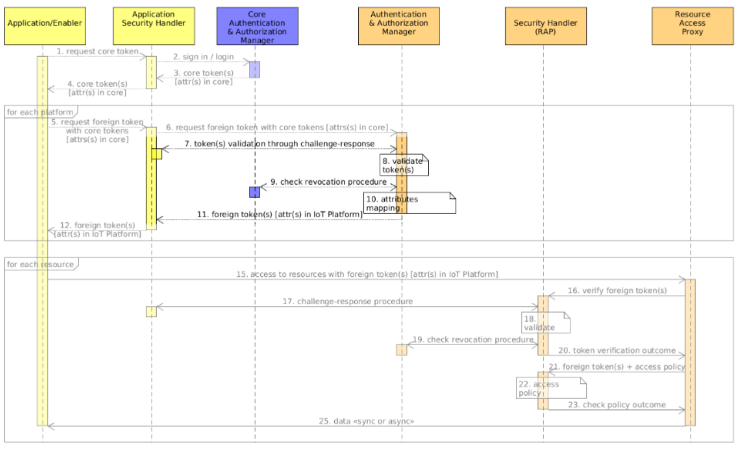
Interfaces: ForeignAAInterface, SecurityInteface

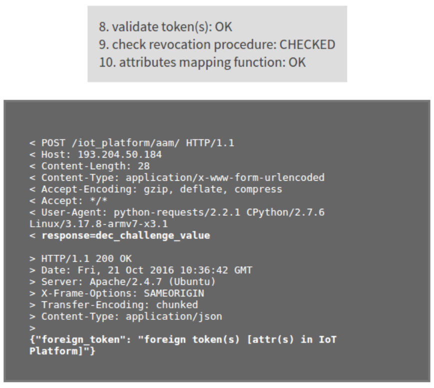
Input: List of Core Tokens, IP address of the Foreign AAM

Output: Challenge value

4.6.2.6 Step 6: Foreign Token request

The Security Handler provides the response to the challenge received at Step 5 and requests the foreign token from the Foreign AAM.


Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»


8. validate token(s): OK
9. check revocation procedure: CHECKED
10. attributes mapping function: OK

```
< POST /iot_platform/aam/ HTTP/1.1
< Host: 193.204.50.184
< Content-Length: 28
< Content-Type: application/x-www-form-urlencoded
< Accept-Encoding: gzip, deflate, compress
< Accept: */*
< User-Agent: python-requests/2.2.1 CPython/2.7.6
Linux/3.17.8-armv7-x3.1
< response=dec_challenge_value

> HTTP/1.1 200 OK
> Date: Fri, 21 Oct 2016 10:36:42 GMT
> Server: Apache/2.4.7 (Ubuntu)
> X-Frame-Options: SAMEORIGIN
> Transfer-Encoding: chunked
> Content-Type: application/json
>
{"foreign_token": "foreign token(s) [attr(s) in IoT
Platform]"}
```

Figure 4.14: Step 6 of the Proof-of-Concept implementation

Interfaces: ForeignAAInterface, SecurityInterface, PlatformAAInterface

Services: ForeignAAInterface_RequestForeignTokens, SecurityInterface_ChallengeResponse,PlatformAAInterface_CheckHomeTokenRevocation

Input: Response, List of Core Tokens, IP address of the Foreign AAM

Output: List of Foreign Tokens

© Copyright 2016, the Members of the symbIoTe Consortium

4.6.2.7 Step 7: Return Foreign Token to Application

The Security Handler of the Application/Enabler returns the foreign token to the Application/Enabler, previously requested in Step 4.

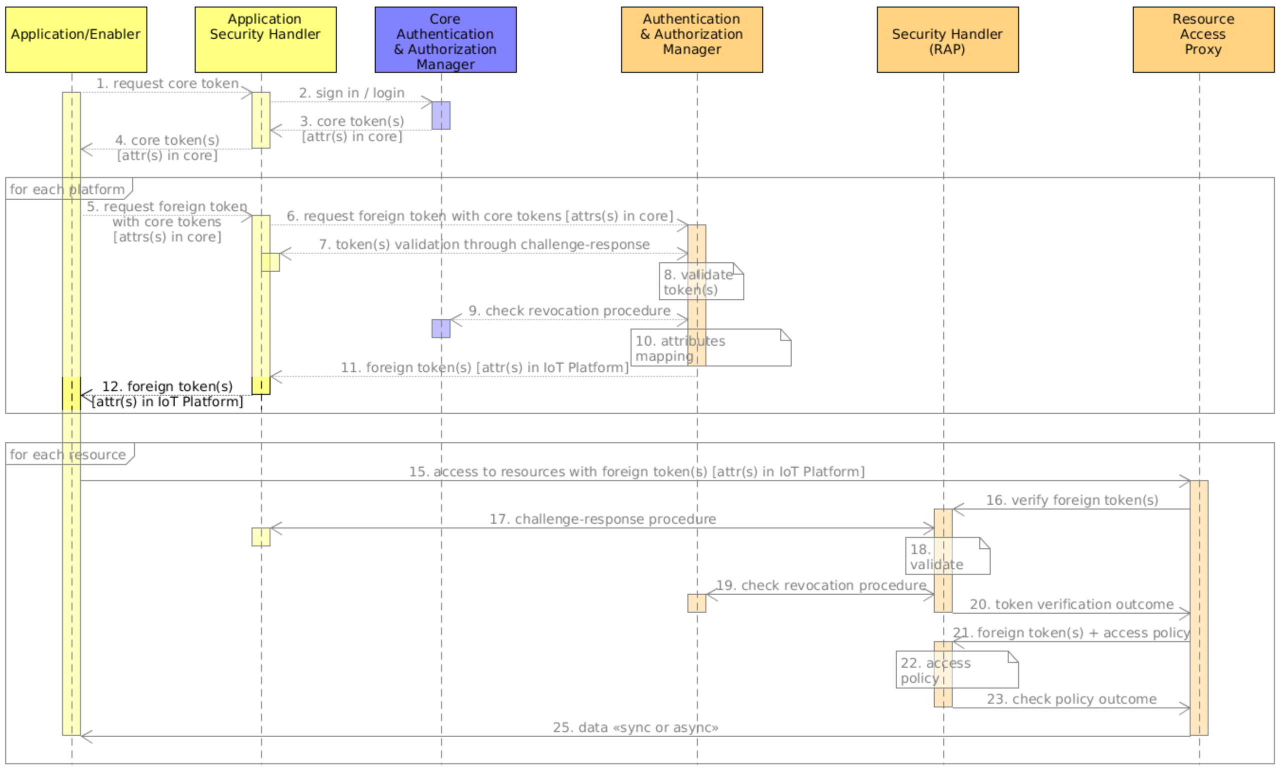

Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»

Figure 4.15: Step 7 of the Proof-of-Concept implementation

As this is the return call to the function initiated in Step 5, the related API is the same.

Interfaces: ForeignAAInterface, SecurityInterface, PlatformAAInterface

Services: ForeignAAInterface_RequestForeignTokens,
SecurityInterface_ChallengeResponse,
PlatformAAInterface_CheckHomeTokenRevocation

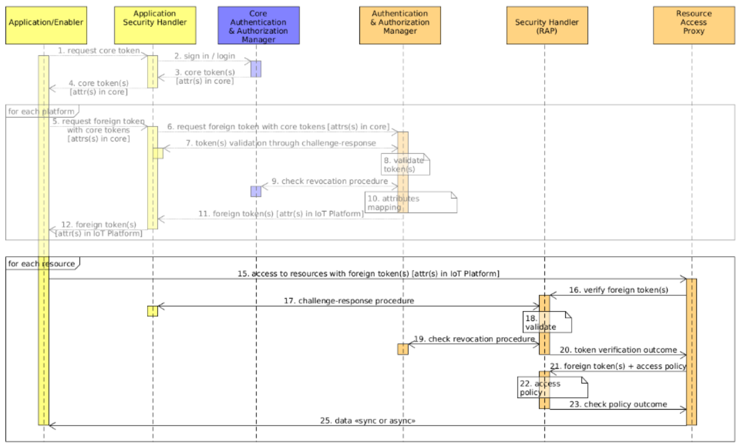
Input: Response, List of Core Tokens, IP address of the Foreign AAM

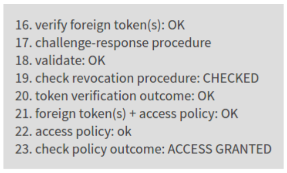
Output: List of Foreign Tokens

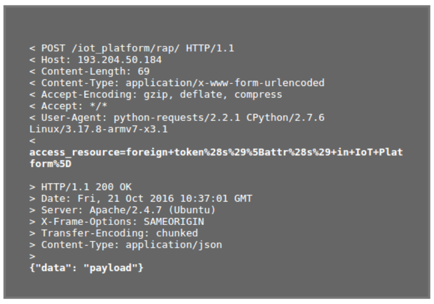
4.6.2.8 Step 8: Access Foreign Resources with Foreign Token

The Application/Enabler makes an access request to the foreign RAP to obtain the desired resource.

© Copyright 2016, the Members of the symbIoTe Consortium


Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge-response
8. validate token(s)
9. check revocation procedure
10. attributes mapping
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge-response procedure
18. validate
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy
23. check policy outcome
25. data «sync or async»


16. verify foreign token(s): OK
17. challenge-response procedure
18. validate: OK
19. check revocation procedure: CHECKED
20. token verification outcome: OK
21. foreign token(s) + access policy: OK
22. access policy: ok
23. check policy outcome: ACCESS GRANTED


```
< POST /iot_platform/rap/ HTTP/1.1
< Host: 193.204.50.184
< Content-Length: 69
< Content-Type: application/x-www-form-urlencoded
< Accept-Encoding: gzip, deflate, compress
< Accept: */*
< User-Agent: python-requests/2.2.1 CPython/2.7.6
Linux/3.17.8-armv7-x3.1
<
access_resource=foreign+token%28s%29%5Battr%28s%29+in+IoT+Plat
form%5D

> HTTP/1.1 200 OK
> Date: Fri, 21 Oct 2016 10:37:01 GMT
> Server: Apache/2.4.7 (Ubuntu)
> X-Frame-Options: SAMEORIGIN
> Transfer-Encoding: chunked
> Content-Type: application/json
>
{"data": "payload"}
```

Figure 4.16: Step 8 of the Proof-of-Concept implementation

Interface: Access Resource Interface (RAP)

Service: access resources

Input: RAP IP address, resource ID, list of foreign tokens, metadata (i.e., freshness);

Output: status, resource Value.

4.7 Other Security Related Issues

In symbIoTe, there are much more security concerns than authentication of users and granting access to resources. First, the system must be resistant against Denial-of-Service attacks, which can happen when a lot of entities overload the network with abnormally high number of requests directed to the same network component. Besides, privacy of end users and sensors must be protected by proper data anoymization. The search engine must be designed in such a way that prevents illegitimate users to acquire knowledge on the network structure of foreign platforms and symbIoTe core or gain administrative laws.

The Sections above focused mainly on security at the software level. However, hardware security issues must also be addressed. Sensors, actuators and other network devices must be hardened in terms of operating system, i.e. default entity names available in search engine (cannot expose information about the manufacturer or firmware version), passwords and a default configuration of database management system must be replaced with a dedicated configuration. Passwords change policy and password rules (i.e. at least 8 letter, at least one capital letter, one digit, one special sign) must be obeyed as well. Moreover, anomalous behavior in symbIoTe must be detected.

For this reason, a centralized approach to Anomaly Detection (AD) in symbIoTe was presented in M11. There are two approaches to threat detection in information systems. The first one relies on signatures of known attacks. However, it cannot detect 0-day attacks and other unexpected behaviour in a computer network. Another approach is based on detecting anomalies, which are patterns of data that do not conform to a well-defined notion of behaviour. Anomaly detection can detect flaws or failures not related to security. On the other hand, it can yield improper results that is *false positives* (showing

© Copyright 2016, the Members of the symbIoTe Consortium

that a normal behaviour is an anomaly) or *false negatives* (not detecting threats). Moreover, it needs to properly train and instrument the system to work correctly. Last but not least, huge processing capabilities may be required to launch an anomaly detection module.

Initial discussion considered both a centralized and a decentralized approach to anomaly detection. In the decentralized approach, each IoT platform federated within symbIoTe would have its own anomaly detection module, whereas in the centralized one, an AD module is a dedicated entity within symbIoTe core. The prevalence of the centralized approach over the decentralized one is represented mainly in simplified implementation.

In the centralized approach that is under discussion, an AD module is situated in symbIoTe core. Its performance is based on Complex Event Processing (CEP) approach. It collects data from platforms that expose the logs from all operations that refer to resource access through generic RAP. The AD module has a tunable training phase, where a system learns patterns of ‘typical behaviour’. For instance, it can learn that a sensor in Smart Home is polled 5 times per hour on average. Thus, if this sensor is polled 50 times per minute, then the AD module can detect anomaly suspecting that it is an attempt of a DDoS attack.

The proposed implementation is based on Zipkin Distributed Tracing service for traffic analysis in symbIoTe core and Hystrix Circuit Braker for Health Check on services. However, for correct performance of AD we also need statistics on external traffic which is done by data mining of logged usage traffic reported by interworking API modules in each platform. This data might be available for monthly periods to platform owners as usage statistics reports – statistics module/DB.

Thus, thanks to Netflix (open source software for analysing operations in the cloud) we can analyse the symbIoTe core and thanks to statistics we know how the traffic inside the core is related to the traffic in platforms and how the traffic in Platforms affects the traffic in the core layer. This is necessary to make reasonable decisions and avoid false alarms.

The final decision on the performance of anomaly detection module in symbIoTe and its implementation is still an open point.

4.8 Summary

Chapter 4 gave a description of different access scopes. Useful starting points for the proposal of the authentication and the authorization architecture were the ABAC approach to authorization and its implementation with Oauth2.0, Macaroons, and JSON Web Tokens. Starting from these concepts, we presented a security architecture designed by us and evaluated the functionalities of two competitng token formats. We decided that while both Macaroons and JWTs can successfully be applied to symbIoTe, the implementation with JWTs is much simpler and does not bring additional implementation and key management concerns. Finally, we presented a demo that justified our approach. Besides, we briefly described other security related issues and threats that are present in a network of federated IoT Platforms. Finally, we described our approach to anomaly detection that is currently under discussion. A detailed description of the anomaly detection module in terms of performance and implementation will be given in D3.2.

© Copyright 2016, the Members of the symbIoTe Consortium

5 Next Steps for T3.1 and T3.2

In this chapter, we will briefly outline a few key research topics which, based on the present document, are planned for task T3.1 “Resource Trading and User-centricity” and T3.2 “Security and Access Scopes” as next steps.

While the present deliverable has strongly focused on the B2B case, T3.1 will also have a look into B2C scenarios, where devising an efficient auction scheme is of key importance. Hand in hand with task T1.2, we will also contribute to develop further on the opportunities which two-sided market theory will provide for trading in the symbIoTe case.

As already mentioned earlier, future work will also deal with defining parameters for basic resource trading and detailed prosumer modelling. In this context, suitable modelling of utility functions will contribute to the internal management of resource allocation based on corresponding tradeoffs. For this purpose, also investigating the expected Quality of Service (QoS) and/or Quality of Experience (QoE), e.g. with respect to response time vs. availability, and cost functions for symbIoTe resources will be of specific relevance.

Another important topic to be tackled soon concerns a more general approach for bartering which for instance allows for circular bartering (prosumer 1 offering good 1 to prosumer 2 who offers good 2 to prosumer 3 who offers good 3 to prosumer 1) as well as voucher composition (prosumer 1 composing two or more vouchers to receive a desired good from prosumer 2 and/or 3 in exchange for good 1).

Moreover, we will contribute to the evolving description of several symbIoTe modules which will further facilitate the bartering and trading functionality, including the Resource Access Proxy (RAP), the Federation Manager and the Search Engine.

Next steps in T3.2 will be the implementation of anomaly detection module and making final decision on Level 2, Level 3 and Level 4 sequence diagrams. First, the implementation of security components that were described in Section 4.4 and 4.5 (i.e Core AAM, SH, platform AAM) will be our main concern in next months. Interfaces and services described in Section 4.5 will be implemented and will be ready for the next release of the symbIoTe demo (planned for the end of M13), while the complete solution, including the full set of security functionalities, will be provided by the end of M15.

Moreover, a specific definition of “Attributes” in the system will be provided, in strict coordination with the work of the WP2. Accordingly, a logic for the Attributes Mapping Function will be defined, in order to manage attributes assigned to applications also in foreign platforms.

Finally, technical solutions for the challenge-response authentication procedure and details about the token content will be further detailed.

© Copyright 2016, the Members of the symbIoTe Consortium

6 References

[1] Hal Varian: *Intermediate Microeconomics. A Modern Approach*. W. W. Norton, New York, 2010.

[2] A. Lazar, N. Semret: *Design and Analysis of the Progressive Second Price Auction for Network Bandwidth Sharing*. Telecommunication Systems – Special issue on Network Economics, 1999.

[3] D. Hardt (Ed.): *The OAuth 2.0 Authorization Framework*. RFC 6749, IETF, Oct 2012, available online https://tools.ietf.org/html/rfc6749.

[4] M. Jones, J. Bradley, N. Sakimura: *JSON Web Token (JWT)*. RFC 5719, IETF, May 2015.

[5] L. Macvittie: *ABAC not RBAC. Welcome to the (IoT) world of contextual security*. September 2015. Available online: https://f5.com/about-us/blog/articles/abac-not-rbac-welcome-to-the-iot-world-of-contextual-security

[6] V. Hu, D. Ferraiolo, R. Kuhn, et al.: *Guide to Attribute Based Access Control (ABAC) - Definition and Considerations*. NIST Special Publication, January 2014. Available online:

http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf

[7] symbIoTe project Deliverable D1.2: *Initial Report on System Requirements and Architecture*. September 2016.

[8] A. Birgisson, J. Gibbs Politz, U. Erlingisson, M. Lentczner: *Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud*. Proc. of the Conf. on Network and Distributed System Security Symposium, 2014.

[9] M. Jones, D. Hardt: *The OAuth 2.0 Authorization Framework: Bearer Token usage*. RFC 6750, IETF, Oct. 2012.

[10] *What is a Payment System?* (PDF). Federal Reserve Bank of New York. 13 Oct 2000.

[11] B. Bossone, M. Cirasino: *The Oversight of the Payment Systems: A Framework for the Development and Governance of Payment Systems in Emerging Economies*. The World Bank, July 2001, p.7

[12] E. Turban, D. King, J. McKay, P. Marshall, J. Lee, D. Vielhand: *Electronic Commerce 2008: A Managerial Perspective.* London, Pearson, 2008, p.550

[13] *Mastercard: Security Rules and Procedures*-Merchant Edition (PDF). 2009. Retrieved: May 12, 2009

[14] https://www.searchenginejournal.com/the-10-most-popular-online-payment-solutions/, Visited October 2016

[15] http://electroniccommercepaymentsystem.weebly.com/3-smart-cards.html

[16] https://www.paypal.com/webapps/mpp/paypal-fees

[17] http://uk.creditcards.com/credit-card-news/paypal-vs-plastic-1372.php

© Copyright 2016, the Members of the symbIoTe Consortium

[18] http://money.cnn.com/infographic/technology/what-is-bitcoin/

[19] J. Kopstein: *The Mission to Decentralize the Internet.* The New Yorker, 12 Dec 2013), retrieved 30 December 2014.

[20] https://www.google.com/wallet/faq/

[21] https://www.android.com/pay?referrer=utm_source=wallet&utm_medium=website&utm_campaign=faqs

[22] https://en.wikipedia.org/wiki/Comparison_of_payment_systems

[23] http://www.digitaltrends.com/mobile/android-pay-guide/2/

[24] http://fortune.com/2016/03/24/apple-pay-gets-easier-to-use-this-year/

[25] http://www.theverge.com/2016/10/26/13414232/samsung-pay-online-purchases-us-2017

[26] https://interledger.org/interledger.pdf

[27] https://webonanza.com/2015/10/07/why-w3cs-payment-system-proposal-interledger-wont-work/

[28] https://github.com/rescrv/libmacaroons

[29] https://github.com/jwtk/jjwt

© Copyright 2016, the Members of the symbIoTe Consortium

7 List of Acronyms

AAM	Authentication and Authorization Manager
ABAC	Attribute-Based Access Control
AD	Anomaly Detection
AMF	Attributes Mapping Function
API	Application Programming Interface
AS	Authorization Server
B2B	Business to Business
B2C	Business to Customer
B&T	Bartering and Trading
CEP	Complex Event Processing
DB	Database
DDoS	Distributed Denial-of-Service
DoS	Denial-of-Service
DoW	Description of Work (Technical Project Annex)
GBAC	Group-Based Access Control
HMAC	Hashed-based Message Authentication Code
HTTP	Hypertext Transport Protocol
IBAC	Identity-Based Access Control
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
LETS	Local Exchange Trading System
NFC	Near-Field Communications
OAuth	Open Authentication
OS	Operating System
PoP	Proof of Possession
PSP	Progressive Second Price
QoS	Quality of Service
QoE	Quality of Experience
PIN	Personal Identification Number

© Copyright 2016, the Members of the symbIoTe Consortium

PKI	Public Key Infrastructure
RAP	Resource Access Proxy
RBAC	Role-Based Access Control
RO	Resource Owner
RF	Radio Frequency
RFC	Request for Comments
RS	Resource Server
SH	Security Handler
SLA	Service Level Agreement
TRL	Token Revocation List
TLS	Transport Layer Security
URL	Uniform Resource Locator
VGC	Vickrey-Clarke-Groves

© Copyright 2016, the Members of the symbIoTe Consortium