Secure, Centralized Multi-Stakeholder Telemetry Data Acquisition and Distribution for the Internet of Things

Martin Maritsch

Martin Maritsch BSc

Secure, Centralized Multi-Stakeholder Telemetry Data Acquisition and Distribution for the Internet of Things

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Priv.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner
Institute of Interactive Systems and Data Science

Graz, July 2017

Martin Maritsch BSc

Sichere, zentralisierte Aggregation und Distribution von Telemetriedaten zwischen mehreren Stakeholdern im Internet der Dinge

Masterarbeit

für den akademischen Grad

Diplom-Ingenieur

Masterstudium: Software Engineering and Management

an der

Technischen Universität Graz

Begutachter

Priv.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner
Institute of Interactive Systems and Data Science

Graz, Juli 2017

Diese Arbeit ist in englischer Sprache verfasst.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

_____ _____

Date/Datum Signature/Unterschrift

Abstract

Increasing the efficiency and flexibility of industrial processes is a major goal of ongoing initiatives like *Industry 4.0* or the *Industrial Internet*. Smart and connected industrial equipment will substantially change organizational structures and business processes [Porter and Heppelmann, 2015]. For example, recurring maintenance, repair and overhaul (MRO) activities for sophisticated industrial equipment requires personnel to service equipment to minimize downtime, prevent failure or improve performance. However, a vendor supplies its equipment to its global customer base, requiring to send MRO technicians there. Thus, preemptively scheduling MRO activities is a central use case and often labeled *Intelligent Maintenance Services* or – as it is referred to in this thesis – *Smart Maintenance Services (SMSs)*.

Lesjak, Hein, Hofmann, et al. [2015] designed and implemented a prototype system for equipment status data (*snapshot*) acquisition on customer-side. Our work focuses on the snapshot being in transport and the vendor-side backend. Thus an analysis of existing Internet of Things (IoT) protocols for data transmission is performed and they are compared to other popular protocols.

In the proposed system, a broker-based message exchange system is used to transfer snapshots and other arbitrary information (e.g. firmware updates) from industrial equipment to vendors or customers and vice versa. This broker based system is described by Maritsch, Lesjak, and Aldrian [2016]. The data exchange systems based on a broker provides at least two major advantages for the scenario: First, all participants need to actively push or pull data to the broker. Thus, a vendor does not need to initiate connections to customer premises to reach its equipment. Second, when all snapshots are exchanged via the broker, customers are able to audit transmitted data to check that no sensitive non-maintenance relevant data was send off its equipment.

The possibilities of how to systematically structure data and flows inside the SMS framework to allow an easy aggregation of desired data for different stakeholders are shown. Emphasis is placed on the scalability, extendability and failure resistance of the system.

In parallel to the broker-based system implementation, the security aspect is always kept in mind. It is vital, to ensure the confidentiality and authenticity of sensitive customer data. For this purpose current, standardized security techniques for authentication and authorization are shown. A mechanism is presented, which allows to securely transmit data in a globally distributed SMS framework. Furthermore, multiple security functions which protect data on different stages of SMS framework dataflows, are proposed.

Kurzfassung

Die Effizienzsteigerung und Flexibilität von industriellen Prozessen ist eines der Hauptziele von Initiativen wie *Industrie 4.0* oder dem *Industriellen Internet*. Intelligente und vernetzte industrielle Geräte werden sukzessive organisatorische Strukturen und Businessprozesse ändern [Porter und Heppelmann, 2015]. Beispielsweise erfordern sich wiederholende Wartungs-, Reparatur- und Überhol-Tätigkeiten (WRÜ) für hochkomplexe industrielle Geräte spezielles Personal um Standzeiten zu minimieren, Fehler vorzubeugen und die Leistung zu steigern. Jedoch unterhalten HerstellerInnen solcher Geräte oft einen weltweiten KundInnenstock, welcher es erfordert, dass SpezialistInnen von HerstellerInnen dorthin entsandt werden. Deshalb wird das präventive und dadurch kosteneinsparende Einplanen solcher WRÜ-Tätigkeiten oft als *Intelligent Maintenance Services* oder *Smart Maintenance Services (SMSs)* bezeichnet.

Lesjak, Hein, Hofmann u. a. [2015] entwarfen und implementierten ein prototypisches System um Statusdaten von Geräten (*Snapshots*) zu erfassen. Diese Arbeit fokussiert sich auf den Transport dieser Snapshots zum Hersteller/zur Herstellerin und auf das HerstellerInnen-seitige Backend. Daher werden bestehende Internet-Of-Things-Protokolle (IoT) für Datenübertragung evaluiert und mit anderen populären Protokollen verglichen.

Im vorgeschlagenen System wird ein Broker-basiertes Nachrichtenaustausch-System implementiert um Snapshots und andere beliebige Daten, wie zum Beispiel Firmware-Updates, von industriellen Geräten zum Hersteller/zur Herstellerin oder zum Kunden/zur Kundin und vice versa zu übertragen. Dieses Broker-basierte System wird von Maritsch, Lesjak und Aldrian [2016] beschrieben. Der Datenaustausch über einen zentralen Broker hat zumindest zwei große Vorteile: Erstens müssen alle TeilnehmerInnen aktiv Daten an den Broker schicken oder von dort abholen. Daher müssen HerstellerInnen nicht aktiv Verbindungen zu deren Geräten bei KundInnen initiieren. Zweitens können KundInnen, wenn alle Snapshots über diesen zentralen Broker übertragen werden, diese einfach prüfen und sicherstellen, dass keine sensitiven, nicht-wartungsrelevanten Informationen von seinen/ihren Geräten übertragen werden.

Es werden verschiedene Möglichkeiten aufgezeigt, wie man Daten und Datenflüsse systematisch strukturieren kann um eine einfache Datenaggregation innerhalb des SMS-Frameworks für verschieden StakeholderInnen zu gewährleisten. Dabei wird spezieller Wert auf die Skalierbarkeit, Erweiterbarkeit und Fehlertoleranz des Systems gelegt.

Parallel zur Implementierung des Broker-basierten Nachrichtenaustausch-Systems wird immer den Sicherheitsaspekt im Blick behalten. Es ist unumgänglich, dass die Vertraulichkeit und

Richtigkeit von Gerätedaten sichergestellt wird. Es wird ein Mechanismus gezeigt, welcher es erlaubt Daten in einem global verteilen SMS-Framework sicher zu übertragen. Des Weiteren werden mehrere Sicherheitsmaßnahmen aufgezeigt, welche die Daten in verschiedenen Layern des SMS-Frameworks schützen.

Contents

List of Abbreviations

ACL

Access Control List. ii, xiii, 53, 54, 63, 81

AE

Authenticated Encryption. 24

AES

Advanced Encryption Standard. iii, xiii, 24, 26, 30–33, 49, 67, 70, 89

API

Application Programmer Interface. 52, 61, 64

ASN.1

Abstract Syntax Notation One. 29, 32, 64, 89

AVL

AVL List GmbH. ix, 2, 5–7, 87, 89, 90

BPM

Business Process Model. iii, x, 71, 72, 75

CA

Certificate Authority. 23, 24, 27, 50, 51, 63, 64, 84

CMS

Cryptography Message Syntax. ii, iii, ix, xiii, 22, 28–33, 55, 57, 59, 64, 65, 67–69, 83, 84, 88

CoAP

Constrained Application Protocol. 16

CPS

Cyber-physical system. 1, 4, 5

CRL

Certificate Revocation List. 23, 24, 49, 50, 64, 83, 84

CSR

Certificate Signing Request. 50

DMZ

Demilitarized Zone. 40–44, 60, 61, 80, 81

ECC

Elliptic Curve Cryptography. 30, 31, 48, 77

ECDSA

Elliptic Curve Digital Signature Algorithm. 26, 61

ECMQV

Elliptic Curve Menezes-Qu-Vanstone. 31–33, 67

FM

Fuel Meter. 36, 40

GCM

Galois/Counter Mode. iii, xiii, 24, 30–33, 67, 70, 89

HTTP

Hypertext Transfer Protocol. x, 15–18, 50, 79, 80, 85, 87

IIoT

Industrial Internet of Things. 3, 14

IoT

Internet of Things. i–iii, IX, XI, 2, 3, 10, 11, 13–17, 19, 21, 78, 87, 89, 90

IP

Internet Protocol. 17, 18, 28, 79

ISO

International Standards Organization. 16, 77, 89

M2M

Machine-to-Machine. 10, 16, 89

MAC

Message Authentication Code. 24, 25, 28, 31, 49, 55, 67

MQTT

Message Queue Telemetry Transport. ii, iii, ix–xi, 11–13, 15–22, 26, 28–30, 35–46, 50–55, 59, 61, 63, 64, 77–85, 87–90, 93, 94

MQV

Menezes–Qu–Vanstone. 30

MRO

maintenance, repair and overhaul. IX, 2, 5

OASIS

Organization for the Advancement of Structured Information Standards. 16, 89

OSI

Open Systems Interconnection. ix, 26, 28, 29

PC

Particle Counter. 36, 40

PKCS #12

Public Key Cryptography Standard #12. 63, 67

PKI

Public Key Infrastructure. 23, 24, 49, 84

QoS

Quality of Service. ii, ix, xi, 18–21, 36, 80, 85

SHA

Secure Hash Algorithm. 26, 49

SMS

Smart Maintenance Service. ii, IX, XI, XII, 2, 5–10, 12–14, 17, 35, 36, 39, 41, 47–52, 54–56, 58–60, 71, 72, 77–84, 87–91

SMTP

Simple Mail Transfer Protocol. 17, 18, 79

SOA

service-oriented architecture. ii, iii, ix, x, 2, 13, 14, 77, 78, 87

SPOF

Single Point of Failure. 41, 42

TACS

Topic Access Control System. ii, iii, ix, 12, 47, 52, 53, 56, 78, 81, 83

TCP

Transmission Control Protocol. 9, 16–18, 28, 79, 80

TLS

Transport Layer Security. ii, iii, ix, xiii, 12, 18, 22, 26, 28–30, 35, 37, 39, 40, 47–52, 55, 61, 63–66, 77, 82–84, 88

UDP

User Datagram Protocol. 16

URI

Uniform Resource Identifier. 23, 50

XML

Extensible Markup Language. 16

XMPP

Extensible Messaging and Presence Protocol. 16

List of Figures

List of Tables

List of Listings

Acknowledgements

I would like to thank the following individuals for their great support and collaboration throughout my work on this thesis:

- The great folks at evolaris next level GmbH. Beginning with my supervisor Thomas Ebner, who provided me with great technical and organizational guidance, to all the colleagues I got to work with during this thesis (Martin Lessacher, Matej Vuković, Richard Hable, Ernesto Rico-Schmidt, Markus Schüssler, Christian Kittl). I am very grateful that I have had the opportunity to work with them.
- The colleagues in the Arrowhead team: Andreas Aldrian (AVL GmbH); Christian Lesjak, Thomas Ruprechter (Infineon Technologies Austria AG); Michael Hofmann, Günther Pregartner (GUEP Software GmbH); Daniel Hein (Institute for Applied Information Processing and Communications, Graz University of Technology). Their great competencies were the foundation of our fruitful work in the smart services area. Special thanks go to Christian Lesjak for encouraging and supporting me in the process of writing research papers.
- My academical supervisor Martin Ebner from the Institute of Interactive Systems and Data Science (ISDS) at Graz University of Technology for his guidance and support in the academical realization of this thesis.
- My family, especially Arnold and Gerhild, which provided me with ever ongoing support throughout my studies which finally led to this thesis. I am grateful for the sheer endless energy and resources they have put into motivating and encouraging me.

Martin Maritsch
Graz, Austria, August 2017

Chapter 1

Introduction

“ Industry 4.0 is more than just a flashy catchphrase. A confluence of trends and technologies promises to reshape the way things are made. ”

[Cornelius Baur and Dominik Wee, McKinsey & Company]

1.1 Motivation

Industrial organizations are constantly facing challenges to optimize their productivity and processes in order to survive in global competition. As technology found its way into industrial production and factories, many processes were improved and automatized.

From a historical point of view, industrialization started back in the late 18th century, with the *First Industrial Revolution*. Water and steam power was used to power mechanical machines. This was the basis for optimization of repeating work tasks.

With the *Second Industrial Revolution* starting in the late nineteenth century, electrical power completely changed processes in factories and led to still existing technologies like the conveyor belt. Mass production became the new form of fabrication.

The *Third Industrial Revolution* – also known as the *Digital Revolution* – was caused by electronics and information technology which led to further automation. Undebatable, the invention of electronic chips and computers had an dramatical impact on everyday live.

According to [Kagermann et al., 2013] mankind is now just at the beginning of the *Fourth Industrial Revolution*, which is based on Cyber-physical systems (CPSs). Industrial production should be interconnected with modern information technology tools. The goal is a widely autonomous and self-organizing production line by introducing communication between machines, humans and all other entities in factories.

Factories in the future will benefit from interconnected devices and big scale data analysis. Holmes [2015] described the *Factory of the Future* as follows:

> *"The factory of the future today is automated, connected, and integrated. It is also a journey rather than an absolute end state. It is also dynamic in nature. A key attribute is the culture of the organization: to be able to proactively act upon the real-time information and insight into the manufacturing process, and to be able to continuously improve the process is fundamental to making the factory of the future successful."*

1.1.1 The *Arrowhead* project

Arrowhead[1] is an international research project funded by the European Union. With more than eighty partners from all over Europe, it addresses efficiency and flexibility of industry at a global scale. It is the ultimate goal to enable automation based on the collaboration of network enabled devices. The challenge is, to interconnect all these devices and enable interoperability; Internet of Things (IoT) technologies shall be used for this purpose. The *Arrowhead* project aims to produce a framework which allows this interoperability. This framework is realized as a service-oriented architecture (SOA) and publicly available[2].

The Austrian consortium of this project, being led by AVL List GmbH (AVL), focuses on building an IoT automation system for enabling Smart Maintenance Services (SMSs) (Section 1.3) in automotive testbed engineering systems.

1.2 Background

Increasing the efficiency and flexibility of industrial processes is a major goal of ongoing initiatives like Industry 4.0 or the Industrial Internet. Smart and connected industrial equipment will substantially change organizational structures and business processes [Porter and Heppelmann, 2015]. For example, recurring maintenance, repair and overhaul (MRO) activities for sophisticated industrial equipment requires personnel to service equipment to minimize downtime, prevent failure or improve performance. However, a vendor supplies its equipment to its global customer base, requiring to send MRO technicians there. Thus, preemptively scheduling MRO activities is a central use case and often labeled *Intelligent Maintenance Services* or – as it is referred to throughout this thesis – *SMSs*.

Priller, Aldrian, and Ebner [2014] proposed a SMS framework which brings connectivity to a vendor's end of line testing equipment being used in engine production. They sketched the

[1] `www.arrowhead.eu`, Accessed: May 3, 2017

[2] `www.arrowhead.eu/arrowhead-wiki/`, Accessed: May 3, 2017

concept of the *mediator*, a device that equips legacy industrial equipment with the necessary connectivity to send maintenance relevant status data to remote endpoints via the Internet.

1.2.1 The Internet of Things (IoT)

The term *Internet of Things* describes, how (electronic) devices are getting connected to the Internet. One can perceive this trend with everyday items such as fridges, coffee makers or washing machines becoming "smart" when they are connected to the Internet. The IoT trend has already arrived in everyday products; just recently a *smart hairbrush*[3] was announced by a subsidiary of Finnish *Nokia*.

However, not the devices themselves are becoming smart. They will normally just exchange data or transmit status data to a central backend, which is usually located at the vendor. The real intelligence is in those vendor controlled backends, where value is generated by evaluating the aggregated data.

When speaking of IoT devices, it is usually talked of technical devices, with access to the Internet. Not all of these devices will have a good availability of power and network resources like for example a fridge in a personal home would. Many devices will be located in restricted areas and will only have limited access to those resources. Thus, when developing standards for the IoT, an emphasis needs to be placed on efficiency and resource usage. Due to different communication patterns, some traditional methods will not work anymore and new challenges arise.

The *Industrial Internet of Things (IIoT)* describes the use of the IoT in an industrial context, specifically in relation to Industry 4.0. The IIoT will connect devices in factories and enable smart services.

In industrial applications, the IoT is facing different challenges compared to usages in private environments. Security and data privacy have a completely different value. Not only for keeping trade secrets but also since data processing guidelines are required by law.

The IoT is certainly one key contributor to what is called *Big Data*; the collection and analysis of tremendous amounts of data. IoT enables organizations to collect data in a previously unknown fashion and dimension. This kind of data aggregation and the subsequent analysis of data is especially relevant for applications in an industrial context. If an organization was for example able to predict machine failures and act before they even happen, downtimes could be minimized and maintenance costs reduced.

[3] `www.withings.com/eu/en/products/hair-coach`, Accessed: May 3, 2017

1.2.2 Smart Services

There is no clear definition of what a "smart service" is. It rather depends on the context where it is applied. However, it is often described as generating value from the extraction, aggregation and evaluation of existing products' data.

Tillotson and Lundin [2012] explained smart services as follows:

> *"Smart Services are the commercial realization of the Internet of Things for manufacturers, where post-sales product support capabilities enable a fundamental change in the value equation of business."*

When speaking of a smart service, a system is meant, which anticipates service needs and preemptively organize or perform service needs. The service's task is, to avoid incidents by preemptively reacting and evading them.

CPSs are the foundation for smart services. They provide a framework for the functionality needed to aggregate and evaluate data. These CPSs will influence the way of how work is conducted in industrial factories.

Herterich, Uebernickel, and Brenner [2015] identified the following impacts of CPSs on manufacturing:

1. **Improvement of equipment**
   The development of future versions of equipment can benefit from leveraging performance data of current versions.

2. **Optimization of equipment utilization**
   Operators of industrial equipment are able to evaluate usage over time and to optimized operations based on historic usage patterns.

3. **Remote control and management of equipment**
   CPSs can receive remote information and commands. This allows the operator as well as the vendor to easily access equipment in remote locations.

4. **Predict and trigger actions**
   Due to continuous evaluation of equipment telemetry and performance data the operator and vendor are able to prevent machine failure. The service activities can be preemptively scheduled in an efficient way.

5. **Remote diagnostics**
   Service agents can remotely access equipment diagnostics and do not need to travel to the machine location.

6. **Optimization of service efficiency**
   With the use of CPSs, field services can be conducted more efficiently. For example a component, which is scheduled to be replaced every six months might – depending on the machine usage – still be in good condition after its expected lifetime. With evaluation of usage data, such replacements can be scheduled based on the actual usage and not only in fixed intervals. Furthermore, field service agents can be remotely supported by experts in back-offices.

7. **Information and data-driven services**
   The gathered data allows for detailed insights and can reveal unexpected information. Equipment operators are able to generate additional value by evaluating data.

1.3 Scenario

Preemptive scheduling of MRO activities requires the vendor to have knowledge of the status of industrial equipment. However, industrial equipment is usually situated on a customer site without the vendor having remote access to it. Therefore, it is necessary to embed connectivity into industrial equipment.

Already today, industrial equipment is producing tremendous amounts of valuable data – both for the customer and vendor. With the data, a vendor would be able to preemptively schedule MRO operations and thus avoid costly downtimes. For example, some sensors of AVL devices can only be calibrated in the global headquarter in Graz if they break. Including shipping, this could mean a loss of the machine's capabilities for up to six weeks.

Naturally, added Internet-connectivity to industrial equipment requires tough security considerations, as status data of industrial equipment comprises a wide range of equipment parameters related to maintenance and thus contain valuable and sensitive data which must be kept confidential by the SMS framework. Such a set of equipment values is referred to as *snapshot* since it represents the status of a device at a certain point in time.

The service needs to distinguish between different stakeholders, which should have access to data. First, the vendor needs access to data to preemptively schedule MRO activities. However, also the customer should be able to obtain a copy of the data which was sent by their industrial equipment to the vendor in order to validate the integrity of the data and do further investigations based on the data (e.g. device utilization). The vendor also needs to have the possibility, to directly communicate with industrial equipment, for example in order to publish firmware updates over-the-air. Consequently, the SMS framework must also provide a "way back" from the vendor to the customer.

The aim of this work is to enable the vendor of a device to be able to gather device information in a secure way as well as the customer to supervise this process by being able to monitor

incoming and outgoing data.

In Figure 1.1 the architecture in AVL before the implementation of the SMS framework is depicted. Equipment is located in field and does not have a connection to transmit data to the vendor. The vendor has some knowledge about devices such as service data records in their existing data collections. However, the data is often not up-to-date. The SMS framework will introduce the possibility to transmit data by linking field devices with the vendor backend.

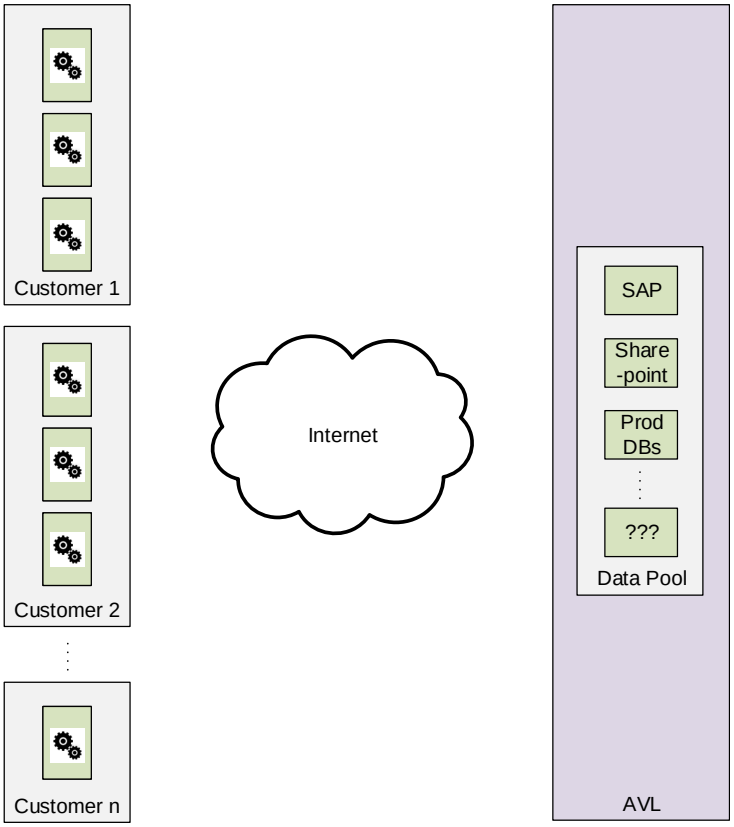


Figure 1.1: The system architecture in AVL before the use of the proposed system. [Drawn by the author of this thesis.]

For the vendor, the following advantages it will have from the SMS framework are anticipated:

- **Utilization Analysis**
  Analyze device utilization and inform customers about saving potentials.
- **Product Improvement**
  Evaluate the gathered data, identify the most error prone components and specifically improve them for succeeding versions of products.

- **Update Rollout**
  Easily roll out device firmware updates remotely.

- **Remote Service Commands**
  Issue service commands such as triggering a cleaning procedure or a reboot to a device remotely.

- **Proactive Maintenance**
  Offer proactive maintenance service by analyzing historical data and identifying potential machine failures before they even happen; thereby increase customer satisfaction.

- **Remote Diagnosis**
  A vendor's back-office employee should be able to conduct remote diagnosis and device investigation.

The customer is able to benefit in the following ways:

- **Proactive Maintenance**
  At AVL, maintenance intervals are currently typically selected by service technicians by "rule of thumb" [Priller, Aldrian, and Ebner, 2014]. However, according to Priller, Aldrian, and Ebner [2014], "service needs of devices typically depend on multiple factors in addition to operating time, and are often influenced by operating conditions like temperatures, gas flows, peak load, load distribution"; leading to maintenance being scheduled sometimes too early and sometimes too late. With smart estimation based on data gathered in the SMS framework, the maintenance can be specifically arranged based on the device's condition.

- **Remote Monitoring**
  Remotely monitor device status and issue commands, no need to be in front of the device.

- **Utilization Analysis**
  Analyze device utilization and investigate on downtimes.

- **Automatic Reordering**
  Automatic, preemptive ordering of necessary device consumables (e.g. lubricants) based on equipment status data evaluation.

One can see, that the SMS framework not only has potential to improve the device utilization and maintenance itself but can also contribute to cost savings on both the vendor and customer side.

1.3.1 Requirements

In this section, only the identified requirements of the SMS framework, which are dealt with in this thesis are listed.

Regarding data transmission in the SMS framework, the following requirements were identified:

- **Withstand network downtimes**
  The devices in the SMS framework will not always have a working network connection. Nevertheless, they should still be able to receive all relevant information. They should also be able to publish data, which was generated during downtimes, once a network connection is available again.
- **Minimal protocol overhead**
  It is supposed, that target devices are often operating in restricted areas, where bandwidth and network speed is limited. Also, some use cases might require data transmission in a frequencies of one Hertz or even beyond. Thus, it is a requirement to keep the overhead necessary for data transmission at a minimum.
- **Routability to specific network entities**
  Network entities must be able to send their messages to specific other network identities, which have expressed interest in the data. Thus, a routable transmission solution is needed.
- **One-to-many, one-to-one, many-to-one support**
  In the scenario, different communication patterns are used. Multiple entities should be able to receive and process status data from industrial devices. Industrial devices should be able to receive data and commands from different entities, for example the vendor and the device owner.

The following security requirements were identified:

- **Authentication**
  Only authenticated clients should be able to use the SMS framework and transmit information via its infrastructure.
- **Authorization**
  Only authorized clients should be able to read messages. Customers should never be able to gain access to information of other customers.
- **Message authentication and Non-Repudiation**
  Recipients of messages should be able to verify the authenticity and its origin. The originator must not be able to repudiate having sent the message.

- **Data Integrity**
  The recipient must be able to verify the message's integrity and – if any changes occurred to the data while in transit – reject processing it.

- **"Firewall Friendliness"**
  The data connection between the customer and vendor should be "firewall friendly" for the customer. This means, that it should ideally only use a single port and establish connections from the inside and not vice versa. Furthermore, to guarantee that firewalls can follow and track sessions, data should be transmitted on top of Transmission Control Protocol (TCP).

- **Use established standards**
  It is of utmost importance, to convince customers of the security of the SMS framework and the safety of their data. After all, they will be transmitting very sensitive and secret information via the SMS framework. It is a goal, to only use well established standards and not to implement own cryptographic methods.

- **Ability to revoke access for devices**
  If a customer device gets stolen or an advisory gets access to its data store, the vendor should be able to revoke the device's access rights. Sometimes devices will change locations or be resold to different customers. Then, the old customer should not be able to access any device information anymore.

1.4 Proposed Solution

Bases on the elaborated requirements, a data transmission system via a central network linking entity on the vendor's premises is proposed.

During the design of the system, special emphasis should be placed on data security. Not only while data is in transit but also while it is processed by network entities on the path between an originator and recipient. Thus, end-to-end encryption should be applied between network entities.

It needs to be considered, that data from industrial equipment is interesting for multiple different stakeholders, such as the customer and vendor. The applied cryptographic functions thus need to be capable to encrypt data for multiple recipients simultaneously.

Although the goal of this thesis' work is to implement a prototype, it is needed to structure the system in such a way, that it could be moved to a fully productive system without major changes in the system architecture.

1.4.1 Scope and Limitations

Since the implementation of the whole SMS framework is a task for multiple project partners, the focus of this thesis is limited to specific parts of it. The focus of this thesis is on the transmission of data and the backend services.

This thesis does not deal with the data collection or connectivity on the customer device. Furthermore, the implementation of a fully scalable productive SMS framework is also out of scope of this thesis.

To sum up in one sentence: *This thesis focuses on how data can securely be transmitted from an industrial device to multiple stakeholders and show a prototypical implementation of how to do so.*

1.5 Research Questions

Based on the scenario and the specified requirements, the following research questions (RQs) in this thesis are dealt with:

- **(RQ1) Which technology can be used, to centrally aggregate and distribute data from globally distributed clients?**
  Research on the technologies being used in IoT applications today is made. There, it is specifically focused on the variety of transmission protocols and its features. Ways of how to aggregate and distribute data on a global scale for a vendor, which has field clients all over the world are proposed. Specifically, the technology should fulfill the requirements for Machine-to-Machine (M2M) communication in an IoT scenario.

- **(RQ2) How can established standards be used to secure data transmission in the SMS framework?**
  The devices in the SMS framework will be globally distributed. For comprehensive analysis data needs to be aggregated at a central point. One has to transmit this highly sensitive industrial data via insecure channels such as the Internet. Thus, high emphasis is placed on the security of the data while in transit. For security functions, it is desired to make use of established industry standards with a broad acceptance among potential customers. Additionally, wide spread security functions and implementations are well tested. This should help to avoid making security mistakes in cryptographic implementations.

- **(RQ3) Implement a prototype to proof the elaborated concepts and show a working data transmission from customer devices to the vendor's backend using the elaborated concepts**
  In order to proof the feasibility of this work, a prototypical implementation of a SMS

framework using the proposed technologies and features is done. The goal is, to do the breakthrough of data transmission all the way from a field client via the Internet to the vendor's backend. Furthermore, the backend service which handles data processing and storage on vendor side is implemented. Project partners do support on customer device side and implement all necessary functionality there.

1.6 Methodology

This work is affected by the structure of the *Arrowhead* project: the project was divided in generations in which the target was, to have a working prototype at the end of each generation.

Thus, the methodology of *evolutionary prototyping* was applied during the research of this thesis. Here, the system evolves over time and new features are added in each iteration. At the end of each iteration, there should be a working prototype which demonstrates the features implemented in the preceding iteration. However, it is not the goal to have a solution ready for use in a productive environment at the end of each iteration.

In the first of all iterations, a check of the existing IoT landscape is made, the available tools and features. No ready system was found, which would fulfill all of the project's needs. In the next step investigations on the available data transmission protocols for IoT are performed. Multiple protocols in terms of their applicability for the scenario and their security features are tested. In parallel to the data transmission research existing security features, such as authentication and authorization, for IoT are investigated on.

The prototype is also incrementally implemented: In the very beginning, it is started with a simple foundation for the data transmission system. Then, simple security features are applied to the system. These security features are incrementally improved until the final prototype for this project with hybrid data encryption is reached (Chapter 4).

To show the practicality of the system, typical showcases are elaborated on and they are developed in parallel with the prototype (Section 5.7).

1.7 Scientific Contribution

Partial content of this work has already been disseminated in the following peer-reviewed publications:

- Lesjak, Hein, Hofmann, et al. [2015]
- Maritsch, Kittl, and Ebner [2015]
- Maritsch, Lesjak, and Aldrian [2016]

- Lesjak, Bock, et al. [2016]

Additionally, contributions to Chapter 10 “Application System Design - High Security” of [Delsing, 2017] have been made.

Some sections and figures of this thesis are based on and reuse contents from these publications. References to these sources are not always made explicit.

1.8 Thesis Structure

Chapter 2 describes the current *State of the Art* in concepts relevant for the scenario. A summary on transmission protocols for IoT in Section 2.2 and specifically on the Message Queue Telemetry Transport (MQTT) protocol (Section 2.2.1) and its relevant concepts is made. In Section 2.3 the current methods regarding security are described. Specifically, Transport Layer Security (TLS) and certificates are explained in Section 2.3.1 as well as end-to-end encryption in Section 2.3.2.

In Chapter 3 the architectural design of the data aggregation framework based on MQTT is shown. It is discussed how to structure the data in transmission (Section 3.2). Next, the broker placement for the publisher-subscriber architecture (Section 3.3) is elaborated on.

Next, in Chapter 4, the approach to securing the data aggregation framework is shown. Special emphasis is placedon TLS (Section 4.2), authentication (Section 4.3), authorization (Section 4.4) and end-to-end encryption (Section 4.5). An implementation of a Topic Access Control System (TACS) for MQTT in Section 4.4.1 is shown as well.

In Chapter 5 the specific implementation of functionality in the backend system is described. After showing the dataflows in the SMS framework architecture (Section 5.2), the implementation and configuration of components is demonstrated. These include the configuration of the MQTT message broker in Section 5.3, the MQTT subscriber (Section 5.4), the end-to-end cryptography service (Section 5.5) and the design of a database schema for data collection in Section 5.6. To proof the working of the prototypical implementation, exemplary workflows which are based on the aggregated data in Section 5.7 are shown.

Chapter 6 discusses the implementation which is made based on this research. It is elaborated on the advantages and disadvantages of decisions which are made regarding the architecture and implementation.

In Chapter 7 finally, the raised research questions are evaluated, the lessons learned are concluded and an outlook is given on the next steps to be performed.

Chapter 2

State of the Art

" The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so. "

[Kevin Ashton, British technology pioneer]

The emerging of the IoT quickly obtained attention from many organizations and companies. Thus, the technical standards and implementations in the field evolved quickly.

This chapter describes the most important concepts which are used for the implementation. First, the connection between the SMS framework and a SOA is discusses. Next, data transmission in the IoT in Section 2.2 is elaborated on. MQTT and some of its features important for the use case are specifically described in Section 2.2.1. Finally, Section 2.3 describes concepts, which are relevant for the authentication and authorization features in the system.

Parts of this chapter are reproduced from the publications (Maritsch, Kittl, and Ebner, 2015 as well as Lesjak, Hein, Hofmann, et al., 2015).

2.1 service-oriented architecture (SOA)

A SOA is a paradigm of distributed systems. It consists of different *services*, which provide functionality. It is not important, how the functionality is achieved; thus, the service can be imagined as a blackbox. The service can be used independently and it has a defined interface specifying how to communicate with it. The SOA also involves a *service registry*, where services can register and offer their functionality. There, the services can be queried and used by *consumers*. Optionally, there can be an *authorization system*, which manages access to services [The Open Group, 2009, pages 1–2].

According to Priller, Aldrian, and Ebner [2014] one can portray the SMS framework as a kind of SOA: The vendor will host a *service registry*, where customer devices can register and offer services. An exemplary service is the provision of a snapshot. A *consumer* can then use the services provided by the devices, like the snapshot provision, and process the gathered data. Another service would be the firmware update provision service by the vendor. It will offer this service in the registry and interested clients can consume this service. Additionally, an authorization service is needed, to only allow authorized devices to consume specific services.

Figure 2.1 shows a basic SOA with a publisher, consumer, registry and authorization service. The arrows depict the interactions inside the SOA.

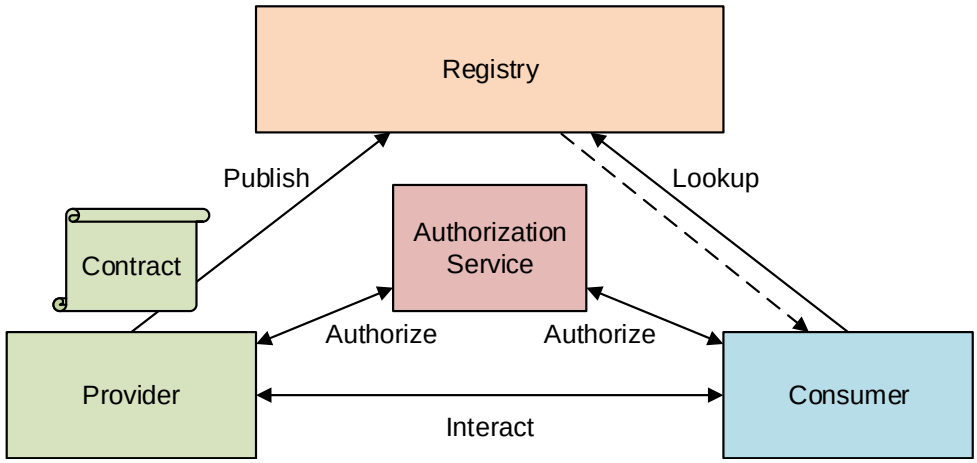


Figure 2.1: Basic SOA consisting of a service *publisher*, *consumer*, *registry* and an *authorization service*. The publisher provides a service contract to the registry. [Drawn by the author of this thesis.]

2.2 Data transmission in IoT

When speaking of the IIoT one is usually talking of *things* with very limited computing power (e.g. devices in sensor networks). In order to integrate these devices into a system, several properties are desirable for transmission protocols to be used:

- **Scalability**
  In order to support a large amount of clients in a network, the protocol should allow for simple scalability; both in terms of the amount of connected clients and the messages being transmitted via the network.

- **Efficiency**

The protocol should transmit payloads as efficiently as possible and only require minimal overhead for transmitting the payload.

- **Portability**
  As there can be devices with different architectures and operating systems involved, the protocol should be portable to all of these devices, regardless of their specific technology.

- **Simplicity**
  Since IoT devices may have only limited computing power, it is of advantage, if the protocol is kept simple so that it is easy for clients to compose and parse messages.

- **Failure tolerance**
  Due to their use in restricted environments, IoT devices may be unable to be reached sometimes. Also, the network might experience some downtimes. The transmission system should cache messages until a network connection is available again.

Before looking at some specific protocols, it is needed to distinguish between two basic patterns of communication in Internet protocols:

- **Request/Response**
  Communication in these kinds of protocols is based on the principle, that a client has to make a request in order to get information from another entity. The other entity would not publish information without a prior request. A prominent example would be Hypertext Transfer Protocol (HTTP), where a client needs to make a request to a web server. Only then, the web server will return information in a response. But the server would never initiate a data exchange.

- **Publish-Subscribe**
  In a publish-subscribe pattern, the communication is bidirectional. Every entity in the system can initiate data exchange. Network entities can subscribe for desired messages and publish messages to other clients, which have expressed interest. Clients can publish information without being requested for doing so. This communication pattern provides loose coupling; subscribers might not be aware of publishers and vice versa and subscriptions can occur before publications (for example “What is the temperature tomorrow”). Since no data is transmitted without explicit subscription, the approach provides a good protection against spam messages and Denial-of-Service attacks. MQTT is a prominent publish-subscribe-pattern based protocol.

Over the last decade, many protocols were declared to be made for IoT. The following paragraphs describe a few of them.

Extensible Messaging and Presence Protocol (XMPP) The Extensible Markup Language (XML)-based protocol XMPP [RFC6120] (previously known as *Jabber*), is a client-server protocol with a decentralized architecture. It allows for one-to-one as well as many-to-many communication. The connection between two XMPP client is established via at least one server.

Constrained Application Protocol (CoAP) CoAP [RFC7252] aims to be a lightweight substitution for HTTP. It works in a request/response manner and offers the well-known HTTP methods such as `GET`, `POST`, `PUT` and `DELETE`. Thus it is easy to interchange existing HTTP-based solutions with it. It is mainly designed for M2M communications but it can also be used in communication between machines and general Internet and in communication between machines over Internet. As the name implies, it is specifically designed to work in constrained networks that are lossy or low-powered. For transportation, it uses User Datagram Protocol (UDP) by default but can also be run over other transport protocols such as the reliable TCP. CoAP messages are encoded in a simple binary format and the protocol adds a fixed four byte header to each message. Furthermore, it has built-in discovery methods in order to allow clients to query servers for functionality they offer. In terms of security, bindings to the Datagram Transport Layer Security protocol are available. However, it only offers one-to-one communication.

MQTT The MQTT protocol is exhaustively described in Section 2.2.1.

2.2.1 The Message Queue Telemetry Transport (MQTT) protocol

With the hype of the IoT, many dedicated protocols evolved. One that emerged quickly and gained broad acceptance was MQTT.

In 2014, it became an Organization for the Advancement of Structured Information Standards (OASIS) standard in version 3.1.1 [MQTT Version 3.1.1]. Then, in 2016, MQTT gained a boost in recognition when it finally became an International Standards Organization (ISO) standard [ISO20922].

MQTT is a binary publish-subscribe protocol, which was specifically developed with respect to M2M communication. Due to its structure, which offers a dramatically reduced protocol overhead compared to other protocols (see Table 2.1 for details), it is specifically useful for communication in environments with restricted network access, limited bandwidths as well as low available computing power. Additionally, MQTT distinguishes from other protocols with its infrastructure: based on a publish-subscribe-model (see Figure 2.2) messages are exchanged between senders and recipients via a central unit: the *MQTT message broker* (referred to as

broker throughout this document). An example of the described publish-subscribe architecture with the broker can be seen in Figure 2.2.

Furthermore, the MQTT protocol specifies the building of a hierarchy of message brokers (*bridging*). To give an example, in a system a dedicated broker can be installed at every production location, serving as a bridge between the local machines and a central master broker.

Since MQTT has a huge user base, many implementations of the protocol are available for all major programming languages. Moreover, many different broker implementations exist – commercial (e.g. *HiveMQ*[1], *IBM Websphere*[2]) as well as open-source (e.g. *mosquitto*[3], *Moquette*[4], *Apache ActiveMQ*[5]). In Table A.1 in Appendix A the features of different open-source and commercial MQTT broker implementations are evaluated.

Table 2.1 compares some important attributes of MQTT with two of the most widely spread protocols on top of TCP/Internet Protocol (IP): HTTP and Simple Mail Transfer Protocol (SMTP). Contrary to for example HTTP, MQTT is a data-centric protocol. It does not care which type of content is transferred and every payload will simple be treated as a sequence of bytes.

The unsuitability of HTTP for IoT applications, where small payloads of data are transferred is also described by Yokotani and Sasaki [2016]. They show the significant differences in data overhead between HTTP and MQTT.

MQTT is a publish/subscribe based protocol. Thus an implementation of various communication patterns, such as one-to-one, one-to-many or many-to-one is possible. This allows for a wide covering of use cases in the SMS framework.

2.2.2 Features

MQTT provides various features, which make it very useful for the use in an IoT scenario. The most important features for the use case are described in the next sections.

2.2.2.1 Topics

The categorization and clustering of messages is made available by the MQTT concept of *topics*. They offer the possibility, to manage messages in a hierarchy; comparable to how it is in a forum. Thus, subscribers are able to specifically state their interested topics and receive messages for them. Topics are treated as a hierarchy, using a slash (/) as a separator. Figure 2.2 shows a simple MQTT topic structure.

[1] `www.hivemq.com/`, Accessed: April 23, 2017

[2] `www.ibm.com/software/products/en/appserv-was`, Accessed: April 23, 2017

[3] `www.mosquitto.org`, Accessed: April 23, 2017

[4] `github.com/andsel/moquette`, Accessed: April 23, 2017

[5] `activemq.apache.org`, Accessed: April 23, 2017

	MQTT	HTTP	SMTP
Transport protocol	TCP/IP	TCP/IP	TCP/IP
Security	optional (TLS)	optional (TLS)	optional (TLS)
Differentiation between subscribing and publishing clients	yes	no	no
One-to-many/many-to-one communication	yes	no	yes
Simple integration of new clients	yes	no	no
Quality of Service (QoS)	yes (at most one, at least one, exactly once)	no	no
typical protocol overhead	2 bytes + variable header ($\approx$ 3 bytes)	200 - 2000 bytes (typical 500 bytes)	$\approx$ 1000 bytes

Table 2.1: Comparison between MQTT, HTTP and SMTP.

In a multi-level topic hierarchy, single levels are divided by forward slashes (/). Thus, exemplary MQTT topics could be the following:

- `temperature/roomA`
- `factoryA/hallB/device567/operatinghours`

Wildcards In a topic hierarchy, publishers can use wildcards to subscribe to multiple topics at once. MQTT specifies two wildcards: the single-level wildcard (+) as well as the multi-level wildcard (#). While the single-level wildcard represents one arbitrary topic level, the multi-level wildcard represents arbitrarily many topic levels. Thus, the multi-level can only be used as the last character in a topic subscription string (e.g. `temperature/#` but also #).

To continue the example from above, a client interested in all data from *factoryA* would subscribe to the MQTT topic `factoryA/#`. A client interested in operating hours data from all devices, would need to subscribe to `+/+/+/operatinghours`.

However, `factoryA/#` and `factoryA/+` would not represent an equal subscription, since for example `factoryA/hallB/device567` will not be matched by the latter subscription while it will by the first.

Figure 2.3 depicts several examples and possibilities of wildcard subscriptions.

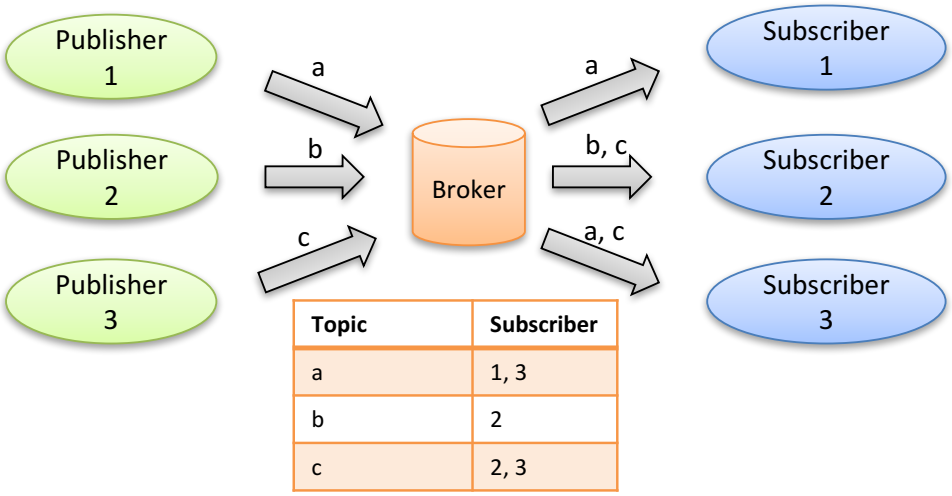


Figure 2.2: A simple example of how subscribers and publishers can make use of MQTT's topic system. [Drawn by the author of this thesis and Thomas Ebner.]

2.2.2.2 Quality of Service (QoS)

Another feature of MQTT is Quality of Service (QoS). Every message can be provided with a QoS level, which tells entities in the system how to deal with its delivery. Depending on the QoS level of a message, different treatments regarding its delivery are intended (refer to Table 2.2).

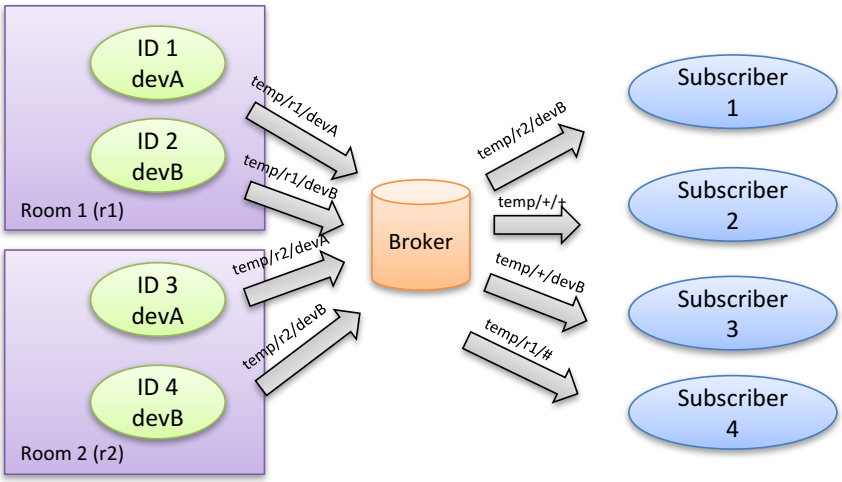
QoS level	Delivery	Delivery guarantee
0	maximal once	no guarantee; "best effort"
1	minimal once	guaranteed delivery; duplicates possible
2	exactly once	guaranteed delivery; no duplicates possible

Table 2.2: QoS levels specified by the MQTT standard.

QoS 0 corresponds to the *Fire & Forget* principle: as soon as a message has left the client, it is dropped. In contrast, when using QoS levels 1 or 2, the client must wait for an acknowledgment, that the message has been delivered (at least) once. Only after receiving the acknowledgment, the sender is allowed to drop the message. However, higher QoS levels mean more network traffic and higher latencies, since several control and acknowledge packets need to be exchanged between the entities in the system. But since MQTT was specified and developed as a lightweight protocol and not all applications do require guaranteed delivery, the choice of the QoS level has been left over to the user.

Figure 2.4 shows a network protocol analysis made with *Wireshark*[6]. Subfigures 2.4a, 2.4b

[6] `www.wireshark.org`, Accessed: April 23, 2017



Subscriber	Topic	Messages from device ID
1	temp/r2/devB	4
2	temp/+/+	1, 2, 3, 4
3	temp/+/devB	2, 4
4	temp/r1/#	1, 2

Figure 2.3: This advanced MQTT topic example shows the publications and subscriptions to topics with more levels. In the fictional setup, two rooms (*Room 1* and *Room 2*) are used with two devices each (*Device A* and *Device B*). These devices measure some temperature and publish it into their specific topic. On the subscriber side, there are clients with interests in different partitions of the data: *Subscriber 1* subscribes to messages from a specific device, *Subscriber 2* subscribes to messages of all rooms and devices, *Subscriber 3* subscribes to messages from all instances of *Device B* and *Subscriber 4* subscribes to messages from all devices in *Room A*. [Drawn by the author of this thesis and Thomas Ebner.]

and 2.4c show the packets being transfered for QoS 0, QoS 1, and QoS 2 respectively. For the test, a publicly accessible MQTT broker[7] was used.

2.2.2.3 Persistent Session

MQTT specifies persistent sessions, which is a kind of a contract between a client and the broker. This contract instructs the broker, to keep a client's session alive, also when it disconnects. This means, that while the client is unreachable and does not acknowledge publishes, the broker needs to queue all messages of QoS levels 1 and 2 until the client connects the next

[7] `iot.eclipse.org`, Accessed: April 23, 2017

No.	Time	Source	Destination	Protocol	Length	Info
4	0.181555	172.20.10.4	198.41.30.241	MQTT	105	Connect Command
6	0.360059	198.41.30.241	172.20.10.4	MQTT	70	Connect Ack
8	0.360292	172.20.10.4	198.41.30.241	MQTT	88	Publish Message
9	0.360360	172.20.10.4	198.41.30.241	MQTT	68	Disconnect Req

(a) MQTT publish with QoS 0

No.	Time	Source	Destination	Protocol	Length	Info
4	0.271766	172.20.10.4	198.41.30.241	MQTT	105	Connect Command
6	0.454166	198.41.30.241	172.20.10.4	MQTT	70	Connect Ack
8	0.454519	172.20.10.4	198.41.30.241	MQTT	90	Publish Message
9	0.637658	198.41.30.241	172.20.10.4	MQTT	70	Publish Ack
11	0.638071	172.20.10.4	198.41.30.241	MQTT	68	Disconnect Req

(b) MQTT publish with QoS 1

No.	Time	Source	Destination	Protocol	Length	Info
4	0.201746	172.20.10.4	198.41.30.241	MQTT	105	Connect Command
6	0.363095	198.41.30.241	172.20.10.4	MQTT	70	Connect Ack
8	0.363452	172.20.10.4	198.41.30.241	MQTT	90	Publish Message
9	0.541008	198.41.30.241	172.20.10.4	MQTT	70	Publish Received
11	0.541328	172.20.10.4	198.41.30.241	MQTT	70	Publish Release
12	0.709657	198.41.30.241	172.20.10.4	MQTT	70	Publish Complete
14	0.709801	172.20.10.4	198.41.30.241	MQTT	68	Disconnect Req

(c) MQTT publish with QoS 2

Figure 2.4: MQTT publishes with different QoS levels. An MQTT message to topic `foo/bar` with the content "Hello World" has been sent to the publicly available MQTT broker. The figure shows the flow of messages needed for the completion of the message for the respective QoS levels. The network traffic was captured with the *Wireshark* network protocol analyzer. [All screenshots from *Wireshark* made by the author of this thesis.]

time. Then, all messages will be delivered immediately. The session is identified by the MQTT `clientId` [MQTT Version 3.1.1, Section 3.1.3.1], which must be specified on the client upon every connection attempt.

This feature is extremely useful in the IoT scenario, where devices might not always be online and available for receiving messages from the vendor.

2.2.2.4 Last Will

Another concept relevant for the IoT, which is offered by MQTT, is *Last Will*. During the initial connection handshake, a client can tell the MQTT broker a *last will*, which consists of a message and an MQTT topic, which will be posted by the broker when the connection to the

client is dropped. The cancellation of a connection to a client is detected by the broker when a ping attempt does not succeed or the client intentionally cancels the connection. Thereby, for example a failure of devices in a system can be monitored.

2.3 Security

Understandably, the use case has some strong security requirements (described in Section 1.3.1). A vendor's customers might be exchanging very sensitive data. Leaking of any information may allow unauthorized third parties to draw conclusions and – in the worst case – gain competitive advantages.

Since MQTT does not specify any security features (apart from username and password fields, which can be used by the client during the initial connection procedure), one needs to make use of other methods[8].

First, it is elaborated on the TLS protocol and digital certificates in Section 2.3.1. Then, end-to-end encryption using Cryptography Message Syntax (CMS) in Section 2.3.2 is discussed.

To promote a general understanding, a few terms are described beforehand.

Digital Certificate A digital certificate binds a cryptographic (public) key to an identity and is issued by an authority. The entity holding the certificate is in possession of the corresponding private key. With this key, it can prove its identity to other entities.

Digital certificates can be used to get the public key of a remote identity, and verify the identity. This happens everyday in the world wide web, when a client wants to establish a secure connection to a web server. The server will provide the client its digital certificate and prove its identity. The client will verify the certificate and check, whether it was issued by a trusted authority.

Certificate Chain A digital certificate always has an issuer, which signs it. Certificates can also be self-signed (also known as *Root Certificate Authorities*). Thus, a chain can evolve, where each level represents a different entity. Figure 2.5 shows an exemplary certificate chain. The trustworthiness of a certificate is always assured by the certificate on level above (except for self-signed certificates). The upper certificate is also known as a *trust anchor*. As for the root certificate there is no upper level, it is signed with its own key. Here trust can only be established by the client, if it already knows the certificate and is trusting it.

[8]The very specific details as well as the justifications why certain security algorithms have been chosen for this project are out of scope for this thesis. They have been elaborated during the project and were mainly selected and defined by project partners at the Institute of Applied Information Processing and Communications at Graz University of Technology.

Certificate Revocation List (CRL) A CRL is usually attached to a Certificate Authority (CA) and contains all certificates of the root CA, which had to be revoked. Revocation reasons for a certificate include compromising of key material. The revocation list is usually identified via an Uniform Resource Identifier (URI) in the root CA and made available by the root CA issuer, which is also responsible for its maintenance. CRLs are supported by the X.509 certificate standard [RFC5280].

Public Key Infrastructure (PKI) A PKI is a system for exchanging, issuing and verifying digital certificates. It is built upon the concept of asymmetric cryptography, where a key pair for cryptographic operation consists of a *public* and a *private* key. A PKI is used for managing the public keys. This includes for example issuing or revoking certificates. A simple PKI consists of a root CA, digital certificates for entities in the infrastructure and an optional CRL.

Message Authentication Code (MAC) A MAC can be used to verify a message's authenticity and integrity. It is a value calculated by the originator of a (encrypted) message, transmitted alongside the payload and verified by the recipient. The following two functions allow to calculate and verify a MAC:

$$(mac) = \text{generateMAC}(m, k)$$
$$(valid) = \text{validateMAC}(m, mac, k)$$

generateMAC() generates a MAC *mac* for the message m using a secret key k. The function *validateMAC()* takes as input parameters the same message m, the calculated previously calculate MAC *mac* as well as the secret key k. Iff the calculated MAC equals the transmitted MAC, the message can be considered valid and unaltered.

Authenticated Encryption (AE) AE is a symmetric-key scheme, which allows for the authenticated exchange of encrypted data. In a symmetric-key scheme, the same key will be used for encryption and decryption. One well-known symmetric-key scheme is Advanced Encryption Standard (AES).

An AE schema specifies the following two (deterministic) functions:

$$(m', mac) = \text{aeEnc}(m, k, iv)$$
$$(m, v) = \text{aeDec}(m', mac, k, iv)$$

The function *aeEnc()* encrypts and authenticates a message m using a symmetric key k and an initialization vector *iv*, which is optionally necessary depending on the kind of encryption algorithm used (the Galois/Counter Mode (GCM) requires one for example). The output of this

function are two values: the encrypted message m' and a MAC *mac*, which can later be used to verify the message's integrity.

The function *aeDec()* receives as input the encrypted message m', the MAC *mac*, the symmetric key k and optionally the initialization vector *iv*. The function will decrypt the encrypted message and return the plain message m along with a validity value v which states, whether the message was successfully validated with the MAC *mac* or not.

While the parameters m' and *mac* are public, the key k must not be accessible to unauthorized entities. The optionally required initialization vector *iv* is also public. However, depending on the encryption scheme, for a strong security it must only be used once.

An AE scheme will yield confidentiality, integrity and authenticity at once:

- *Confidentiality* is provided by the encryption of the message. The plain message is not visible.
- *Integrity* is yielded by the MAC. Any change of the message or the MAC lead to the recipient rejecting the message.
- *Authenticity* is given by the MAC. The recipient will be able to detect if the message has been altered while in transport in which case the authenticity of the originator is not given anymore.

Digital Signature A digital signature can be used to verify both the integrity and authenticity of a message. The originator generates a signature for a message, which can be verified by the recipient.

A digital signature scheme consists of two functions:

$$(s) = \text{sign}(m, k_{priv})$$
$$(v) = \text{verify}(m, s, k_{pub})$$

The originator of a message uses the function *sign()* to calculate a signature value s from a message m using a private key k_{priv}. Upon receiving, the recipient can check the validity v of the signature s of the message m with the corresponding public key k_{pub} of the originator.

Key Wrapping This is a cryptographic function, which wraps a key with a symmetric key. This feature is used in hybrid encryption, where some payload is encrypted symmetrically and the symmetric key needs to be securely transmitted alongside the encrypted message. Thus, a key wrapping algorithm consists of the following two functions:

$$(sk') = \text{wrap}(sk, kwk)$$
$$(sk) = \text{unwrap}(sk', kwk)$$

The *wrap()* function wraps the secret key *sk* using a key wrapping key *kwk*. The *unwrap()* function can restore the original secret key *sk* given the wrapped secret key *sk′* and the key wrapping key *kwk*.

Now the question might arise, why would one wrap the symmetric key *sk* with another symmetric key *kwk*? Why not just use *kwk* for encryption? The reason is rather simple: when an entity wants to send encrypted data to multiple recipients, it first encrypts the data with *sk*. As *sk* should be kept secret between the originator and the recipients and not be shared publicly, the originator establishes the key wrapping key *kwk* as a *shared secret* which is different for every recipient. This *kwk* is then used to wrap the symmetric key. The advantage clearly is, that the originator needs to encrypt the data only once for all recipients.

Key Agreement A key agreement scheme is a cryptographic function, which allows two parties to establish a shared secret. A well-known key agreement scheme is *Diffie-Hellman key exchange*. It allows two parties to establish a secret over an insecure channel in multiple communication rounds. However, this may be sometimes difficult when the two communicating parties cannot directly connect or the message is processed asynchronously at the recipient. In such cases, one can make use of a *one pass key-agreement scheme*. Information needs to be sent only from one client to the other so it can construct the same secret. No response or bidirectional communication is necessary.

Cipher Suite For TLS, many different algorithms exist and can be used for cryptographic security functions. To agree on the algorithms to be used, the interacting parties need to agree on them. A cipher suite is a standardized identifier for a set of algorithms being used in TLS. It specifies, which algorithms are to be used for the following functions:

- Message Authentication (e.g. Elliptic Curve Digital Signature Algorithm (ECDSA))
- Key exchange (e.g. Diffie-Hellman)
- Hashing (e.g. Secure Hash Algorithm (SHA))
- Encryption (e.g. AES)

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 is an example of a cipher suite identifier. Its parameters are explained in Section 4.2.

2.3.1 Transport Layer Security (TLS)

The TLS protocol is well known for securing connections in the world wide web and is used whenever an user's web browser connects to a remote server via a secure connection. On a high

level, in this scenario the server typically presents a digital certificate which is then validated by the user's web browser.

In the Open Systems Interconnection (OSI) reference model, TLS acts on the transport layer. This means, that it is separated from the procedures and contents on the application layer (MQTT). From a technical point of view, a *TLS tunnel* between a client and a server is established according to the following procedure [RFC5246, Section 7.3]:

1. **Client:** `ClientHello`
   The client sends a connection request to the server. This request includes a random number to be used later on, a session identifier and a list of supported cipher suites.

2. **Server:** `ServerHello`
   The server responds to the connection request. This message includes a random number, session identifier and a selected cipher suite. If there is no ciphers suite supported by both the client and server an error message is returned.

3. **Server:** `Certificate` (optional)
   The server sends a message containing its certificate (chain). The public key inside the certificate must be supported by the agreed cipher suite.

4. **Server:** `ServerKeyExchange` (optional)
   The server sends a premaster secret to establish a shared key with the client.

5. **Server:** `CertificateRequest` (optional)
   If the client needs to present a certificate as well, the server requests for it with this message. It might include a list of allowed certificate types, CAs and supported signature algorithms.

6. **Server:** `ServerHelloDone`
   This message signalizes to the client, that the server has finished sending relevant messages of the `ServerHello`.

7. **Client:** `Certificate` (optional)
   The client responds to the server's `CertificateRequest` by sending its certificate.

8. **Client:** `ClientKeyExchange`
   In this step, the client sets the premaster secret and transmits any relevant information to the server, for example the parameters for a Diffie-Hellman key exchange. The client and server are not able to compute the *master secret* for the connection. The specific calculation of the master secret depends on the used key exchange algorithm.

9. **Client:** `CertificateVerify` (optional)
   If the client has provided a certificate, it must send a `CertificateVerify` message. This allows the server to verify the certificate ownership of the client. In this message, the client sends a signature of the previous `ClientKeyExchange` message, which it has generated with its private key. The server can verify the signature with the corresponding public key of the client's certificate.

10. **Client:** `ChangeCipherSpec`
    With this message, the client indicates to the server, that it sends encrypted and authenticated messages from now on.

11. **Client:** `Finished`
    The client sends an encrypted and authenticated `Finished` message. The server attempts to decrypt it and verify its MAC. In case any of these steps does not succeed, the handshake was a failure and the connection needs to be dropped.

12. **Server:** `ChangeCipherSpec`
    As above, With this message, the server indicates to the client, that it will be sending encrypted and authenticated messages from now on.

13. **Server:** `Finished`
    The server sends a `Finished` message in the same way the client has done earlier. The client verifies the message internally.

14. *Handshake complete*
    From now on, the parties can exchange application data in an encrypted and authenticated way.

In a system based on an MQTT, the broker plays the role of the remote server. Thus, it has its own certificate for being able to prove its identity to clients. However, as the broker is publicly accessible and only certain devices should be able to establish a connection to it, some sort of authentication mechanism already at the data transport layer is favorable. The TLS protocol also specifies the use of *client certificates* [RFC5246, Section 7.4.6], which allows to also equip entities, which are clients to the broker, with a certificate proving their identity.

Typically, MQTT is transported over TCP/IP. Thus, one can make use of TLS operating on the data transport layer which is beyond the application layer, where MQTT operates. Figure 2.6 visualizes the TLS and MQTT protocol as layers in the OSI reference model.

2.3.2 End-to-end Encryption

Since data of many different parties are processed by the MQTT broker, it needs to be assured, that not only the paths between the clients and the broker are secure but also the application

data while in process.

Thus, a kind of encryption needs to be applied to the application data, which protects it all the way from leaving the originating client until reaching the final recipient processing the data. This is commonly referred to as *end-to-end encryption*. For this purpose, the interacting parties need something in common. In particular, the originator needs to be aware of the recipient as they need to compose the message already initially in such a way, that the recipient is able to decrypt it.

Ideally, the originating party should be able to define a set of recipients, which will be able to decrypt the message. This can be achieved using CMS, which is described in Section 2.3.2.1.

As a bonus, adversaries being able to compromise the MQTT broker and reading the messages being exchanged via it, can not do anything with the data. The only thing they are capable of, is determining the originator and the intended recipients of the message. However, if they do not have access to any of the recipients cryptographic material (in particular the private key needed for the cryptographic operations) they will be unable to decrypt the application data.

Figure 2.7 illustrates an end-to-end encrypted packet being delivered through a TLS tunnel.

2.3.2.1 Cryptography Message Syntax (CMS)

CMS [RFC5652] is a platform-independent standard for exchanging cryptographic data using the Abstract Syntax Notation One (ASN.1) format.

CMS is designed to fit many different purposes and different cryptographic methods. In the use case, it is made use of its authenticated-enveloped-data content type which is suited for *authenticated encryption* functionality as described in [RFC5083]. This allows to *encrypt* and *authenticate* operational data in such a way, that only predefined parties will be able to decrypt it.

As mentioned before, authenticated encryption is a symmetric cryptography scheme that achieves data authenticity and confidentiality using a single, ephemeral symmetric key. CMS provides key encapsulation capabilities to wrap the symmetric key under an ephemeral wrapping key using the *One-Pass MQV, C(1e, 2s, ECC MQV) Scheme* as described by Barker et al. [2013, Section 6.2.1.4].

Simply spoken, this procedure uses features of Elliptic Curve Cryptography (ECC) to wrap the symmetric key needed in the AES-GCM encryption process for each recipient in such a way, that only they can unwrap it. In this process, the Menezes–Qu–Vanstone (MQV) protocol is used for key agreement between the entities.

CMS specifies a range of content types for different kinds and levels of security. One of them is the *Authenticated-Enveloped-Data* content type, which is summarized by [RFC5083] as follows:

> *"The authenticated-enveloped-data content type consists of an authenticated and encrypted content of any type and encrypted content-authenticated-encryption keys for one or more recipients. The combination of the authenticated and encrypted content and one encrypted content-authenticated-encryption key for a recipient is a "digital envelope" for that recipient. Any type of content can be enveloped for an arbitrary number of recipients using any of the supported key management techniques for each recipient. In addition, authenticated but not encrypted attributes may be provided by the originator."*

The *Authenticated-Enveloped-Data* content type is intended for authenticated modes of encryption such as the prior described AES-GCM. For each recipient, the key for AES-GCM encryption is pairwise established in a key agreement process using the recipient's public key and the originator's private key with the Elliptic Curve Menezes-Qu-Vanstone (ECMQV) key agreement protocol.

Besides encrypting content, CMS provides capabilities to include unencrypted but authenticated attributes. This means, that for example an originator could securely pass an unencrypted MAC along the encrypted payload. Since the attribute is authenticated, it cannot be changed while in transport without the recipient getting aware of it.

For the sake of completeness, it should be mentioned, that CMS also supports the encapsulation of unauthenticated attributes. However, these attributes can be changed without the recipient getting aware of it. Thus, these attributes are not used.

ECMQV ECMQV is an one-pass key-agreement scheme [Barker et al., 2013]. It uses three key pairs for operation: the static key pair of the originator, the static key pair of the recipient and an ephemeral key pair. Furthermore, it uses the ECMQV primitive ecmqv() [Barker et al., 2013, Section 5.7.2.3] and assumes, that all keys are derived from the same ECC domain.

It allows an originator O and a recipient R to establish a shared secret in a *single pass*. This means, that there is no bilateral communication between the clients necessary and processing at the originator is timely independent from processing at the recipient.

ECMQV assumes, that both parties are in possession of a static key pair $O_{k_{pub},k_{priv}}$ and $R_{k_{pub},k_{priv}}$. It is assumed, that originator O is in possession of recipient R's public key $R_{k_{pub}}$ and vice versa.

Then, the procedure for key establishment is as follows:

1. Procedure at originator O
   - Generate a random ephemeral key-pair $E_{k_{pub},k_{priv}}$
   - Compute the shared secret Z by calling ECMQV:
     $Z = \text{ecmqv}(O_{k_{priv}}, R_{k_{pub}}, E_{k_{priv}}, E_{k_{pub}}, R_{k_{pub}})$. $R_{k_{pub}}$ *is intentionally used twice.*

2. Transfer the public ephemeral key contribution $E_{k_{pub}}$ from originator O to recipient R.

3. Procedure at recipient R

   - Compute the shared secret Z by calling ECMQV:
     $Z = \text{ecmqv}(R_{k_{priv}}, O_{k_{pub}}, R_{k_{priv}}, R_{k_{pub}}, E_{k_{pub}})$.

Both parties have now obtained the shared secret Z. *Note:* usually Z is not directly used for cryptographic functions such as encryption but a *key derivation function* is applied to Z in order to get a key in desired format.

AES-GCM AES is a cryptographic function, which is used to encrypt and decrypt data with a symmetric key. GCM describes the mode of operation of the encryption/decryption. GCM introduces additional security to AES by using an initialization vector and a counter value for block encryption in AES. The initialization vector is protected by the scheme [Dworkin, 2007].

Figure 2.8 shows and describes the creation, transmission and unpacking of the CMS message. Further information can be found in [Barker et al., 2013, Section 6.2.1.4] for ECMQV respectively in [RFC5084] for AES-GCM in a CMS context.

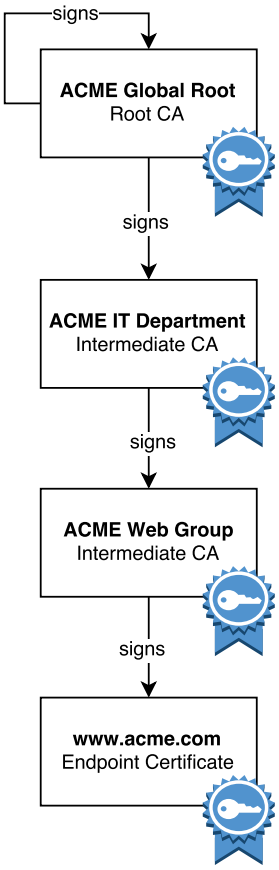


Figure 2.5: This figure illustrates an exemplary certificate chain. A self-signed certificate is used to sign an intermediate certificate. This again is used to sign another intermediate certificate. Finally, an endpoint certificate issued to *www.acme.com* is signed. A web server reachable under this domain would present this certificate to a client and verify that it has the corresponding private key. The client would traverse the certificates' signatures in the chain up to a root. If all signatures are valid and the client trusts the root certificate, it can also trust the endpoint certificate and thus verify, that the entity they are communicating with is really the web server at *www.acme.com*. [Drawn by the author of this thesis.]

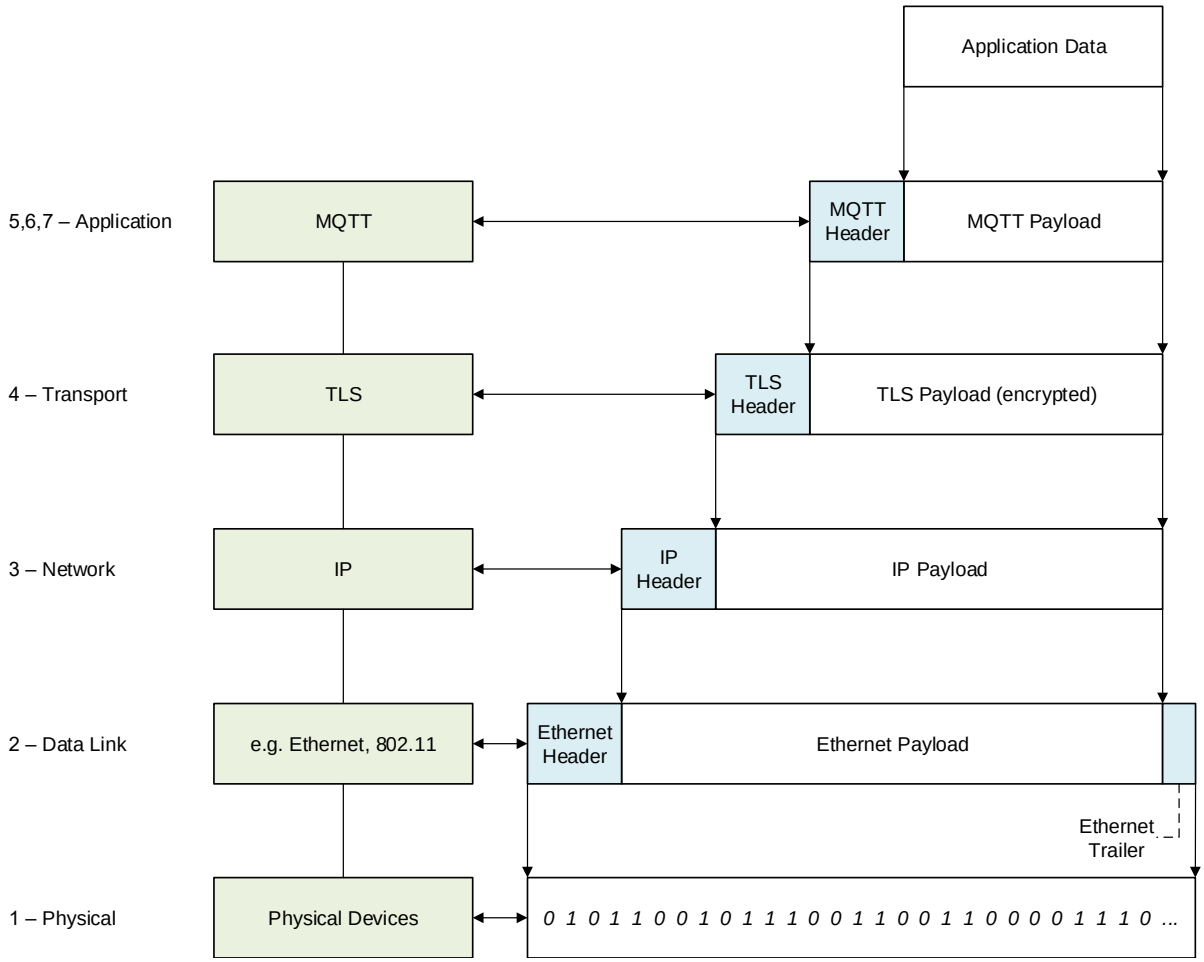


Figure 2.6: MQTT and TLS in the OSI reference model. MQTT works on the application layers and is encapsulated in TLS packets. [Drawn by the author of this thesis.]

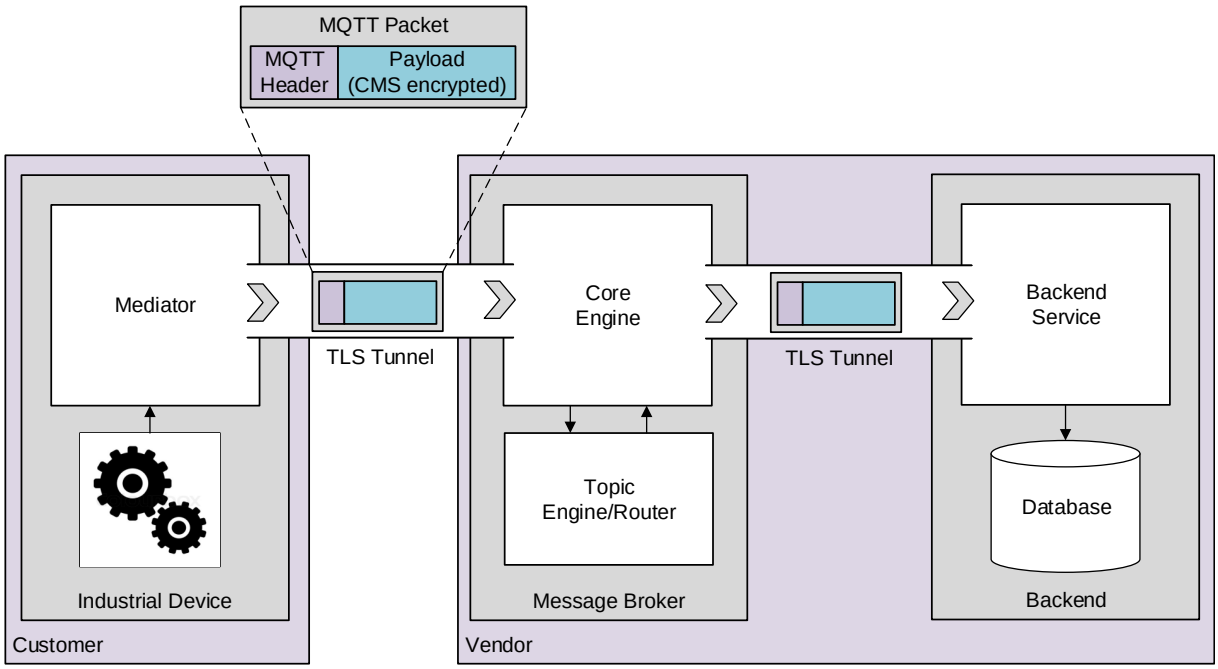


Figure 2.7: This figure illustrates an end-to-end encrypted CMS message as payload of an MQTT packet being delivered through an established TLS tunnel. Although the TLS secured connection is only established between network entities, the data is not available in plain on the message broker since it is secured with an additional layer. [Drawn by the author of this thesis.]

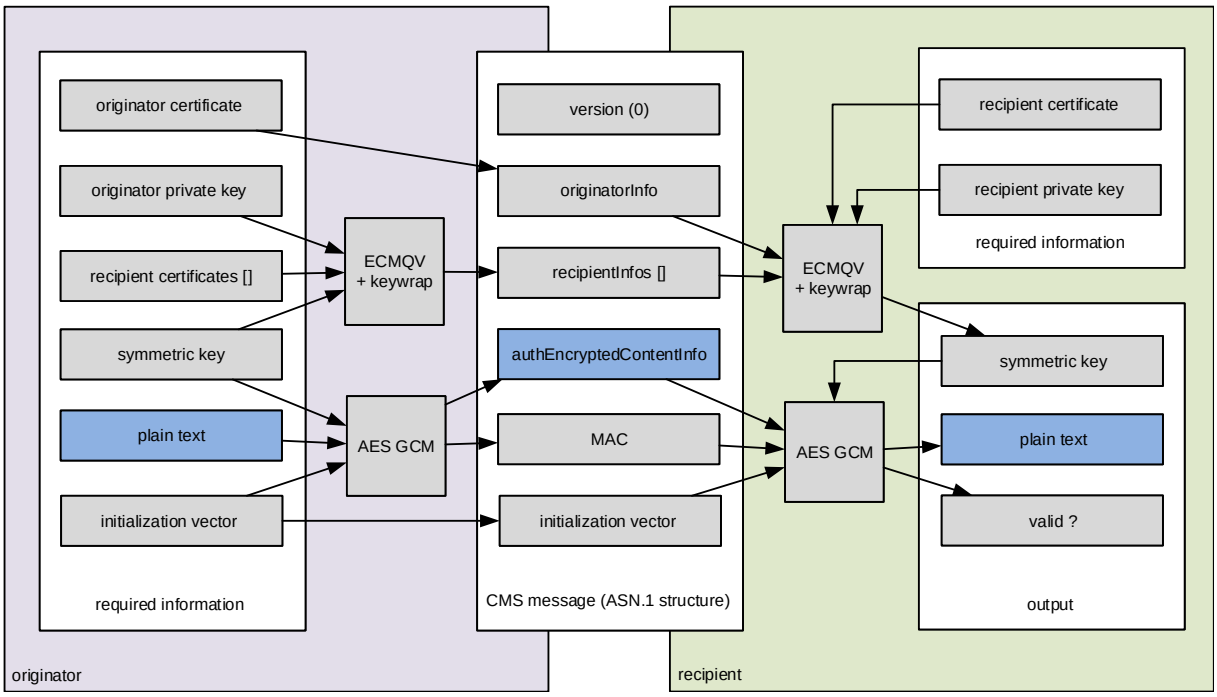


Figure 2.8: This figure shows the abstracted composition of a CMS packet in the use case. The left side shows the encryption of a plain text (colored in blue) and all information, which is required to encrypt the text for particular recipients (identified by their certificates). The resulting CMS message (encoded in ASN.1) is indicated in center. The encrypted text is now encapsulated in the `authEncryptedContentInfo` field (colored in blue). On the recipient again, the information required to unwrap the key and decrypt the text is indicated as well as the results after processing the CMS message. The two engines needed to perform all necessary steps for encryption and decryption (ECMQV and AES-GCM) are here represented as black boxes, as their very detailed mode of operation is out of scope. [Drawn by the author of this thesis.]

Chapter 3

The MQTT Data Aggregation Framework

> *" You can have data without information but you cannot have information without data. "*
>
> [Daniel Keys Moran, computer programmer and science fiction author]

In this chapter, we will show our implementation considerations for the proposed SMS framework regarding data aggregation. Section 3.1 describes the usage of MQTT in our SMS framework. Here, we specifically focus on how to make smart use of MQTT's topic hierarchy capabilities in Section 3.2 as well as the infrastructure of brokers in Section 3.3.

Parts of this chapter are reproduced from the author's publications [Maritsch, Kittl, and Ebner, 2015], [Maritsch, Lesjak, and Aldrian, 2016] as well as [Lesjak, Bock, et al., 2016].

3.1 MQTT

For the data transmission protocol, we chose to use MQTT based on our elaborations documented in Section 2.2.1.

On vendor side, we make use of the open-source MQTT broker implementation *Mosquitto*[1]. It implements version 3.1.1 of the MQTT standard. The broker is implemented in C and C++ and also offers client libraries. Mosquitto has an authentication plugin system, which allows users, to built custom authentication solutions.

Figure 3.1 shows the communication sequence between an MQTT client and a broker. During the initial TLS handshake, the two parties will exchange certificates and verify the respective other. Upon the successful completion of the TLS handshake, the client will initialize the

[1] `mosquitto.org`, Accessed: April 23, 2017

MQTT connection procedure. Afterwards, the client will be able to subscribe and publish messages. For subscriptions, the client will send its topic and QoS desires to the broker, which will acknowledge them. For the publishing of a message with QoS 2, the client will first send the message (`PUBLISH`). If the message is valid, the broker will respond that it has received the publish desire (`PUBREC`). The client requests the broker to release the message (`PUBREL`). Finally, the broker will acknowledge the complete procedure (`PUBCOMP`). This handshake-like procedure guarantees, that the message was successfully received by the broker and the client can verify the delivery. The disconnection procedure of MQTT is unidirectional; the client will send a disconnect message to the broker without having to wait for acknowledgment.

3.2 MQTT topic structuring

In the SMS framework, one goal is to make smart use of MQTT's topic system. We want to achieve a system, which on the one hand allows clients to easily subscribe for intended messages on the broker and on the other hand complies with our security standards.

3.2.1 Topic candidates

For topic structuring, there are a number of suitable topic types. The following (non-exhaustive) list enumerates a number of those potential categories. However it is up to the very specific use case to identify the topic categories to be used.

- `serialnumber`
  The serial number string of an equipment uniquely identifies an instance of a specific industrial equipment.

- `class`
  The class string of an equipment identifies its type, for example a Fuel Meter (FM) or Particle Counter (PC).

- `customer`
  The customer string uniquely identifies a vendor's customer operating equipment instances.

- `premise`
  The premise string identifies the customer's site, on which the customer is operating the industrial equipment.

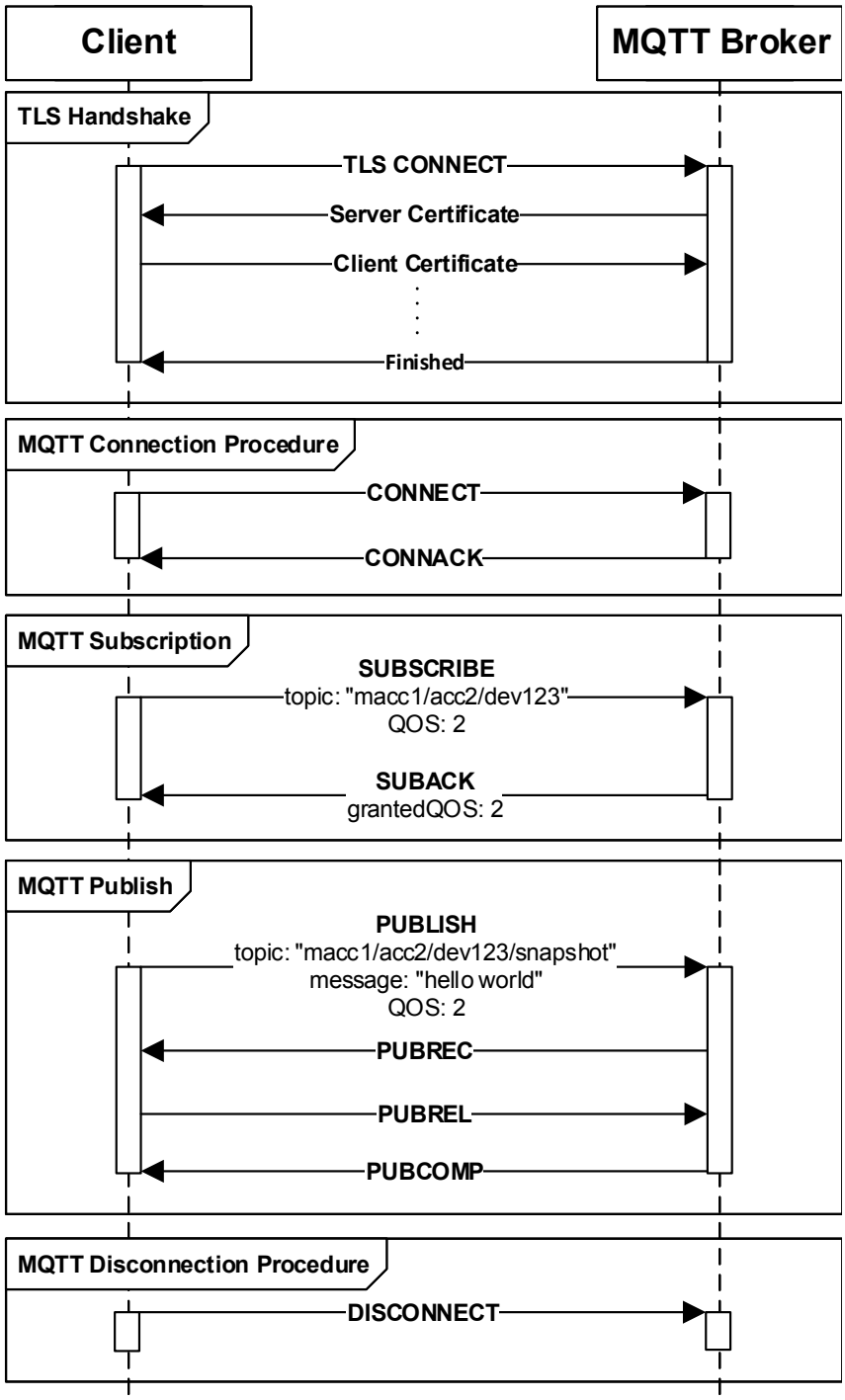


Figure 3.1: This sequence diagram illustrates the procedures to be taken by the client and server in order to establish an MQTT connection and exchange data via a TLS tunnel. For readability purposes, only the most important parameters are shown and some steps in the TLS handshake have been omitted. [Drawn by the author of this thesis.]

- `country`
  The country string identifies the country of a customer's site, in which the customer is operating the equipment. This might be used to structure equipment by vendor subsidiaries which are responsible for certain geographic areas.

3.2.2 Potential topic structuring variants

In this section, we discuss and evaluate a range of different topic structuring possibilities. The goal is, to find the most suitable one for efficient collection of desired data as well as efficient access restriction of MQTT clients on the broker.

1. `/snapshots`
   Being the simplest structure, we would use a single topic to which all clients connect and where data is aggregated. This could be any static string (such as `/snapshots`) or just the MQTT root topic (`/`). However, in terms of data aggregation, this system would be rather unsuitable since all clients of all customers have access to all messages and all messages would effectively be published as broadcasts. Furthermore, topic restriction cannot be applied since client can only get access to the complete topic or not at all.

2. `/serialnumber`
   It would probably be the most reasonable approach to use a unique equipment identifier – such as the serial number – as a topic and allow every client to only access their dedicated topic. This deems to be a fruitful approach where industrial equipment of only one specific type and customer is used. Additionally, clients will only receive messages which have specifically been published for them. Nevertheless, subscribing clients are not able to efficiently group subscriptions. A topic restriction system is also difficult to realize since rules for every single equipment and topic would be needed. It is for example not easily possible without configuration, that a customer gets access to all its devices.

3. `/customer/serialnumber`
   The broker would structure data based on equipment serial numbers and a customer identifier. In this system data can be easily assigned to a customer and thus data aggregation on a per-customer-basis is easily possible. This also allows customers to easily aggregate data for all of their devices.

4. `/class/serialnumber`
   In this scenario, the topic would include an class identifier (such as FM or PC) to the equipment's serial number. This might be useful in case there are multiple different types of industrial equipment using the SMS framework. The topic structure would allow clients to easily aggregate data for certain classes. A vendor's department in charge for all devices of a specific class could for example easily subscribe to data of a specific

equipment class *XYZ* by subscribing to the MQTT topic `/XYZ/#`. However, data can not be aggregated on a per-customer-basis.

5. `/customer/premise/class/serialnumber`
   This approach proposes a deeper and more sophisticated hierarchy than the ones previously mentioned. Thus, it offers a more detailed distinction of industrial equipments, its classes and customers. This approach is specifically useful when there are multiple customer's exchanging data in the same SMS framework. This sophisticated topic hierarchy allows to make full use of MQTT's topic wildcards. For example, if a customer named *ACME* wants to aggregate data for all of their equipment they could simply subscribe to the topic `/ACME/#`. Since we also include premises in the topic identifier they might also restrict their subscription to data of equipment from specific premises. On vendor side, if for example a vendor's department was in charge for equipment of class *XYZ* they could subscribe to the topic `/+/+/XYZ/+` in order to receive data from all industrial equipment of this class. The approach offers various subscription possibilities for all different stakeholders. The subscription possibilities can be used to intelligently aggregate relevant data. Figure 3.2 visualizes this system

6. `/class/customer/premise/serialnumber`
   In fact, this approach has no practical difference to the previous one. It depends on how one wants their data structured and the decision is up to the implementor.

MQTT topics are both routing information and metadata. Thus not only the MQTT message payload needs to be protected but the potentially valuable or sensitive information in the topic information as well. TLS protects topic labels while in transport. However, if an adversary should be able to compromise the broker, the labels will be available to them unencrypted.

In order to not disclose any valuable customer identifiable information on the broker, we decided to replace customer and equipment names with unpredictable identifiers. We use the following values in our final topic structure:

- `master_account`
  An identifier for the overall account of a customer.

- `account`
  An identifier for a specific customer account inside the `master_account`.

- `equipmentid`
  An unique identifier (e.g. a serial number) for an industrial equipment owned by the account.

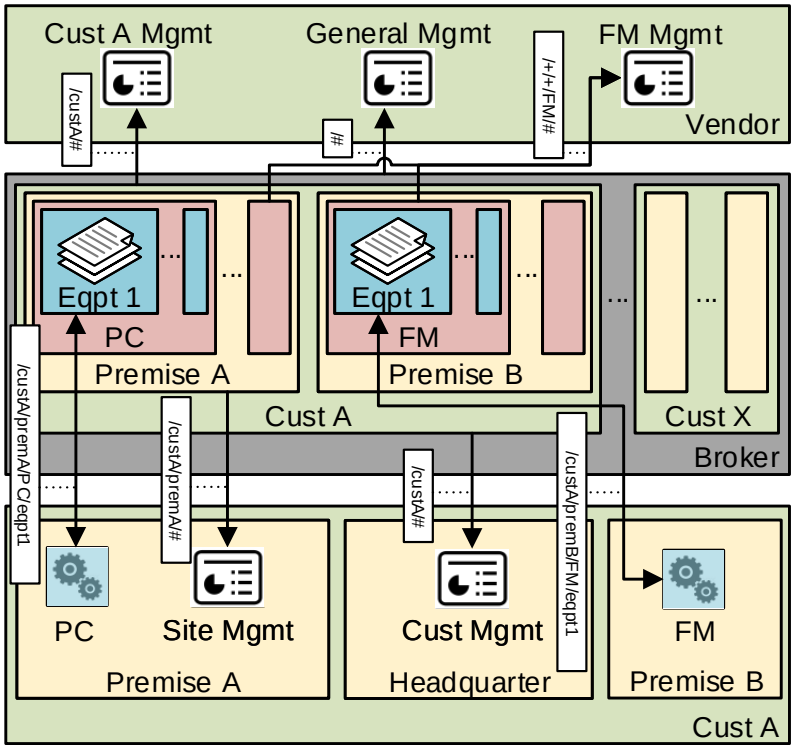


Figure 3.2: Visualization of the topic structure `/customer/premise/class/serialnumber`. The levels in the topic hierarchy represent the *customer*, *premise*, *class* and *equipment* (in this particular order). These topics are divided into different groups in the broker in order to visualize their membership to parent topics and mutual dependencies. On customer side, we have different premises, classes (PC, FM) and management stakeholders. Arrows indicate all possible client connections; the arrowheads indicate the direction of allowed dataflows. The arrow labels describe the MQTT topics which would be used by the clients in this particular scenario. [Drawn by the author of this thesis; adapted from [Maritsch, Lesjak, and Aldrian, 2016].]

The resulting topic structure is as follows: `master_account/account/equipmentid/`. All the sub-topics of such a topic are for the equipment identified by the `equipmentid`. An exemplary used topic could be `000001/00002/EQID00000234/firmwareupdate`.

Using this structure, we have a reasonable trade-off between data aggregation possibilities and information visible to the outside: On the one hand, enough information is available for customers to aggregate and monitor data for all of their devices. On the other hand, not too much sensitive information is visible to the outside. One could only do statistical research, for example of how many snapshots a customer publishes per day. However, all of this information is also hidden to the outside by TLS.

3.3 MQTT Broker Architecture and Cascading

In this section we discuss where to physically place the broker(s). One important aspect is that the broker must be reachable by either customer equipment or another customer-side broker. Thus, one broker instance needs to be placed inside a vendor-controlled Demilitarized Zone (DMZ).

The basis of the following considerations are the SMS use cases, where all equipment snapshots shall be available to the vendor's backend services that enable SMSs.

We have drafted and evaluated four different broker architectures which are discussed in the following.

3.3.1 Single broker on vendor's premises

In this variant for a broker architecture, the only broker is put in a DMZ inside the vendor's premises. All instances of equipment – regardless of the customer owning them – directly publish their data into the central point of data aggregation.

One main advantage of this architectural variant is its extendability. New customers and equipment can easily be added without the need for any additional vendor-side configuration or setup. Thus, there are only initial installation and administration efforts necessary for the single master broker.

However, from a security point of view, data of different customers is not physically segregated. For some customers, this lack of segregation might arouse security concerns since highly sensitive equipment data is transmitted. Consequently, a customer might refrain from allowing the equipment to connect to the Internet without any additional security measures such as strong payload encryption.

In the considered architecture, the broker acts as a Single Point of Failure (SPOF). This means, that in case the broker fails to process messages, clients would have to retry the delivery of their messages. Furthermore, the single central broker requires customers to allow per-device direct connections to the vendor's DMZ for publishing their data. This may represent a security issue for customers striving for minimum outbound Internet connections for the purpose of easier control. They maybe would prefer one central point of aggregation inside their premises which transmits data to the vendor. We elaborate on this in the following sections.

This architecture is visualized in Figure 3.3.

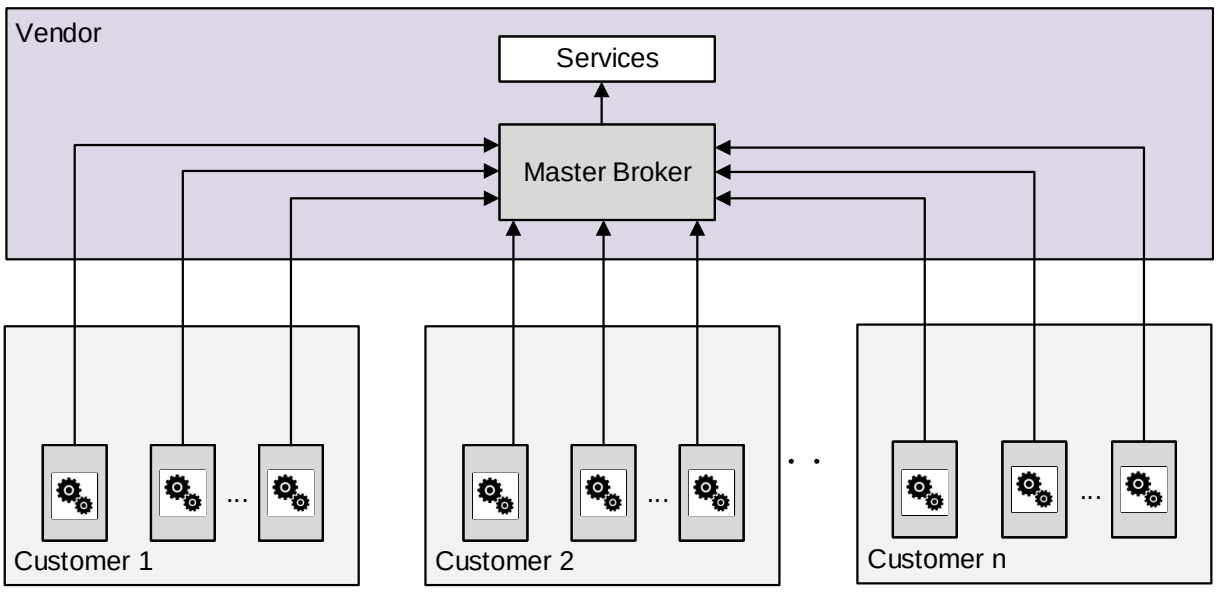


Figure 3.3: Single MQTT broker system. All customer devices need to establish a dedicated connection to the broker inside the vendor's DMZ. [Drawn by the author of this thesis.]

3.3.2 Per-customer-broker on customer's premises plus central master broker

A customer-side security-by-isolation characterizes this approach. Here we use one broker for snapshot aggregation per customer on customer's premises. Thus, data of different customers is physically segregated. One master broker in the vendor's DMZ collects data from all globally distributed customer brokers.

Customers are able to experience several advantages using this approach. First of all, the broker on customer's premises will act as an intermediate broker and fallback solution in case the master broker should be unavailable. Also, the whole system will not be affected by failures of single customer brokers. Customers are able to monitor their data traffic of equipment services at one central point. Lastly, customers only need one network connection to the master broker in the vendor's DMZ.

Nevertheless, the system still does not provide physical segregation of equipment data on a per-customer-basis in the vendor's DMZ. Furthermore, the usage of such a system would also require the customer to set up, operate and maintain their own MQTT brokers. Thus, this might be a solution for customer's with big amounts of equipment but certainly not for smaller ones.

Figure 3.4 visualizes this approach.

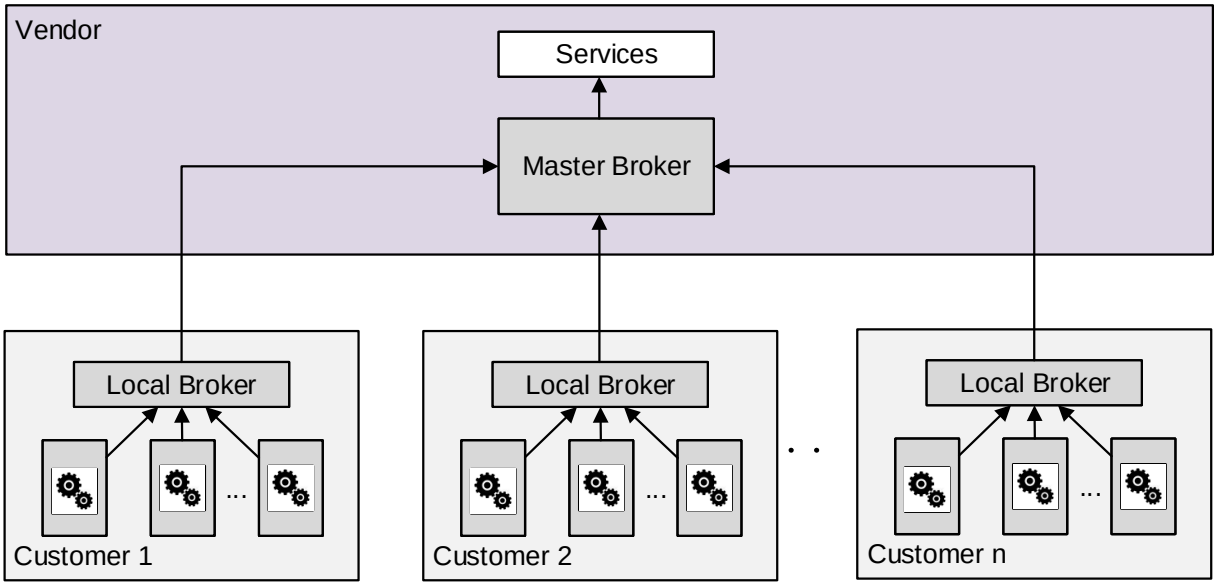


Figure 3.4: Distributed MQTT broker system. The customers have local brokers, to which their devices can connect. Only the brokers need to establish a dedicated connection to the master broker inside the vendor's DMZ. [Drawn by the author of this thesis.]

3.3.3 Per-customer-broker on vendor's premises plus central master broker

In this approach we propose a system using a per-customer point of data aggregation on vendor's premises or more specifically in its DMZ. One master broker in the vendor's backend serves as the point of aggregation for all customer brokers.

In the proposed system data is physically segregated on a per-customer-basis in the vendor's DMZ. Additionally, we do not have a SPOF on vendor side as intermediate customer brokers are able to cache messages in case the master broker should be unavailable. Also, if one of the customer brokers fails, the other customers are not affected.

But this system would require notable infrastructural overheads for the vendor compared to other approaches: one dedicated broker instance for every customer is needed on vendor side. This leaves us with bad extendability of the overall system since a new broker is needed for every customer. As a side note, this architecture might also increase the latency of data acquisition in the vendor backend.

This approach is depicted in Figure 3.5.

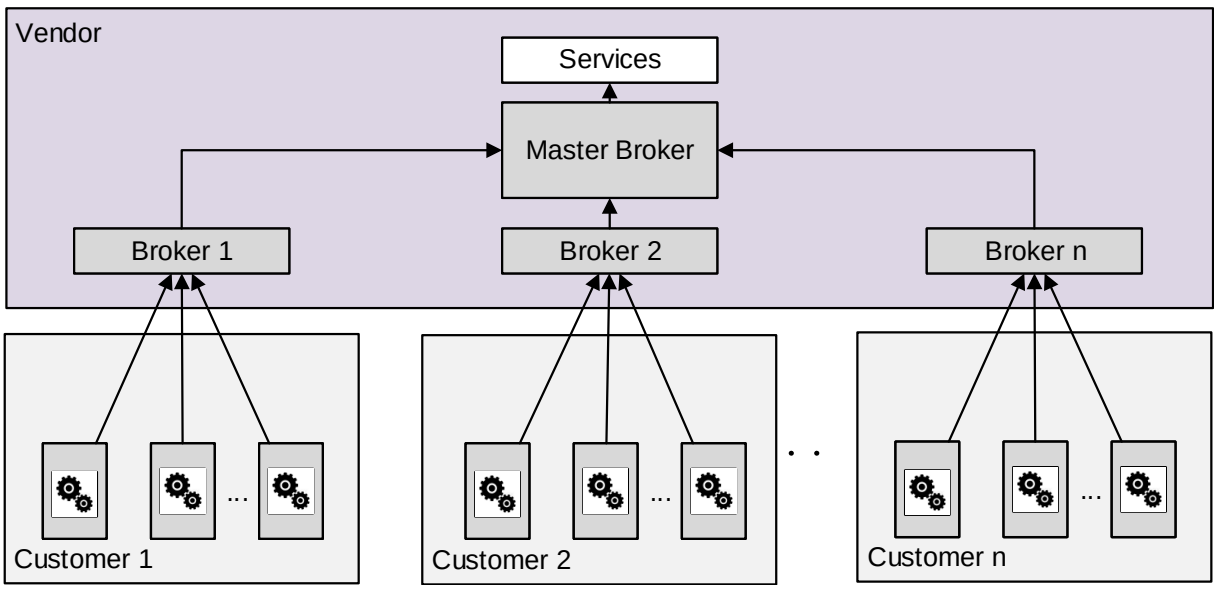


Figure 3.5: Multiple customer MQTT brokers at vendor system. All customer devices need to establish a dedicated connection to their specific customer broker inside the vendor's DMZ. A master broker will aggregate data of the customer brokers. [Drawn by the author of this thesis.]

3.3.4 Distributed vendor brokers plus central master broker

Here we propose to distribute a small number of brokers globally on vendor's premises. One master broker in the vendor's DMZ collects data from all those globally distributed brokers.

One of the advantages is, that there will be shorter latencies between customer and vendor. This might become relevant, since equipment might be operating in restricted areas, where Internet connection is at a premium. In this approach, it might be easier for a customer or rather its equipment to set up a stable connection to a geographically close broker of the vendor.

Nevertheless, the vendor is still left with the task of aggregating data of all globally spread brokers at a central point. This might be not as much of a difficulty as it seems, since the corporate network of the vendor will usually be well established between different locations. However, the architecture we propose here might also increase latencies of data acquisition in the vendor backend where it is processed.

Figure 3.6 illustrates this architecture.

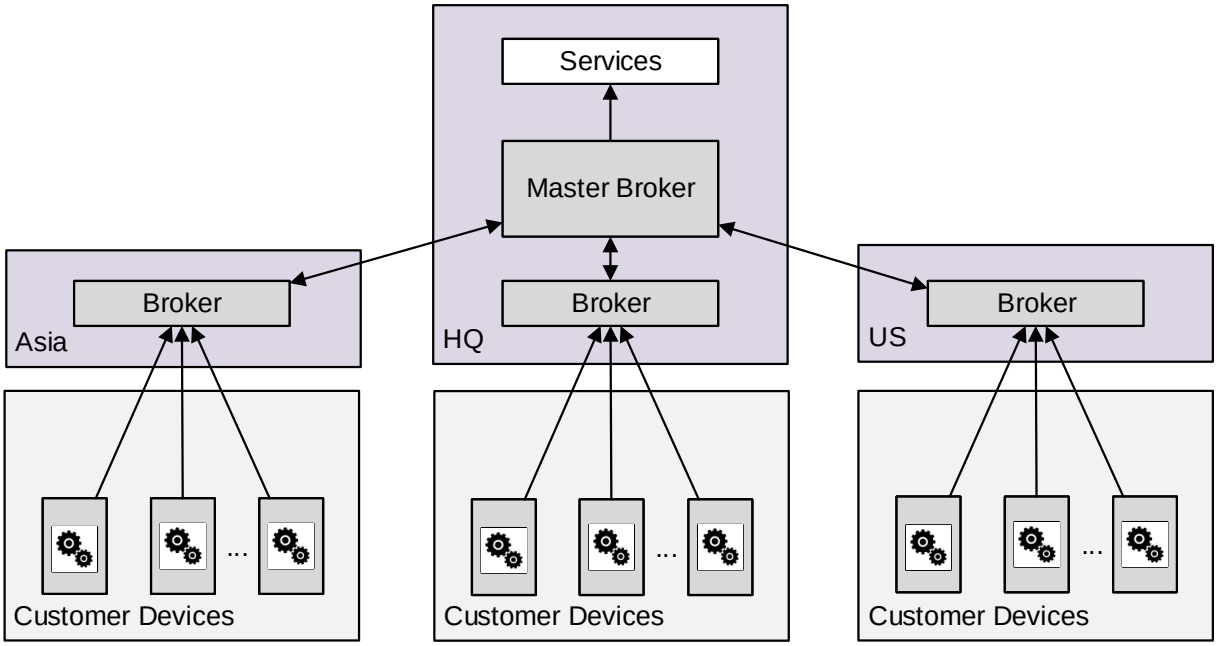


Figure 3.6: Globally distributed MQTT brokers system. Customer devices need to establish a dedicated connection to the respective geographically closest broker. A master broker will aggregate data of the globally distributed vendor brokers. [Drawn by the author of this thesis.]

Chapter 4

Securing the SMS Framework

> *“ New threats will emerge through 2021 as hackers find new ways to attack IoT devices and protocols, so long-lived "things" may need updatable hardware and software to adapt during their life span. ”*
>
> [Nick Jones, vice president and distinguished analyst at Gartner]

Regarding security, it is first proposed how to use TLS in the SMS framework in Section 4.2. It is shown how authentication (Section 4.3) and authorization features (Section 4.4) based on a TACS (Section 4.4.1) is implemented in the broker. Besides that, it is outlined how end-to-end encryption can be achieved in Section 4.5.

Parts of this chapter are reproduced from the author’s publications [Lesjak, Hein, Hofmann, et al., 2015], [Maritsch, Lesjak, and Aldrian, 2016] as well as [Lesjak, Bock, et al., 2016].

4.1 Overview

In general, two levels of security are applied, which results in a *hybrid encryption*. First, data is secured on the transport layer by standard TLS which is described in Section 4.2.

However, TLS does not only secure data while in transport between different network components in the SMS framework. When data is processed – for example on the broker – the application data is not protected anymore by TLS. Thus, data is additionally encrypted from end to end between target entities in the SMS framework. This security mechanism is described in Section 4.5. Figure 4.1 shows the encryption levels of data on its way through the SMS framework.

For the purpose of hybrid encryption, three different mechanisms required in the use case are elaborated on: *authentication* in Section 4.3, *authorization* in Section 4.4 and *end-to-end data encryption* in Section 4.5.

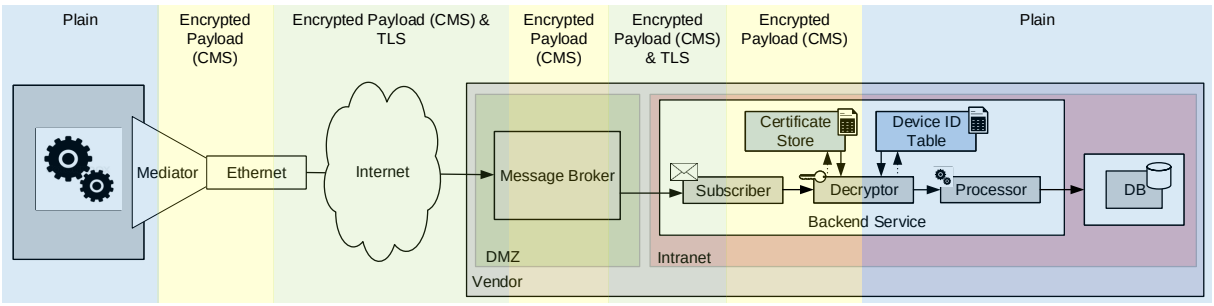


Figure 4.1: Security levels of the dataflow. The blue shade represents parts of the dataflow, where data is processed in plain. The yellow shade illustrates stages of the dataflow, in which end-to-end encryption has been applied to the data. The green shade represents the state of hybrid encryption, where data is end-to-end-encrypted and transferred via a TLS tunnel. [Drawn by the author of this thesis.]

4.2 Transport Layer Security (TLS)

Regarding the TLS connection between client devices and the message broker, the *TLS_ECDHE_ECDSA_WIT* cipher suite is used, which utilizes ECC for key agreement and signatures. The cipher suite is – on a high level – explained here:

- *TLS*
  - Transport Layer Security protocol
- *ECDHE*
  - Elliptic curve Diffie-Hellman exchange
  - Anonymous key agreement protocol
  - Used to establish a shared secret over an insecure channel
- *ECDSA*
  - Elliptic Curve Digital Signature Algorithm
  - Sign messages with ECC keys
  - 521 bit key length

- *AES256*
  - Advanced Encryption Standard
  - 256 bit key length
  - Used for the encryption of transferred information
  - AES is a symmetric encryption algorithm and due to its simpler mathematic operations compared to asymmetric encryption much faster. Also, many hardware modules have dedicated AES engines.
- *GCM*
  - Galois/Counter Mode
  - Provides data authenticity and confidentiality
  - Calculates a MAC while encrypting with AES which is used for authenticity verification by the opposite party
- *SHA384*
  - Secure Hash Algorithm
  - 384 bit key length
  - Hashing of messages
  - *Note: SHA384 is used for TLS, as the more secure SHA2 hashing function (512 bit key length) is not yet standardized for TLS*

4.2.1 Key Management

For hybrid encryption, it is needed to distribute quite some keys in the SMS framework. For easier handling of those keys, digital certificates are used which are part of a PKI.

This allows, to use the following features of a PKI, which are useful for the use case:

- **Identity verification**
  By checking the signature of a certificate, a client can easily verify the authenticity of the remote party.
- **Key revocation**
  Using CRLs, certificates – and thus identities – which can not be considered secure anymore can be revoked. This comes in handy when a key gets compromised, a device is stolen or is moved from one customer to another.

- **Hierarchical structuring of identities**
  As described in Section 2.3, certificates can be used to sign other certificates. A hierarchical structure for all entities in the SMS framework can be implemented, which allows for an easier authentication and classification of the entities.

The customer-device architecture in the SMS framework can easily be represented in an X.509 certificate hierarchy. A root CA is used to sign *master account* certificates. These *master account* certificates can be used as intermediate CAs and sign account certificates. Finally, an *account* can also serve as an intermediate CA and sign device certificates. Device certificates do not need and will not be allowed to sign any other certificates further down the chain. The root CA is also configured to allow a maximal path length of 2 for validation, in order that the customer does not introduce any other intermediate CAs.

Figure 4.2 shows an exemplary implementation of a certificate hierarchy, which can be used in the system. The hierarchy was designed to comply with the MQTT topic structure proposed in Section 3.2.

Certificate Revocation Lists (CRLs) Distribution points of these CRLs are usually provided by the root CA issuers. They are commonly offered via HTTP and its URIs are incorporated in the root CA certificates. However, in the SMS framework, devices are only allowed to make MQTT connections and block every other connection attempts. Thus, the approach is to provide the CRLs via MQTT to the devices in timely fixed intervals.

4.3 Authentication

For authentication purposes, every client in the system, needs an unique identifier, which can be validated by the entity granting or denying access.

When having a look at the world wide web, these identifiers are commonly (unique) combinations of an username and a password. Other authentication methods include one time tokens (e.g. in online banking services) or simple numerical combinations (e.g. disarming an alarm system).

Another approach is, to equip every client with their own, unique certificate, proving their identity. These certificates can then be used for the establishment of the TLS tunnel as well as the authentication in the MQTT broker.

Using the same certificates as with TLS brings along another major advantage: when the broker starts to operate on the MQTT layer, the certificate and its values have already been verified during the establishment of the TLS tunnel one layer beyond. Thus, when the communication reaches the MQTT layer, the broker can safely assume, that the client at the remote end of the tunnel really is the client described by the identity in the certificate.

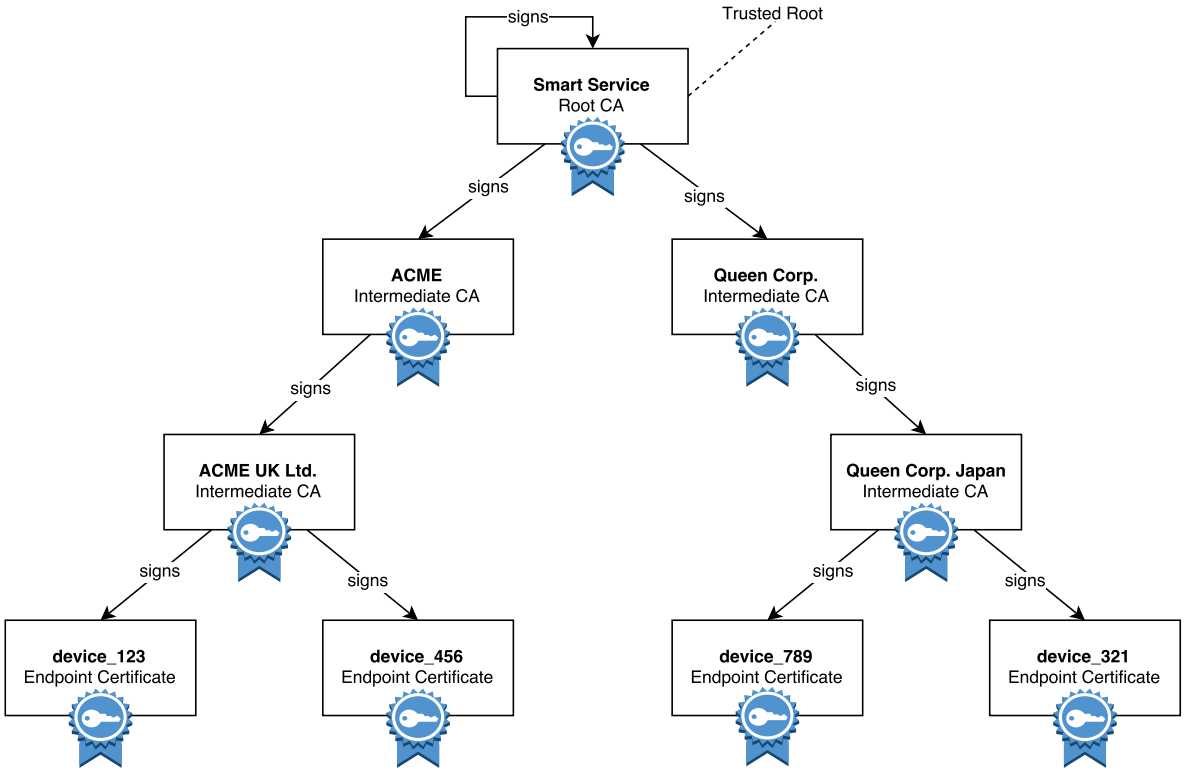


Figure 4.2: An exemplary certificate hierarchy in the system. The root CA must be trusted by all entities in the SMS framework. Every *master account* get their own intermediate CA. These again are used to sign another level of intermediate CAs for *accounts*. Finally, the *account* certificates are used to sign the endpoint certificates of customer devices. [Drawn by the author of this thesis.]

The identifier is given by the *Subject* field's value in the X.509 certificate [RFC5280, Section 4.1.2.6], which identifies the entity associated with the public key. It is required upon the initial connection to the MQTT broker and verified by the broker's TLS module.

4.4 Authorization

The authorization again is also based on the client certificate's identity.

When connecting to the broker, a client first needs to authenticate by sending its whole certificate chain to the trusted root which is anchored in the broker; i.e. the root CA. Once the TLS tunnel is established, the client sends MQTT commands to publish or subscribe to specific topics. The broker authorizes the client's desires based on the *Subject* value in the client's certificate.

In the configuration the MQTT topics are assumed as namespaces. This means, that clients are granted access to all topics in their subtree. Thus, a *master account* client is able to access topics for all of his *accounts* and *devices*, while an *account* is only able to access information for its *devices* and *devices* is only able to access their own information (see Figure 3.2).

Table 4.1 shows some examples of access requests, which is allowed or denied by the broker.

This control mechanism inside the broker is called *Topic Access Control System (TACS)*.

Certificate Subject	Topic Access Request	Allowed
ms-acc-1	`ms-acc-1/#`	✓
ms-acc-1	`ms-acc-1/acc-1/dev-1/snapshots`	✓
ms-acc-1	`#`	✗
ms-acc-1/acc-1	`ms-acc-1/#`	✗
ms-acc-1/acc-1	`ms-acc-1/acc-1`	✓
ms-acc-1/acc-1	`ms-acc-1/acc-1/#`	✓
ms-acc-1/acc-1	`ms-acc-1/acc-1/dev-1/snapshots`	✓
ms-acc-1/acc-1/dev-1	`ms-acc-1/acc-1/dev-1/#`	✓
ms-acc-1/acc-1/dev-1	`ms-acc-2/acc-1/dev-1/#`	✗
ms-acc-1/acc-1/dev-1	`ms-acc-1/acc-1/dev-2/snapshots`	✗
ms-acc-1/acc-1/dev-1	`ms-acc-1/acc-1/#`	✗
ms-acc-2	`ms-acc-1/#`	✗
ms-acc-2	`ms-acc-2/acc-1/#`	✓
ms-acc-2/acc-1/dev-2	`ms-acc-2/acc-1/dev-2/#`	✓
ms-acc-2/acc-1/dev-2	`ms-acc-2/acc-1/dev-1/#`	✗

Table 4.1: MQTT topic access examples.

4.4.1 Implementation of a Topic Access Control System (TACS) in *Mosquitto*

In Section 3.2 it is described, how the topic structuring possibilities of MQTT are utilized.

However, it is also needed to ensure, that only authorized clients are able to write into and read from topics, which are intended for them.

Using the topic structure `/master_account/account/identifier` (proposed in Section 3.2) in combination with the certificates used in the TLS client authentication process, a straightforward TACS can be implemented. Under the assumption, that the certificate and all its values are valid – which has already been verified in the authentication process – the values specified in the certificate chain's *Subject* fields [RFC5280] can be used.

```
1 # grant access rights to all topics to clients
2 # using a certificate with identity 'Root'
3 user readwrite Root
4 topic #
```

Listing 4.1: Simple Mosquitto ACL example

Mosquitto ships with two different Application Programmer Interfaces (APIs) for implementing a TACS. These systems apply restrictions based on the MQTT username – which in this case is not in use by clients. Thus, the Mosquitto broker has been configured to use the client certificate's *Subject* field value as MQTT username.

Figure 4.3 shows a basic overview of what the TACS does.

4.4.1.1 Authentication Plugin

An alternative to implement a TACS based on an Access Control List (ACL) is to provide a plugin, which is implemented as an external module and is integrated by Mosquitto as a shared library. The module must implement specific callback functions, which can validate subscriptions and publications of clients.

4.4.1.2 Source Code Modification

Alternatively – as Mosquitto is an open-source MQTT broker implementation – a TACS may be also directly integrated into the broker source.

4.4.1.3 Access Control List (ACL)

When using this approach, it is needed to specify a list containing topics, in which allowed topic accesses are denoted in a specific syntax. For example one can include an entry

```
pattern write devices/%u/snapshots
```

for general restrictions applying to all clients, where `%u` is substituted by the client username – the certificate's *Subject* field value in this case. This would restrict publishing clients to only be able to publish into their very own topic. One can also define exemptions from general rules for specific users. The configuration in Listing 4.1 grants a client, which is identified as *Root* – i.e., the *Subject* field's content of the client's certificate equals the string "Root" – access to all topics (using the multilevel wildcard).

As this is a rather straight-forward approach and it is possible to cover the needs with it, it was chosen to implement the TACS based on an ACL. The final working version of the ACL

configuration is shown in Listing 4.2. Since `%u` is substituted by the identity of the connecting client, it grants access to the subtree of topics relevant for a client. There are several exceptions from this basic rule:

- **Grant an administrator access to all topics**
  If a client's identifier equals "MRO-Admin", the broker grants access to all topics. This client identity is useful for testing and maintaining purposes. Due to its privileges it needs to be carefully used and stored.

- **Grant an administrator access to system relevant topics**
  The Mosquitto MQTT broker implementation provides a special set of topics under the topic `$SYS`, which offer system relevant information lie for example the number of connected clients or the amount of memory used. This data will *not* be included with a wildcard subscription (#) but needs the specific `$SYS` prefix to be accessible. For maintenance reasons, the administrator can also access these topics.

- **Grant the snapshot processing service access to all snapshot topics**
  The snapshot processing service needs to be able to subscribe to snapshot topics of all devices of the SMS framework. Thus it is granted read access to `+/+/+/snapshots`.

- **Grant the firmware updating service access to all firmware topics**
  The firmware updating service needs to be able to publish to firmware topics of all devices of the SMS framework. Thus it is granted write access to `+/+/+/firmware`.

4.5 End-to-end Encryption

To transport application data in an *authenticated* and *encrypted* manner, CMS' *Authenticated-Enveloped-Data Content Type* is used as described in [RFC5083] and elaborated on in Section 2.3.2.1.

The end-to-end encryption in the SMS framework adds an additional layer of security. While data is being protected when in transport with TLS, CMS protects data on application layer. It enables the originator of a message to be in full control to restrict the potential readers of the message.

The contents of the CMS structure are the following:

- **Originator Info**
  This part describes the originator of the CMS message. The specification allows different types of identities to be inserted here. These include for example the complete certificate, only the public key or the (unique) pair of certificate issuer and version number. In the

```
# grant every client in the system access rights to their subtree of topics
pattern readwrite %u/#

# grant admin access rights to all topics
user MRO-Admin
topic readwrite #

# grant admin access rights to Mosquitto-specific topics (load info, ...)
user MRO-Admin
topic readwrite $SYS/#

# grant vendor-snapshot-service read rights to snapshot topics
user vendor-snapshot-service
topic read +/+/+/snapshots

# grant vendor-firmware-service write rights to firmware update topics
user vendor-firmware-service
topic write +/+/+/firmware
```

Listing 4.2: Final Mosquitto ACL configuration

SMS framework the whole certificate chain is used as an originator info. Thus, recipients are able to verify the validity of the certificate without any additional information necessary.

- **Recipient Infos**
  For each recipient, the CMS message contains an information object. It contains an identifier of the recipient, which is able to unwrap the encrypted key (c.f. Section 2.3 *Key Wrapping*). The recipient is e identified by issuer and serial number of its certificate. The structure also includes all material to establish the key wrapping key for the symmetric key (c.f. Section 2.3 *Key Agreement*). Finally, the wrapped key is included.

- **Encrypted Content Info**
  This structure contains the identifier of the CMS type (`authEnvelopedData`) as well as the encrypted content and the algorithm used for encryption.

- **Authenticated Attributes**
  The optional authenticated attributes field contains a set of key-value attributes. The timestamp, at which the originator has signed the message, is included.

- **MAC**
  The final part contains the MAC over the message as calculated by the originator. The recipients can use it, to verify the message's authenticity.

Figure 4.4 summarizes the contents of the CMS structure. The entire data structure is being transferred as the payload of an MQTT message to recipients.

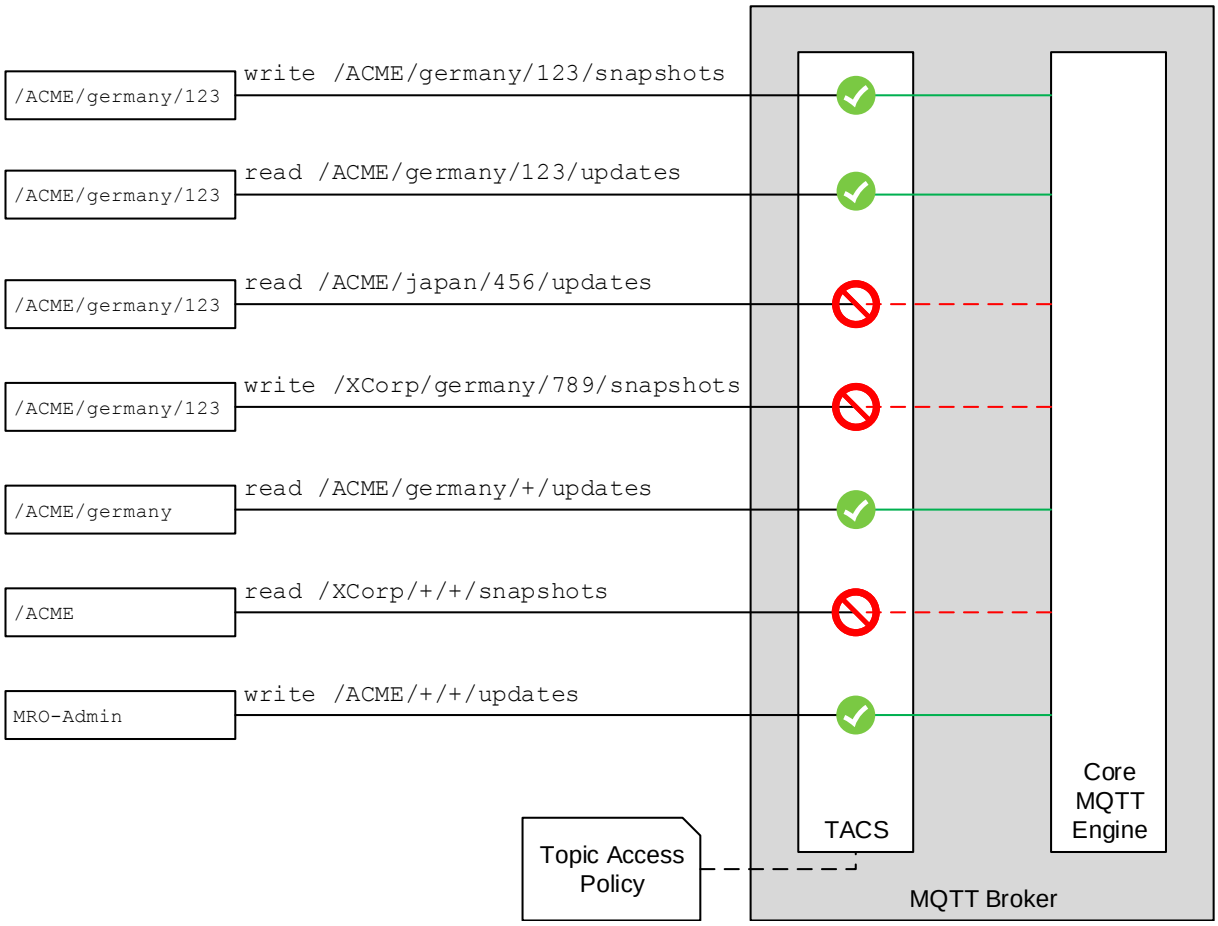


Figure 4.3: Basic overview of the TACS implemented for *Mosquitto*. The rectangles on the left-hand side indicate clients requesting read or write access to topics. The descriptions in the rectangles are the identity of the client as proven by their identity certificate. Upon finding a topic access request match in its predefined topic access policy, the TACS grants access to the topic. Otherwise access is denied and the topic access request is rejected. For the purpose of this example it is assumed, that the certificates used by clients have already been validated and the clients are authenticated against the broker. [Drawn by the author of this thesis.]

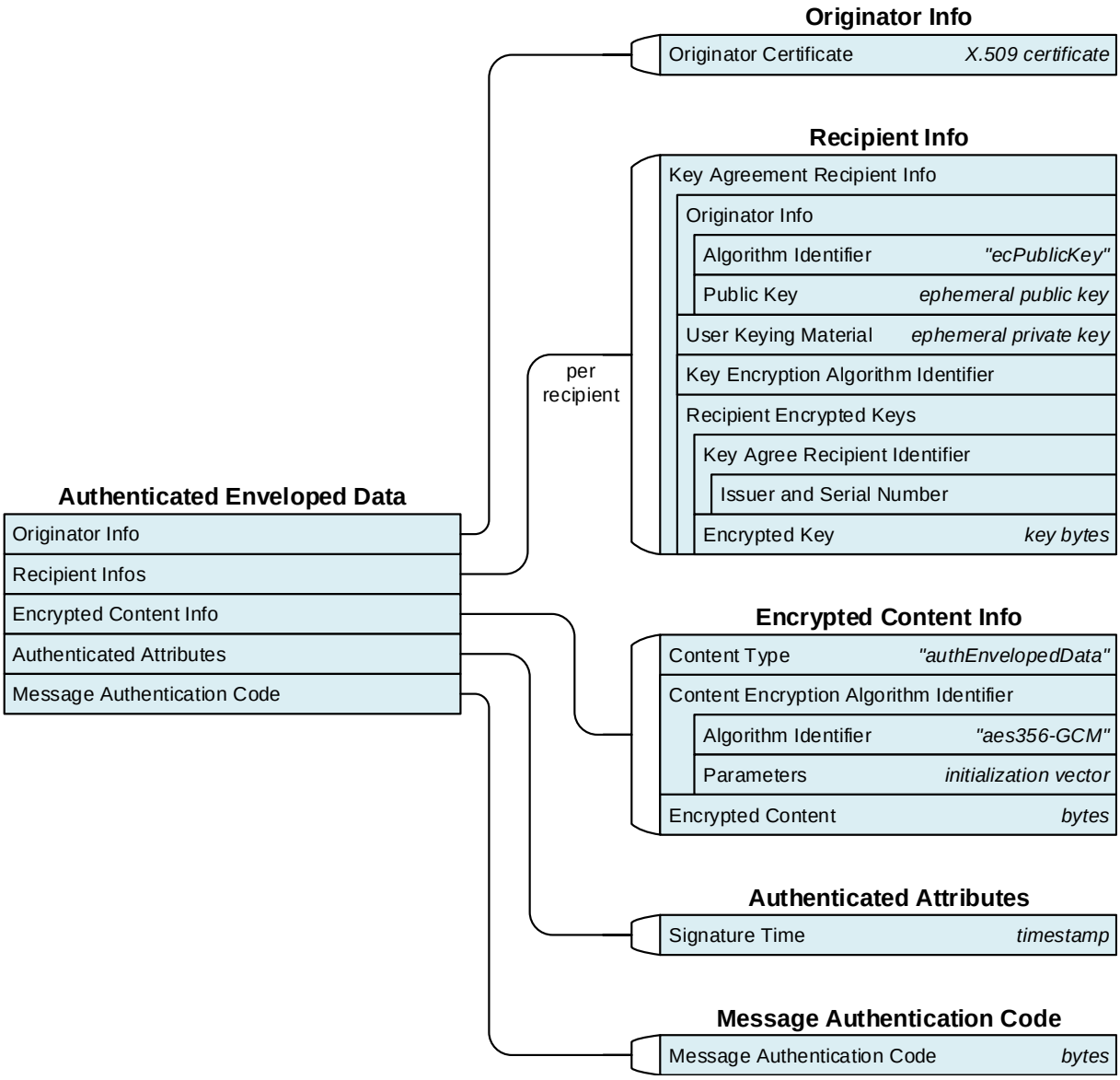


Figure 4.4: Structure of a CMS message with authenticated enveloped data. [Drawn by the author of this thesis.]

Chapter 5

Prototypical Backend Implementation

> *" The IoT is removing mundane repetitive tasks or creating things that just were not possible before, enabling more people to do more rewarding tasks and leaving the machines to do the repetitive jobs. "*
>
> [Grant Notman, Head of Sales and Marketing, Wood & Douglas]

This chapter describes the prototypical implementation of the SMS framework backend. In Section 5.3 it is elaborated on how to set up the MQTT broker to comply with the security concept. While laying focus on the MQTT subscriber in Section 5.4 and on the CMS decryptor in Section 5.5, the elaborated database schema is also described in Section 5.6.

In Section 5.7 finally, exemplary workflows are described, to get a first impression of what can be achieved with aggregated device data.

Parts of this chapter are reproduced from the author's publications [Maritsch, Lesjak, and Aldrian, 2016] as well as [Lesjak, Bock, et al., 2016].

5.1 Introduction

During research, a prototype is implemented to proof the concepts for snapshot aggregation in the SMS framework. The components of the prototypical implementation are illustrated in Figure 5.1.

This data aggregation service involves different components: in Section 5.2 an overview of the dataflows in the SMS framework is given with special focus on the backend. Section 5.3 describes the setup and configuration of the Mosquitto MQTT broker. Next, in Section 5.4 the implementation of the MQTT subscriber in the data processing service is explained. The CMS

decryptor is shown in Section 5.5. Finally, in Section 5.6, the relational data schema which was elaborated for the storage of aggregated snapshots is shown.

For the technical realization of the implementation *Java*[1] is used.

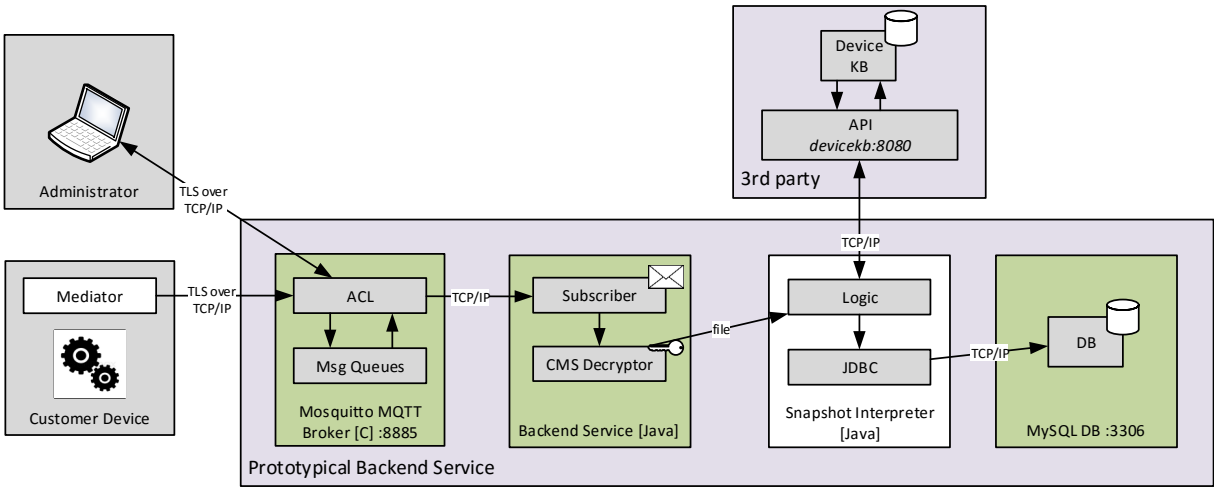


Figure 5.1: Here the components of the prototypical backend implementation are illustrated. The components with light green background have been implemented/set up during the work on this thesis. Other components were provided by other project members. [Drawn by the author of this thesis.]

5.2 Dataflows

To promote a general understanding of communication and data exchange in the SMS framework, the dataflows inside the system are shown. From an abstract point of view, one is able to observe two different ways of communication: On the one hand, transferring equipment information such as snapshots from customer devices to the vendor backend. This is usually an one-to-one way of communication. However, industrial equipment might also be distributed to customer clients which yields an one-to-n connection. On the other hand, the vendor might distribute data from its central backend to field devices in a one-to-one way; for example for signed firmware updates. Data might also be distributed in a one-to-n way; for instance security policy issues.

Figure 5.2 shows the flow of incoming data inside the vendor controlled DMZ and backend.

Figure 5.3 shows the flow of outgoing data inside the vendor controlled DMZ and backend.

[1] `java.com`, Accessed: May 6, 2017

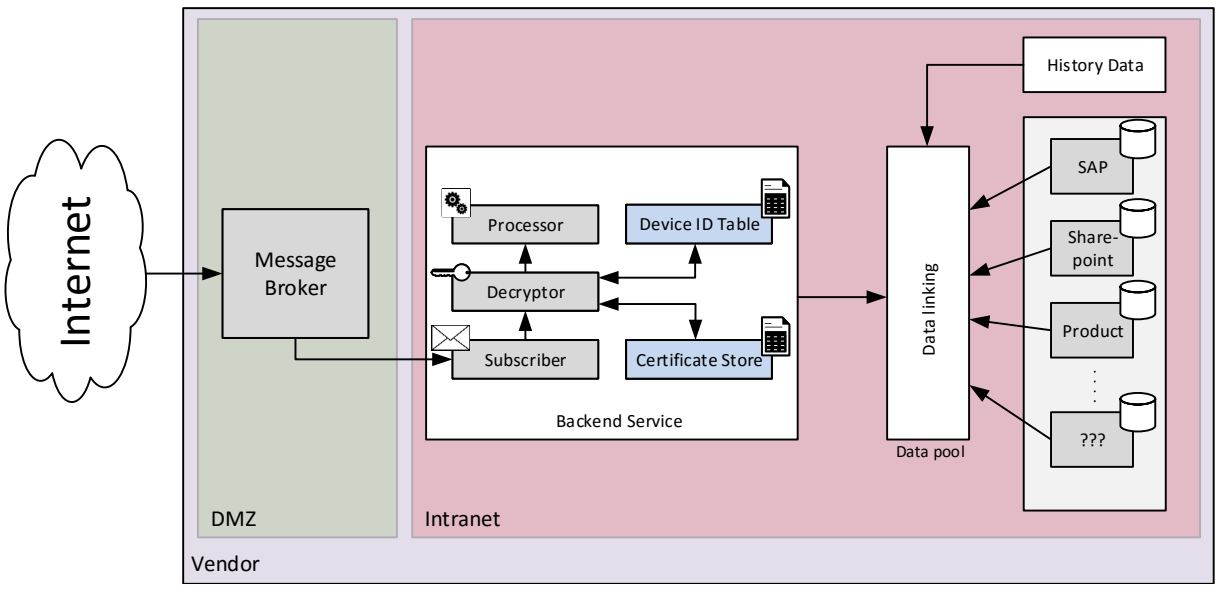


Figure 5.2: Here the flow of data being sent from a customer device through the backend is illustrated. First, the broker in the DMZ forwards the data to an MQTT subscriber in the backend. In the next step, the end-to-end encryption of the payload needs to be processed. For this purpose, the backend decryptor checks the authenticity of the originator and decrypt the data. The payload, now in plain again, can then be processed and forwarded into data stores. [Drawn by the author of this thesis.]

5.3 Mosquitto Setup

Listing 5.1 shows the configuration of the prototypical Mosquitto broker.

For Mosquitto versions earlier than 1.4, it was needed to patch the source code to support the TLS cipher suite. Mosquitto was not yet able to deal with ECDSA signatures. However, starting with version 1.4, the feature got integrated into Mosquitto and no source code adaption was necessary anymore. Instead it is possible to configure all security relevant features with Mosquitto's integrated standard configuration options.

5.4 MQTT subscriber

Many client implementations exist for MQTT. In the backend service, one of the probably widest spread ones available is used: the *Paho Java Client*[2] by *Eclipse*. It provides an asynchronous API for MQTT events and the – for this use case important – ability to use custom

[2] `eclipse.org/paho/clients/java/`, Accessed: May 5, 2017

```
# use standard secure MQTT port
port 8883

# define CA and own certificate + key
cafile MQTT_CA.crt
certfile MQTT_Server.crt
keyfile MQTT_Server_Key.pem

# only allow TLS version 1.2 with our specific cipher
tls_version tlsv1.2
ciphers ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384

# clients MUST present certficates from the same CA
require_certificate true

# authorization mechanism: use certificate identity as username
# and our configured ACL
use_identity_as_username true
acl_file topic.access

# logging settings
log_dest stdout
log_dest file broker.log
log_type all
connection_messages true
```

Listing 5.1: Mosquitto configuration for the prototype.

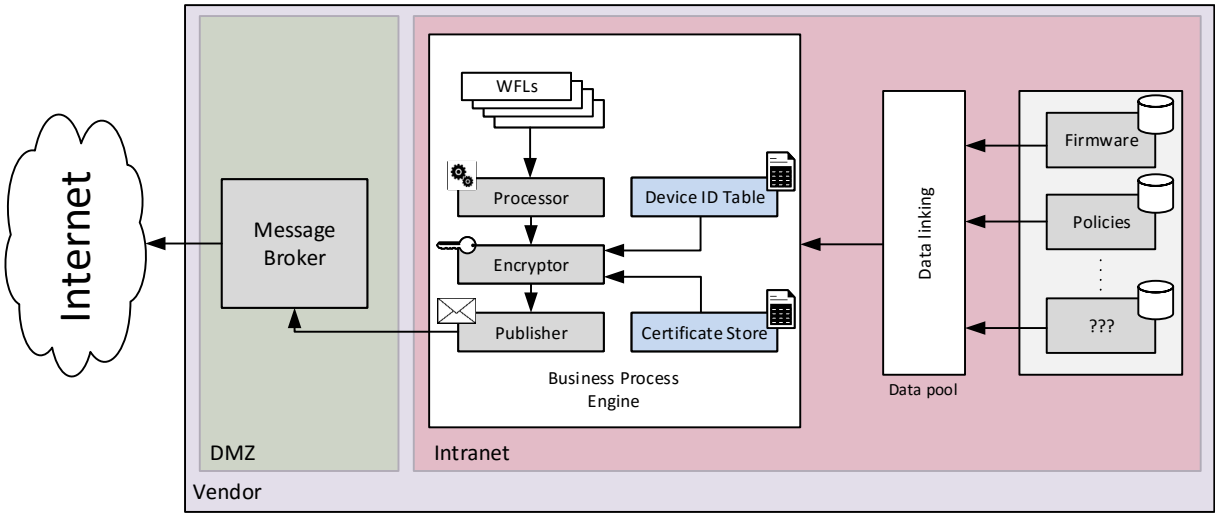


Figure 5.3: Here is illustrated, how data flows from the vendor to the customer. For example, backend workflows (WFLs) can trigger the publishing of new policies or roll out new firmware versions to devices. For end-to-end-encryption, the backend encryptor wraps the payload specifically for desired devices. The MQTT publisher sends the encrypted payload to the broker which then forwards it to subscribed clients. [Drawn by the author of this thesis.]

socket implementations for the connection to the MQTT broker.

The subscriber in the prototype is in possession of an identity certificate with the subject "vendor-backend-service" and the corresponding private key. According to the ACL configuration of the MQTT broker in Listing 4.2, the subscriber is granted access to topics `+/+/+/snapshots`. Thus, it is able to collect snapshots from all devices.

5.4.1 TLS connection initialization

Regarding TLS, the Java framework provides sufficient functionality for general use cases, like for example establishing a connection to remote server. However, the TLS related procedures are quite specific and are not supported out of the box with Java's built in secure socket generator (`SSLSocketFactory`). One feature they are lacking, is the support of client identity certificates. It was needed to develop an own implementation of a socket generator, to fulfill the strong security needs.

Listing 5.2 shows the implementation of the TLS socket factory, which can be used to generate TLS sockets to connect to the MQTT broker. The generated sockets use an identity given in Public Key Cryptography Standard #12 (PKCS #12) format to connect to remote entities. Additionally, the socket connection checks the identity of the remote endpoint by trying to build a chain of trust to a given CA. For this purpose, it uses a custom trust manager.

```java
/**
 * Builds a TLS socket factory using a given identity and trusting
 * remote entities for which a chain of trust can be built to a given CA.
 *
 * @param caFile CA file
 * @param privateCertKeyFile certificate/key file in PKCS12 format
 * @param privateCertKeyPassword password to open the privateCertKeyFile
 * @return TLS socket factory
 */
public static SSLSocketFactory getFactory(
    File caFile,
    File privateCertKeyFile,
    String privateCertKeyPassword
) throws Exception {
  // keyStoreType is either "JKS" or "PKCS12"
  KeyStore keyStore = KeyStore.getInstance("PKCS12");
  keyStore.load(new FileInputStream(privateCertKeyFile),
    privateCertKeyPassword.toCharArray());

  // build keymanager
  KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
  kmf.init(keyStore, privateCertKeyPassword.toCharArray());

  KeyManager[] x509KeyManagers = Arrays.stream(kmf.getKeyManagers())
      .filter(x -> x instanceof X509KeyManager)
      .toArray(KeyManager[]::new);

  // ensure a X509KeyManager was found
  if (x509KeyManagers == null || x509KeyManagers.length == 0) {
    throw new NullPointerException("No X509KeyManager has been found");
  }

  // build trust manager, which trusts all certificates
  // which have a chain to the trusted root
  TrustManager[] x509TrustManagers =
    new TrustManager[]{new TrustedRootTrustManager(caFile)};

  // create the SSLContext with TLS 1.2
  SSLContext sslContext = SSLContext.getInstance("TLSv1.2");

  // initialize the SSLContext with the X509KeyManager and TrustManager
  sslContext.init(
      x509KeyManagers,
      x509TrustManagers,
      new SecureRandom());

  return sslContext.getSocketFactory();
}
```

Listing 5.2: Implementation of the Java TLS socket factory.

Listing 5.3 shows the implementation of the *trust manager* for the backend MQTT subscriber. The custom socket factory uses it to verify certificate chains of remote entities. It utilizes a predefined root CA for validating certificate chains. During the TLS handshake the broker provides its certificate chain. The trust manager first checks, whether the broker also attached the root CA to the chain. According to the TLS specification [RFC5246], this is not mandatory. However, some TLS implementations attach it and it is not a breach of contract to do so. After it was assured, that the chain contains the root CA, it is traversed through the chain and validate each certificate's signature with the public key of the succeeding key. In the same step, it is also verified, that the current time is inside the validity period of the certificate. In the next step, it is checked whether the root CA equals the trusted root (if it was added it in the first step it will of course but if the broker already sent it, it is needed to verify it). In the last step it is verified, that the maximal path length as specified by the root CA is not exceeded. The maximal path length specifies the number of intermediate certificates allowed between the root CA and the endpoint certificate. Thus, it is needed to check against the chain's length minus two (for the endpoint certificate and root CA).

5.5 CMS decryptor

For the needed cryptographic operations *Bouncy Castle*[3], an open-source Java library providing cryptographic APIs, is used.

The *BouncyCastle* cryptography API is structured into a set of packages. The following packages are used:

- **Provider for the Java Cryptography Extension and the Java Cryptography Architecture**
  Provides general security functions compliant with existing Java security APIs.
- **A library for reading and writing encoded ASN.1 objects**
  This package provides functionality for encoding and decoding all kinds of ASN.1 objects and data structures.
- **Generators for Version 1 and Version 3 X.509 certificates, Version 2 CRLs, and PKCS12 files**
  By using this package, one can handle digital certificates and CRLs.
- **Generators/Processors for S/MIME and CMS (PKCS7/RFC 3852)**
  This package allows to generate and read messages in a CMS format.

[3] `www.bouncycastle.org`, Accessed: May 6, 2017

```
1  public void checkServerTrusted(X509Certificate[] chain, String authType)
2        throws CertificateException {
3      try {
4        List<X509Certificate> certChain = Arrays.asList(chain);
5
6        // check if the chain contains the trusted root or not
7        // if not, add it
8        if (!certChain.get(certChain.size() - 1).equals(this.caCert)) {
9          certChain.add(this.caCert);
10       }
11
12       // verify all certificates in the chain are valid and signed by
13       // the succeeding certificate
14       for (int i = 0; i < certChain.size() - 1; i++) {
15         X509Certificate cert = certChain.get(i);
16         cert.verify(certChain.get(i + 1).getPublicKey());
17         cert.checkValidity();
18       }
19
20       // ensure that the server has the same CA as we do
21       // CA is always the last certificate
22       if (!certChain.get(certChain.size() - 1).equals(this.caCert)) {
23         throw new CertificateException("Invalid CA certificate");
24       }
25
26       // assure that the max path length is not exceeded
27       int maxPathLength = this.caCert.getBasicConstraints();
28       if (maxPathLength > -1 &&
29           maxPathLength != Integer.MAX_VALUE &&
30           certChain.size() - 2 > maxPathLength)
31       {
32         throw new CertificateException("Path length is "
33           + (certChain.size() - 2)
34           + ", exceeds allowed maximum of " + maxPathLength);
35       }
36
37     } catch (Exception e) {
38       throw new CertificateException(e.getMessage());
39     }
40   }
```

Listing 5.3: Implementation of the TLS trust manager for verification of the broker certificate.

The CMS relevant operations for end-to-end operation have been implemented in such a way, that they can be used on both the customer device side as well as the vendor side. Thus, first a common set of functions was defined which is needed for all entities, which want to conduct CMS operations.

Interface Listing 5.4 defines a Java interface, which classes need to implement in order to be able to handle CMS messages. Implementations must allow to specify the caller's identity via a PKCS #12 encoded identity file. For CMS operations, a function for encryption as well as decryption of data needs to be implemented.

The encrypting function takes a sequence of bytes to encrypt alongside a list of recipients' certificates. The certificates are needed to wrap the symmetric encryption key for each recipient separately. The decrypting function accepts a sequence of bytes representing a CMS message. It tries to find a wrapped key for the caller's identity and decrypt the payload.

General decryption process In Listing 5.5 the decryption of a received CMS message on an abstract level is shown. The function takes as arguments the digital certificate and the private key of the recipient as well as the encrypted CMS message. Upon successful completion, it returns the decrypted payload.

The first step in the decryptor is to generate a *BouncyCastle* CMS object from the plain message bytes for easier handling. Then the public key of the originator is extracted.

In the next step, it is tried to find the key, which was wrapped for the specific recipient. In case the wrapped key is not found, the message can not be decrypted. The function terminates and throw an exception. If the wrapped key for the recipient was found, it is tried to unwrap the key using ECMQV operations.

Then the initialization vector for the MAC validation needs to be extracted. Using the initialization vector and the symmetric key, it is possible to recalculate the MAC for the encrypted content info and compare it to the MAC, which was calculated by the sender and sent along in the message.

Lastly, using the extracted key from before, the content can be decrypted and returned to the function caller.

Calculation of the shared secret Listing 5.6 shows the specific unwrapping of the wrapped key for AES operations according to the description in Section 2.3.2.1 in more detail.

In order to unwrap the key using ECMQV, the needs as input the originator's public key, the recipients private key as well as the wrapped key and the user keying material, which have been extracted from the CMS data in a step not shown. From the user keying material, the ephemeral public key, which was previously generated by the originator, can be reproduced.

```java
/**
 * Provides an interface for the decryption and encryption of CMS messages.
 * @author martin.maritsch
 */
public interface ICmsStrategy {
    /**
     * Instructs the underlying implementation to use an identity for
     * CMS operations
     * @param identity the identity to be used for CMS operations
     * @param pkcs12FilePassword the password to access the pkcs12File
     * @throws InputException in case the identity file could not be parsed
     */
    void setClientIdentity(Pkcs12File identity, String pkcs12FilePassword)
      throws InputException;

    /**
     * Decrypt an CMS message
     * @param encryptedMessage the byte representation of the CMS message
     * @return the decrypted payload of the CMS message
     * @throws InputException in case an invalid CMS message is provided
     * @throws BugException in case there is an server related issue
     */
    byte[] decrypt(byte[] encryptedMessage)
      throws InputException, BugException;

    /**
     * CMS encrypts gdata for a list of recipients
     * @param recipientCerts X.509 certificate identities of recipients
     * @param data data to be encrypted
     * @return the CmsMessage
     * @throws InputException in case there are any invalid parameters
     * @throws BugException in case there is a server related issue
     */
    byte[] encrypt(List<X509Certificate> recipientCerts, byte[] data)
      throws InputException, BugException;
}
```

Listing 5.4: Implementation of the Java Interface for a CMS strategy.

```
1  /**
2   * Decrypts a CMS message (Authenticated-Enveloped-Data) for a recipient
3   *
4   * @param recipientPrivateKey recipient's private key for the certificate
5   * @param recipientCert       recipient's certificate
6   * @param encCmsBytes         byte array representing CMS message
7   * @return decrypted content of CMS message
8   */
9  public static byte[] decryptData(
10     @NonNull PrivateKey recipientPrivateKey,
11     @NonNull X509Certificate recipientCert,
12     byte @NonNull [] encCmsBytes) {
13   // construct BouncyCastle CMS object from byte array
14   AuthEnvelopedData data = AuthEnvelopedData.getInstance(
15       ASN1Sequence.getInstance(encCmsBytes));
16
17   try {
18     // load originator key, wrapped key and unwrap the key
19     PublicKey originatorPublicKey = getOriginatorPublicKey(
20         data.getOriginatorInfo());
21     byte[] wrappedKey = getWrappedKey(data.getRecipientInfos(),
22       recipientCert);
23     SecretKey symmetricKey = unwrapKey(wrappedKey, originatorPublicKey,
24         recipientPrivateKey,
25         getUserKeyingMaterial(data.getRecipientInfos(), recipientCert));
26
27     EncryptedContentInfo contentInfo = data.getAuthEncryptedContentInfo();
28
29     // get the initialization vector and validate the MAC
30     byte[] iv = ((ASN1OctetString) contentInfo
31         .getContentEncryptionAlgorithm().getParameters()).getOctets();
32     validateMac(data.getMac(), contentInfo, symmetricKey, iv);
33
34     byte[] encBytes = contentInfo.getEncryptedContent().getOctets();
35
36     // do the decryption
37     byte[] decBytes = decryptMessage(encBytes, symmetricKey, iv);
38     return decBytes;
39   } catch ([...]) {
40     [...]
41   }
42 }
```

Listing 5.5: Implementation of the CMS data decryptor.

Afterwards, the key wrapping key can be computed using the ECMQV key agreement service provided by *BouncyCastle*. Finally, the wrapped key can be unwrapped.

5.5.1 AES-GCM engine

Listing 5.7 shows the initialization and the encryption of the implemented AES-GCM engine. The decryption works analogous to the encryption with the only difference being the initialization of the AES engine.

An implementation must allow to set an identity of the calling client. This serves two purposes: for encryption, this identity is used as the originator identity. For decryption, the recipient information set of the CMS message to be decrypted is searched for the given identity. If the set does not contain a recipient information for the identity, the CMS message can not be decrypted since the symmetric key needed for operations in the AES process can not be reconstructed.

5.6 Relational database schema

A relational data schema was designed, which allows to aggregate device snapshot information in a relational database. The database serves as the foundation for the exemplary data workflows described in Section 5.7.

During a few iterations, the schema was tested with 100,000 mocked snapshots comprising a total of 34,000,000 snapshot value rows. Based on the result of commonly needed queries, the schema was revised to execute these queries within seconds instead of minutes.

Figure 5.4 shows the final data schema.

However, the schema might not be perfectly suited for a subsequent use in *big data* applications where the trend is to use non-relational databases.

5.7 Exemplary data workflows

In order to demonstrate the practical use of the prototypical design and implementation, two exemplary workflows are made, which are built on top of the data aggregation framework.

The workflows are fully based on the implemented prototype. They are accessing data in a *MySQL* database set up with the data schema proposed in Section 5.6.

From resulting applications are prototypes and not ready for production use. However, it is believed that they show the great potential of the SMS framework and the variety of applications and services, which can be built on top of the framework.

```
/**
  * Unwraps the symmetric key from the wrapped key.
  *
  * @param wrappedKey          wrapped key
  * @param originatorPubKey public key of originator
  * @param recipientPrivKey    private key of recipient
  * @param ukm                 user keying material for the recipient
  * @return symmetric key
  */
private static SecretKey unwrapKey(byte[] wrappedKey,
      PublicKey originatorPubKey,
      PrivateKey recipientPrivKey,
      ASN1OctetString ukm)
    throws Exception {
  String keyAgreeAlgoId = CMSAlgorithm.ECMQV_SHA1KDF.getId();
  String keyWrapAlgoId = CMSAlgorithm.AES256_WRAP.getId();

  // extract the ephemeral public key
  AlgorithmIdentifier privateKeyAlgo = PrivateKeyInfo
    .getInstance(recipientPrivKey.getEncoded()).getPrivateKeyAlgorithm();
  MQVuserKeyingMaterial mqvUkm = MQVuserKeyingMaterial
    .getInstance(ASN1Primitive.fromByteArray(ukm.getOctets()));
  byte[] publicKeyBytes = mqvUkm.getEphemeralPublicKey()
    .getPublicKey().getBytes();

  X509EncodedKeySpec pubSpec = new X509EncodedKeySpec(
      new SubjectPublicKeyInfo(privateKeyAlgo, publicKeyBytes)
    .getEncoded());
  PublicKey ephemeralPublicKey = KeyFactory.getInstance(keyAgreeAlgoId)
    .generatePublic(pubSpec);

  // calculate the key used for key wrapping usieng ecmqv
  MQVParameterSpec mqvParameters =
    new MQVParameterSpec(recipientPrivKey, ephemeralPublicKey);
  KeyAgreement agreement = KeyAgreement.getInstance(keyAgreeAlgoId);
  agreement.init(recipientPrivKey, mqvParameters);
  agreement.doPhase(originatorPubKey, true);
  SecretKey wrappingKey = agreement.generateSecret(keyWrapAlgoId);

  // unwrap the key to get the plain symmetric key for AES
  Cipher keyCipher = Cipher.getInstance(keyWrapAlgoId);
  keyCipher.init(Cipher.UNWRAP_MODE, wrappingKey);

  return (SecretKey)
    keyCipher.unwrap(wrappedKey, keyAgreeAlgoId, Cipher.SECRET_KEY);
}
```

Listing 5.6: Unwrapping of the symmetric key for AES from the wrapped key.

```
[...]
  private ParametersWithIV gcmParams;
  private byte[] mac;

  /**
   * Constructs an encryption/decryption tool on base of a GCM block cipher
   * @param iv  the initialization vector to be used for GCM
   * @param key the symmetric key to be used for enctyption/decryption
   */
  public AesGcmBcBlockCipher(byte[] iv, SecretKey key) {
    // inititalize the GCM block cipher parameters
    // with the symmetric key and an initialization vector
    this.gcmParams = new ParametersWithIV(
      new KeyParameter(key.getEncoded()), iv);
    this.mac = null;
  }

  /**
   * Encrypt data using a GCM block cipher
   * @param data the bytes to encrypt
   * @return the encrypted bytes
   */
  public byte[] encrypt(byte[] data) {
    GCMBlockCipher gcmEnc = new GCMBlockCipher(new AESEngine());

    // initialize the block cipher for encryption
    gcmEnc.init(true, this.gcmParams);

    // prepare a new byte array for the output data.
    // GCMBlockCipher has a handy function to tell us the output size of
    // a data block before processing it
    byte[] encrypted = new byte[gcmEncrypt.getOutputSize(data.length)];

    // encrypt the input data
    int length = gcmEnc.processBytes(data, 0, data.length, encrypted, 0);

    try {
      // do the final calculations for the message authentication code
      gcmEncrypt.doFinal(encrypted, length);
    } catch (IllegalStateException | InvalidCipherTextException e) {
      return null;
    }
    this.mac = gcmEncrypt.getMac();
    return encrypted;
  }
[...]
```

Listing 5.7: Implementation of the AES-GCM initialization and encryption.

5.7.1 Dashboard

In the SMS dashboard, a visualization of collected snapshots is provided. It is primarily intended for customers but may also be used by other stakeholders such as the vendor's maintenance staff. The user is able to gather an overview of statuses for the industrial equipment they are responsible for. A per-device view provides detailed insights into equipment-specific information, such as the owner or identifiers of the device. Besides providing status relevant information at a glance, gauges show the progress of various parts within their expected lifecycle. Furthermore, period based visualizations – like graphs – can be checked in order to discover abnormal device conditions or investigate on the general device utilization. Figure 5.5 shows an exemplary device status view of the dashboard prototype.

The SMS dashboard is useful for both customer and vendor. The customer can keep track of their inventory, schedule operations and investigate utilizations. Furthermore, they can monitor all their devices and detect possible failures. The vendor is able to check device statuses for its devices and can warn its customers in case abnormal conditions are observed.

To show the potential of the customer dashboard, a mocked feature was integrated, which showed customers specific replacement parts and equipment consumables such as lubricants. Based on device values, specific products were suggested with the possibility to order it with one click.

5.7.2 Business Process Model (BPM)

To demonstrate the ability of the SMS framework to warn customers or the vendor in case of abnormal device conditions, a BPM (Figure 5.6) was drafted, which triggers certain workflows for incoming snapshots of industrial equipment.

The model differentiates between three different equipment states, which are derived from equipments' snapshot values. In the standard case, when a snapshot indicates that the equipment is within regular and expected health condition, a mail is sent to a predefined list of recipients. Should the snapshot values indicate an abnormal or erroneous equipment state, a text message is immediately being sent to a service technician in charge for the equipment. Furthermore, the system generates an investigation task for the service technician. If a snapshot indicates an overdue maintenance appointment for an equipment, a vendor technician is urged to schedule a maintenance appointment with the customer.

In order to execute and process the BPM it was decided to make use of the open-source BPM platform *Activiti*[4]. Compared to other BPM processing engines, *Activiti* was deemed to be a rather simple platform which was perfectly suited for the straightforward show-case. Whenever the maintenance service receives a new equipment snapshot, the backend data interpreter

[4] `www.activiti.org`, Accessed: May 11, 2017

triggers the evaluation of the BPM based on the snapshot's values

A BPM offers an easy-to-understand and pleasant view of underlying workflows in the system. This was an advantage in so far as it allowed to show the practical use of the SMS framework on a high level and in an understandable way to potential customers and interested persons.

However, the design options of a BPM are very limited and the implementation of custom service tasks for the BPM, which was executed in the BPM platform *Activiti*, truly is a challenging task.

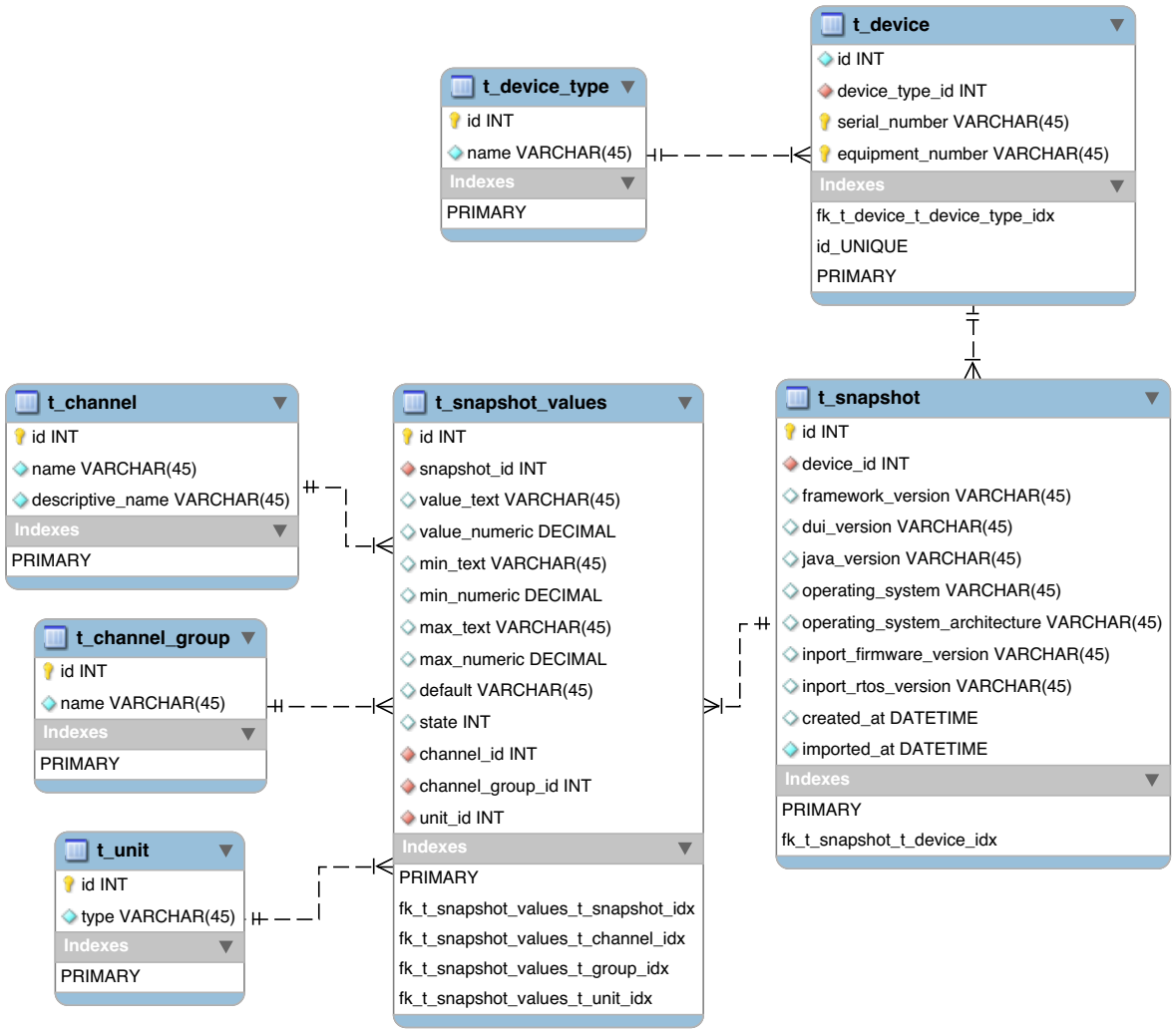


Figure 5.4: The elaborated relational data schema for the storage of snapshot data. Devices are clustered based on their type (`t_device_type`). Entities in `t_device` represent single devices. A snapshot (`t_snapshot`) consists of many snapshot values (`t_snapshot_values`). For performance reasons of the queries, often queried parameters have been moved and fields with little different values into extra, indexed tables (`t_channel_group`, `t_channel` and `t_unit`). [Drawn by the author of this thesis.]

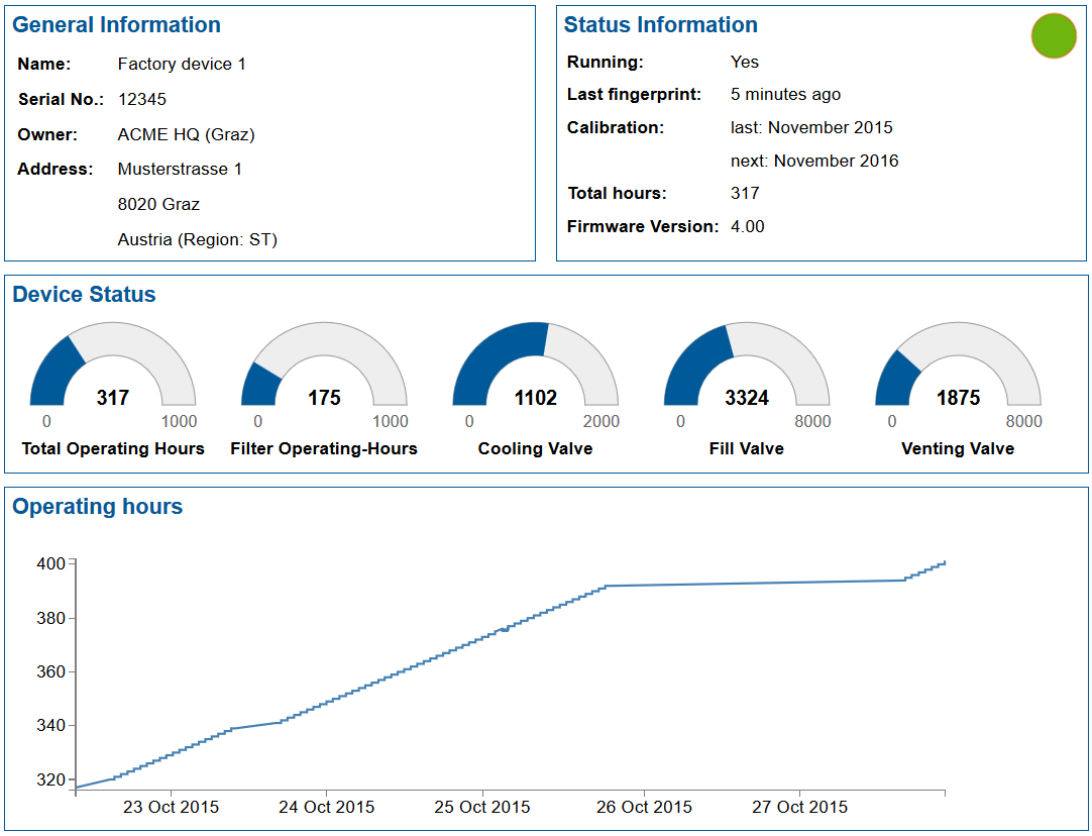


Figure 5.5: An exemplary dashboard for browser-based visualizing of equipment data. The dashboard contains several metadata and basic status information about the equipment. Besides that, there are multiple gauges showing the progress of component's values towards the end of their life cycle and recommended replacement. The evolution of the equipment's operating hours is shown in an line graph, which allows to discover system downtimes and utilize the equipment's overall utilization. [Screenshot by the author of this thesis.]

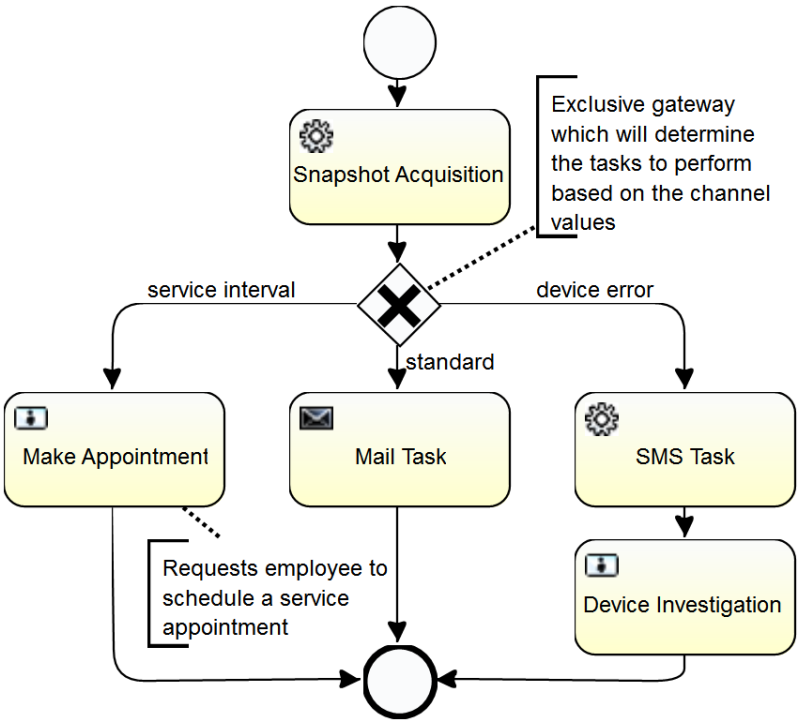


Figure 5.6: Exemplary BPM. The model uses an exclusive gateway which triggers one of three procedures based on the input values. [Screenshot by the author of this thesis.]

Chapter 6

Discussion

> *" If you went to bed last night as an industrial company, you're going to wake up today as a software and analytics company "*
>
> [Jeff Immelt, chairman and CEO of General Electric]

Throughout the whole design and implementation of the SMS framework one of the main goals was, to use established standards. The communication foundation of the system – MQTT – is a recognized, industrial standard [MQTT Version 3.1.1; ISO20922]. The security functionalities, like TLS or ECC, are widely in use and have proven effective even in global-scale applications.

This has several advantages: First, the standards are extensively used, so many of its flaws and drawbacks have already been found and eradicated. Furthermore, the argument of using for example an ISO standard in the case of MQTT, is more likely to convince customers and eliminate their security concerns. Moreover, there are many widely-used implementations available. This is especially convenient for security features, since established implementations can be used (for example the *Bouncy Castle*[1] cryptography libraries for critical security functions).

Parts of this chapter are reproduced from the author's publications [Maritsch, Lesjak, and Aldrian, 2016], [Maritsch, Kittl, and Ebner, 2015] as well as [Lesjak, Bock, et al., 2016].

[1] `www.bouncycastle.org`, Accessed: May 16, 2017

6.1 The SMS framework as a service-oriented architecture (SOA)

The implemented SMS framework adheres to the principles of a SOA as described in Section 2.1.

One can think of the devices as service providers and consumers at the same time. Also, the backend consists of consumers and providers. An exemplary service consumer in the vendor backend would be the snapshot processing service; an exemplary service provider would be the firmware update service.

In the SMS framework the MQTT broker serves as the service registry. The MQTT topics are definitions of services. For example, `macc1/acc2/device123/snapshots` is a service, which provides snapshots of a specific device. Subscribers of this topic are service consumers. The TACS in the broker takes over the function of the authorization service by regulating access to topics. The uncommon thing in the SMS framework as a SOA is, that the interaction between service consumers and producers is conducted directly via the service registry.

Figure 6.1 illustrates these components in the snapshot acquisition use case.

While the representation of the SMS framework as a SOA has no practical effects, it promotes the understanding of the architecture. It helps us, to understand the principles of the SMS framework and allows to figure out how to provide and make use of functionality available in the SMS framework.

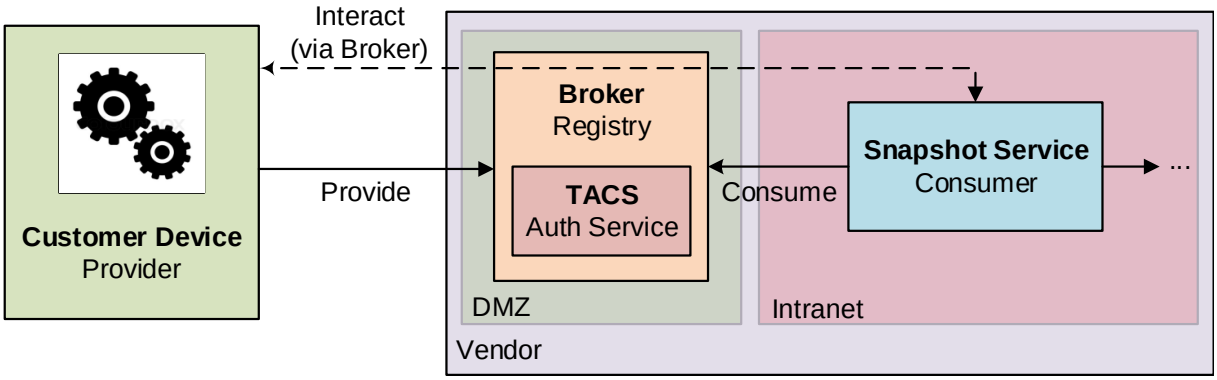


Figure 6.1: This figure portrays the snapshot acquisition use case as a SOA. The customer device represents a service *provider* and the snapshot service the *consumer*. The broker takes over the role of the *service registry* (i.e. the topics) and the *authorization service* with its TACS. [Drawn by the author of this thesis.]

6.2 Data transmission with MQTT

First the use of MQTT in regard to the requirements to an IoT transmission protocol is evaluated, which are mentioned in Section 2.2:

- **Scalability**
  An MQTT architecture is highly scalable to even a large number of clients. Already a single broker instance can handle hundreds of thousands of connected clients[2]. If support for a higher number of clients is needed, multiple MQTT brokers can be used in a bridge or cluster. Table A.1 shows which MQTT broker implementations support bridging as well as clustering.

- **Efficiency**
  MQTT only adds a fixed header of two bytes to a message payload [MQTT Version 3.1.1]. This is significantly less compared to other protocols like HTTP or SMTP (see Table 2.1). Additionally, a variable length header depending on the topic is needed.

- **Portability**
  The only requirement for supporting MQTT is an implementation of the TCP/IP stack which is commonly even available on very basic embedded devices. MQTT implementations are available in many different programming languages[3].

- **Simplicity**
  MQTT is specifically designed to be extremely simple and lightweight, for use in constrained devices. Implementations with a very low code and performance footprint are available.

- **Failure tolerance**
  Due to its design with a central broker and the support of persistent sessions, messages for temporary disconnected devices can be cached and delivered once the devices are available again. Thus, a client will get all its pending messages even after it has been disconnected to the broker for some time.

Also, the following reasons led to choosing MQTT:

- **Publish-subscribe pattern**
  The pattern of MQTT allows bidirectional communication. This allows for example the vendor to push information to devices without them explicitly requesting for them.

[2] `stackoverflow.com/q/10087396`, Accessed: May 16, 2017
[3] `github.com/mqtt/mqtt.github.io/wiki/libraries`, Accessed: May 16, 2017

- **Connection establishing always from the *inside***
  Connections are always established from the inside of networks to the broker. Thus, a customer only needs to open one outgoing port in their firewall to connect to the SMS framework. Connection attempts are never made from the outside of customer networks to devices.

- **Security Features**
  Using MQTT, all of the security requirements can be implemented using standard frameworks and libraries.

In the SMS framework it is the system's duty to reliably deliver certain types of important messages to devices. These message types would for example include remote commands or security policy updates. It must be assured, that these messages get delivered to devices. The reliability of MQTT is ensured as it is working on top of TCP and benefits from TCP's reliability features. Additionally, as QoS features are used, the delivery of messages in the SMS framework can be guaranteed whenever needed.

Figure 6.2 shows two network packets captured with *Wireshark*[4]. Subfigure 6.2a shows an MQTT publish encapsulated in a TCP packet. Subfigure 6.2b shows an HTTP post request encapsulated in a TCP packet. Both requests use an equivalent resource (MQTT topic: `foo/bar`, HTTP resource: `/foo/bar`) and have the same payload ("Hello World"). As for the transferred data, with 167 bytes the HTTP packet is almost twice as large compared to the MQTT packet with 88 bytes. In the screenshots it can be clearly seen, that all of this overhead is imposed by the application layer protocol since all underlying protocols are equal. The content actually needed to be transferred has a size of 18 bytes ("foo/bar" + "Hello World"). The MQTT packet itself is 22 bytes; only an overhead of 4 bytes is added. For HTTP the packet size is 101 bytes which means an added overhead of 83 bytes. Thus, the overhead of the HTTP message is more than 20 times larger as for MQTT.

Yokotani and Sasaki [2016, Figure 10] elaborate on the significant overhead of HTTP in contrast to MQTT in more detail.

6.2.1 Topic Structuring

The chosen structure it is not the "one to rule them all" structure since it is for example not possible to subscribe to all messages of a device group (for example FMs). However, it is a reasonable trade-off between an extensive topic structure and the amount of information being exposed via the topic. In the structure, it can for example not be derived, how many instances of a device group some customer has. The topic hierarchy allows to easily cluster devices into groups and cluster data as desired.

[4] `www.wireshark.org`, Accessed: May 25, 2017

Furthermore, the structure that was chosen is easily and dynamically expandable to newly added customers. They just need to provide a valid certificate which in turn will be used by the broker to dynamically create the new topics.

6.2.2 Broker Architecture and Cascading

the physical segregation between customer data of the architecture described in Section 3.3.3 (Per-customer-broker on vendor's premises plus central master broker) is considered to be its main advantage. However, there is no need for segregation of data if it is encrypted when being processed in the broker. Thus, an additional layer of security by applying end-to-end encryption to data is proposed. Defining a set of recipients which are authorized to receive and process application data in this security measure is suggested. The proposed hybrid encryption system is a cryptographic measure to realize this.

In practice, a hybrid solution is proposed consisting of the architectures described in Section 3.3.1 (Single broker on vendor's premises) and Section 3.3.2 (Per-customer-broker on customer's premises plus central master broker) since they provide a reasonable trade off between scalability, fault tolerance and installation/administration efforts. There might be customers with only few devices which may not want to set up a dedicated message broker for the connection of industrial equipment to the vendor and struggle with its implementation and administration. However a broker may optionally be set up in order for the customer to be able to monitor outgoing data or when a central gateway to the vendor is desired. On the other hand, customers with considerable huge amounts of devices, which may also be spread over different premises, will certainly consider setting up one or more internal broker(s) serving as gateway(s) to the vendor for industrial equipment.

In general, it is suggested to place the broker(s) on vendor side in a DMZ and not directly in the backend for multiple reasons. First, since the broker is placed at a public location in which all data comes together, it is considered to be the most vulnerable link in the whole dataflow from customer devices to the vendor backend. Thus it might be the most interesting point for adversaries to attack. Additionally, it is usually of vendors interest, to keep remote Internet connections into the backend at a minimum. MQTT does not require connections to be actively established into the backend from the outside. Instead, backend components need to initiate connections and subscribe to the broker in the DMZ.

6.2.3 Topic Access Control System (TACS)

It was shown how to design a straightforward yet powerful topic hierarchy for the SMS framework with MQTT's topic system as a foundation. For limiting topic accesses to only legitimate entities, a simple TACS using *Mosquitto*'s ACL functionality was implemented. Although the implementation seems pretty basic on first sight, it proved to be reliable and robust for all needs

in the SMS framework. According to evaluations, it should also be sufficient for use in a productive environment with tens of thousands of devices [Maritsch, Lesjak, and Aldrian, 2016].

6.3 Security

As described in Chapter 4, up-to-date cryptographic systems and functions for all of the security-related operations are used. Furthermore, wherever possible implementing custom security functions are avoided. Instead, existing implementations which are in broad use were used. The advantage clearly is, that such widely spread implementations are much more thoroughly tested and less likely to contain bugs than custom implementations would.

Vendors need to be aware, that customers are providing their highly sensitive data on a goodwill basis. Of course customers are provided valuable SMSs, however many customers are – understandably – pedantic about their testbed data being transferred outside their premises. Thus, vendors should offer possibilities for customers to precisely monitor the data being sent and received by their devices.

6.3.1 Authentication

Authentication measures in the SMS framework are taken at multiple levels. First of all, on the data transport layer, authentication is performed by means of standard TLS authentication. For the purpose of proving identities, digital certificates are used which allows entities throughout the SMS framework to validate identities of other remote entities.

The nature of the authentication process is largely similar to the way of how web servers authenticate themselves for remote clients in the world wide web. The only difference is, that in the SMS framework not only servers (i.e. the vendor backend and its MQTT brokers) need to authenticate but also the clients (i.e. client devices, vendor backend services) whereas web servers normally allow arbitrary clients to establish connections.

The SMS framework not only authenticates network entities upon establishing connections but also allows recipients to authenticate messages being transferred via the SMS framework. This means, that a recipient of a message can undoubtedly identify the originator of a message. The security mechanisms also yield non-reputability, meaning that the originator cannot deny having authored a message. Thus, the sender can always rely on the message being authentic.

6.3.2 Authorization

Some well-established authorization frameworks exist. One might legitimately wonder, why making use of them in the SMS framework is not wanted. One popular among them is *OAuth*

[RFC6749]. Niruntasukrat et al. [2016] demonstrated the use of *OAuth* in an MQTT architecture. Considering the popularity and the features of *OAuth*, this seems appropriate. However, the *OAuth* features would require the customer, to allow an additional outbound connection of devices into the Internet. This is needed to enable the exchange of tokens with an *OAuth* authorization server. Unfortunately, every granting of network access abilities increases the danger of security gaps and possibly exposes industrial equipment to more attacks.

The system allows customers to keep the allowed outbound connections in their firewall configuration on an absolute minimum; only a single port for MQTT is required. With the certificate based authorization system, the need for additional security frameworks is eliminated. All security requirements regarding authorization can be fulfilled using the proposed certificate-based authorization system.

From an implementation point of view, authorization is mainly conducted in the MQTT broker or more specifically in its TACS, where topic reads and writes are validated against a client's identity.

The chosen certificate hierarchy, which corresponds to the MQTT topic structure, is very useful for granting more powerful rights to device owners. Thus, they can aggregate data and transparently monitor incoming as well as outgoing data for all of the devices they own. Experience has shown, that for many potential customers, the ability of monitoring this data is an absolute must.

6.3.3 End-to-end Encryption

It was decided to introduce an additional layer of security besides TLS on application layer. The used end-to-end encryption system ensures that data is not only securely encrypted while in transport between network entities (i.e. between a client and the MQTT broker). Additionally, data is also encrypted and never available in plain while being processed at any point.

Furthermore, end-to-end encryption allows the originator of messages to specifically address recipients of the data and encrypt it in such a way, that only they can decrypt it.

The security benefits are evident: Most importantly, adversaries which manage to gain access to the message broker, are not able to read any of the application data being transmitted in the SMS framework. Secondly, message payload which were – under whatever circumstances – received by undesired network entities cannot be processed by them.

Message overhead The end-to-end encryption requires some overhead added to the payload which can be approximated with $700 + n \times 500$ bytes where n is the number of recipients according to Lesjak, Bock, et al. [2016]. Although the message overhead could be reduced using other encodings for cryptographic information, CMS should be preferred whenever possible since it is designed as a platform-independent and inter-operable standard.

6.3.4 Key Management

For the key management, the widely spread X.509 standard is used for digital certificates. This allows to make use of a large number of existing security implementations; both on hardware as well as on software side.

Furthermore, the X.509 standard provides some features which can be used to defuse severe security threats. Using CRLs, one can easily revoke invalid certificates – for example if an industrial equipment gets stolen or a security key gets compromised. The up-to-date CRLs should periodically be pushed to industrial equipment.

Many security threats can be eliminated by using a custom Root CA which are pinned in all entities in the SMS framework. Thus, the entities are only allowed to trust certificates, for which they can build and traverse a certificate chain up to the SMS Root CA. They do not have a standard set of Root CAs like for example the ones operating systems ship with – for example *macOS*[5]. The described approach is used for both the TLS PKI as well as the PKI used in end-to-end encryption.

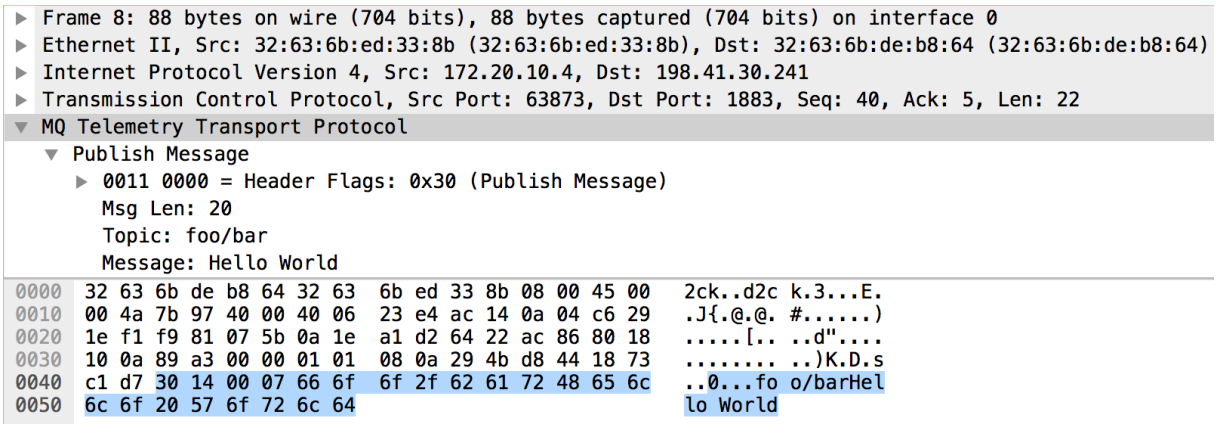
6.3.4.1 Potentially Problematic Scenarios

At least one potentially problematic scenario was identified: In case an adversary is able to compromise the private key of the MQTT broker, it can set up its own MQTT broker using the compromised key and certificate. When it re-routes traffic from customer devices to its own broker, it can trick them to believing, that it is the correct broker. Devices are trying to verify the certificate and since the malicious broker is in possession of a valid certificate and the corresponding private key, the client is trusting it.

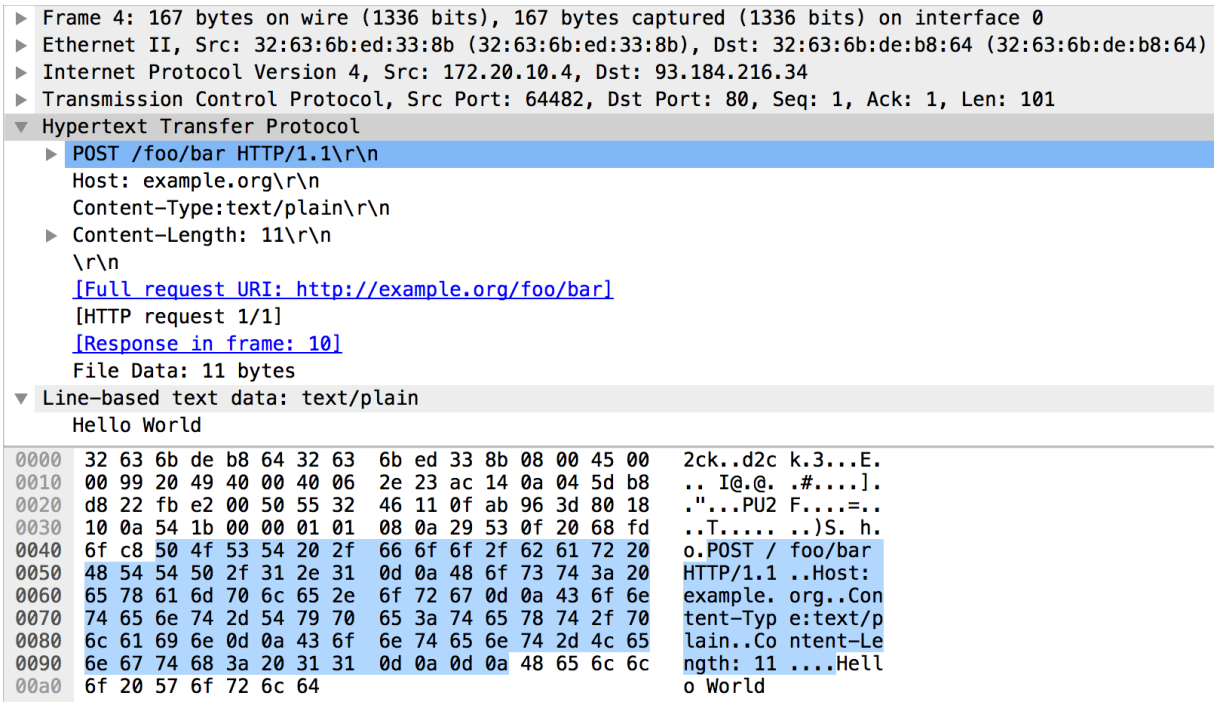
Here, the troublesome circumstance is, that even if the vendor becomes aware of the key being compromised and immediately revokes it, the malicious broker will be trusted. The broker is always the first point the client communicates with and the malicious broker would certainly not deliver the updated CRLs, which includes its own certificate, to the client. Thus, a working MQTT connection, which seems valid to the client, can be established.

However, since additional end-to-end encryption is used, another layer of security is left. The adversary may be able to intercept the device's messages but it can not read them since they are encrypted with CMS. The adversary may also send messages containing remote commands to the devices via the malicious broker but the device simply discards the messages since they are not signed by a trusted originator. This example shows, that it is important, to use two different PKIs for the two security layers. As long as an adversary only gets access to one of them, it cannot do any harm.

[5] `support.apple.com/en-us/HT207189`, Accessed: May 8, 2017



(a) MQTT publish with QoS 0



(b) HTTP POST request

Figure 6.2: MQTT publish with QoS 0 compared to a HTTP POST request. For illustration purposes, the actual payloads of the application protocols have been highlighted in blue in both screenshots. [All screenshots from *Wireshark* made by the author of this thesis.]

6. Discussion

Chapter 7

Conclusion

> *“ The Internet of things is coming, be the disrupter or prepare to be disrupted. There’s no stopping it. ”*
>
> [Joe Tucci, CEO of Dell EMC]

In this chapter, the research and implementation of a SMS is concluded.

While the prototypical implementation was described in this thesis, the transformation of the prototype towards a fully usable production system is in progress at the time of writing. AVL has already attached some of its customers industrial equipment to the SMS framework and is aggregating data.

7.1 Evaluation of the raised Research Questions

In the following list, the research questions which were raised in the beginning of this thesis are evaluated:

- **(RQ1) Which technology can be used, to centrally aggregate and distribute data from globally distributed clients?**
  First of all, to understand and model the components of the SMS framework, it was tried to think of it as a SOA (Section 2.1, Section 6.1). The implementation of the SMS framework in form of a SOA was also a main goal of the project in which this work was carried out.

  In Section 2.2 it was elaborated on data transmission in the IoT. Some popular protocols specifically labeled for IoT scenarios were presented. After comparing the technologies

and its features with traditional protocols, such as HTTP, it became clear that the MQTT protocol and its whole ecosystem are very well suited for the SMS framework.

Chapter 3 described, how one can make use of MQTT in the SMS framework. It was started off with initial considerations about how to structure the topic hierarchy in such a way, that is easily possible for clients to subscribe to desired topics (Section 3.2). Afterwards, it was elaborated on how to intelligently distribute MQTT broker instances in a globally spread SMS framework with potentially tens of thousands of clients (Section 3.3).

- **(RQ2) How can established standards be used to secure data transmission in the SMS framework?**
  First of all, in Section 2.3, some security mechanisms and features, which are important for implementing a secure SMS framework were discussed. Since sensible data is transmitted via insecure data channels – such as the Internet – it was decided to make use of slightly adapted version of TLS as it is in use in the world wide web (Section 2.3.1).

  It turned out that, in order to supply a highly secure system, one would need some additional layer of security on application layer. It was elaborated on the technologies available for end-to-end encryption (Section 2.3.2) and decided to rely on the CMS and its *AuthEnvelopedData* structure (Section 2.3.2.1).

  All of the security mechanisms have been selected in regard to the authentication and authorization issues. These are thoroughly discussed in Section 4.3 as well as Section 4.4. Their use is evaluated in Section 6.3.1 as well as Section 6.3.2.

- **(RQ3) Implement a prototype to proof the elaborated concepts and show a working data transmission from customer devices to the vendor's backend using the elaborated concepts**
  During the implementation phase of the project, a prototype evolved over some three generations. In Chapter 5 the implementation work of the concepts evolved in Chapter 3 and Chapter 4 is described.

  On the one hand, elaborations on how to implement the proposed MQTT topic structure in coherence with the proposed authentication and authorization features have been conducted. On the other hand, a prototypical backend service capable of handling the end-to-end encrypted communication with field clients was shown.

  For the purpose of storing the aggregated data, it was investigated on a simple, relational data schema (Section 5.6).

  Finally, it was thought of some simple workflows and usages of the data which was collected in the prior steps to be able to show the practical use of the system (Section 5.7).

7.2 Problems Solved

With the contributions made during the work on this thesis, it was significantly contributed to the establishment of a SMS framework at AVL. The contents of this thesis are part of highly topical research currently going on in the field of IoT and security for M2M connections.

The concept was proven with a prototypical implementation and were able to make the breakthrough for data transmission from customer devices all the way into a vendor's backend. The data is transmitted in a secure manner, not only while in transit but also when it is processed throughout the SMS framework's network. The published data can also be inspected by other parties than the vendor, such as the customer.

7.3 Lessons Learned

7.3.1 The IoT

What was found from the conducted research and observations is, that the landscape of IoT platforms, protocols and implementations rapidly grew over the last years. One can have a hard time trying to find one's way in the amount of products available.

For the SMS framework, it was aimed to rely on established standards. MQTT was already proposed in 1999[1] and recently became an OASIS [MQTT Version 3.1.1] and ISO [ISO20922] standard. This shows the recognition of the protocol for IoT applications.

7.3.2 Security

The implementation of security features requires careful thinking. It is certainly the most sensitive part of the whole SMS framework. During implementation, one needs to pay attention on every little detail. For example, serious problems can arise from re-using initialization vectors for AES-GCM or using insecure random number generators with predictable output.

To make life easier and avoid common pitfalls, one should not reinvent the wheel and implement cryptographic functions from scratch. Rather, one should rely on well-adapted security libraries such as *OpenSSL*[2] or *Bouncy Castle*.

For the use case, *Bouncy Castle* provided single low-level modules, especially for ASN.1 encoding. However, still lots of efforts were required to combine all those low-level components into an easy-to-use component for specific necessities.

[1] `mqtt.org/faq`, Accessed: May 27, 2017

[2] `www.openssl.org`, Accessed: May 27, 2017

7.4 Next steps

As the feasibility of the SMS framework was proven, the goal is, to deploy it to a productive environment in AVL. However, there are many steps to be taken in order to move the prototype to a fully stable solution for use in a productive environment.

Hence, the following issues need to be addressed:

- **Scalability**
  First of all the desire is, to deploy the service in a highly scalable fashion to cloud architectures. The estimated data amounts to be processed are far larger than for a simple server to handle. The existing field devices of AVL gradually get equipped with Internet connectivity functions, which was developed alongside the prototype implementation by another project group. Ultimately, it is estimated that some tens of thousands of devices are interconnected via the SMS framework.

- **Data storage**
  Furthermore, the database technologies used in the prototype are by far not expected to be sufficient. Instead a data warehouse based on an *Oracle* database system is deemed to be used.

- **Security tests**
  Together with dedicated specialists, the architectural concept needs to be checked against potential leaks and weak spots; especially in terms of security. Thus, penetration tests should be conducted.

- **Certificate management**
  Another challenge for the security of the SMS framework is the certificate management. Over time, thousands of certificates need to be distributed, revoked and replaced. It is still open how this issue should be properly tackled.

- **Key storage**
  Along with certificates many secret keys need to be distributed, both to field devices and servers on vendor premises. These keys are confidential and in case adversaries should be able to compromise them, they can do serious harm to the SMS framework. For this purpose, on client side special hardware security controllers are used [Lesjak, Hein, and Winter, 2015].

- **MQTT**
  For MQTT, the protocol specification version 5[3] has already been drafted and is in review at the time of writing. It brings along useful new features for IoT devices, such as session

[3] `www.oasis-open.org/committees/download.php/59173/mqtt-v5.0-wd08.pdf`, Accessed: May 27, 2017

management or timed message expiration. It is being looked forward to make use of those new features in the SMS framework.

7. Conclusion

Appendix A

MQTT broker comparison

Table A.1 shows the results of our MQTT broker evaluation.

Name	Technology	License	QoS 0	QoS 1	QoS 2	$SYS info	SSL TLS	Bridging	Cluster	WebSocket support	Plugin System
mosquitto	C	open-source	✓	✓	✓	✓	✓	✓	✗	✓	✓
RSMB	C	open-source	✓	✓	✓	✓	✗	✓	✗	✗	?
WebSphere MQ	?	commercial	✓	✓	✓	✓	✓	✓	?	?	?
HiveMQ	Java	commercial	✓	✓	✓	✓	✓	✓	✓	✓	✓
Apache Apollo	Java/Scala	open-source	✓	✓	✓	✗	✓	✗	?	✓	?
Apache ActiveMQ	Java	open-source	✓	✓	✓	✗	✓	✗	✓	✓	✓
RabbitMQ	Javascript	open-source	✓	✓	✗	✗	✓	✗	✓	✓	✓
MQTT.js	node.js	open-source	✓	✓	✓	✗	✓	✗	✗	✓	✗
moquette	Java	open-source	✓	✓	✗	?	✓	?	✗	✗	✗
mosca	node.js	open-source	✓	✓	✗	?	?	?	✗	✓	✗
IBM MessageSight	?	commercial	✓	✓	✓	✓	✓	✗	✗	✓	✗
GnatMQ	.NET	open-source	✓	✓	✓	✗	✗	✗	✗	✗	✗
JoramMQ	Java	commercial	✓	✓	✓	✓	✓	✓	✓	✓	✓
VerneMQ	Erlang	open-source	✓	✓	✓	✓	✓	✓	✓	✓	✓
emqtt	Erlang	open-source	✓	✓	✓	✓	✓	✓	✓	✓	✓
2lemetry	?	commercial	✓	✓	✓	✗	✓	✓	✓	✓	✗
ThingMQ	?	commercial	✓	✓	✓	✗	✓	✓	✓	✓	✓

Table A.1: Comparison of MQTT broker implementations.

Bibliography

Banks, A. and R. Gupta [2014]. *MQTT Version 3.1.1*. Technical report. OASIS Standard, Oct. 2014. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (cited on pages 16, 21, 79, 81, 91).

Barker, E., L. Chen, A. Roginsky, and M. Smid [2013]. *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*. Technical report. May 2013 (cited on pages 28–30).

Cooper, D., S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk [2008]. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. May 2008. http://www.ietf.org/rfc/rfc5280.txt (cited on pages 23, 51, 52).

Delsing, J., editor [2017]. *IoT Automation: Arrowhead Framework*. CRC Press, Feb. 2017. ISBN 9781498756754 (cited on page 12).

Dierks, T. and E. Rescorla [2008]. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. http://www.ietf.org/rfc/rfc5246.txt (cited on pages 26, 27, 65).

Dworkin, M. [2007]. *NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Technical report. 2007. doi:10.6028/nist.sp.800-38d (cited on page 30).

Hardt, D. [2012]. *The OAuth 2.0 Authorization Framework*. RFC 6749. Oct. 2012. http://www.ietf.org/rfc/rfc6749.txt (cited on page 85).

Herterich, M. M., F. Uebernickel, and W. Brenner [2015]. "The Impact of Cyber-physical Systems on Industrial Services in Manufacturing". *Procedia CIRP* 30 (2015), pages 323–328 (cited on page 4).

Holmes, C. [2015]. *Designing and Implementing the Factory of the Future at Mahindra Vehicle Manufacturers*. IDC Customer Spotlight. Apr. 2015 (cited on page 2).

Housley, R. [2007a]. *Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type*. RFC 5083. Nov. 2007. http://www.ietf.org/rfc/rfc5083.txt (cited on pages 28, 54).

Housley, R. [2007b]. *Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)*. RFC 5084. Nov. 2007. http://www.ietf.org/rfc/rfc5084.txt (cited on page 30).

Housley, R. [2009]. *Cryptographic Message Syntax (CMS)*. RFC 5652. Sept. 2009. http://www.ietf.org/rfc/rfc5652.txt (cited on page 28).

ISO 20922: Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1 [2016]. Standard. Geneva, CH: International Organization for Standardization, 2016 (cited on pages 16, 79, 91).

Kagermann, H., J. Helbig, A. Hellinger, and W. Wahlster [2013]. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group*. 2013 (cited on page 1).

Lesjak, C., H. Bock, D. Hein, and M. Maritsch [2016]. "Hardware-secured and Transparent Multi-stakeholder Data Exchange for Industrial IoT". In: *2016 IEEE International Conference on Industrial Informatics (INDIN 2016)*. Futuroscope-Poitiers, France, 2016 (cited on pages 12, 35, 47, 59, 79, 85).

Lesjak, C., D. Hein, M. Hofmann, M. Maritsch, A. Aldrian, P. Priller, T. Ebner, T. Ruprechter, and G. Pregartner [2015]. "Securing smart maintenance services: hardware-security and TLS for MQTT". In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. IEEE. 2015 (cited on pages IX, XI, 11, 13, 47).

Lesjak, C., D. Hein, and J. Winter [2015]. "Hardware-security technologies for Industrial IoT: TrustZone and Security Controller". In: *IECON 2015, 41st IEEE Industrial Electronics Society Conference*. IEEE. 2015 (cited on page 92).

Maritsch, M., C. Kittl, and T. Ebner [2015]. "Sichere Vernetzung von Geräten in Smart Factories mit MQTT [Secure connection of devices in smart factories using MQTT]". In: *Mensch und Computer 2015, Stuttgart*. in German. 2015 (cited on pages 11, 13, 35, 79).

Maritsch, M., C. Lesjak, and A. Aldrian [2016]. "Enabling Smart Maintenance Services: Broker-based Equipment Status Data Acquisition and Backend Workflows". In: *2016 IEEE International Conference on Industrial Informatics (INDIN 2016)*. Futuroscope-Poitiers, France, 2016 (cited on pages IX, XI, 11, 35, 40, 47, 59, 79, 84).

Niruntasukrat, A., C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, and A. Panya [2016]. "Authorization mechanism for MQTT-based Internet of Things". In: *2016 IEEE*

International Conference on Communications Workshops (ICC). May 2016, pages 290–295. doi:10.1109/ICCW.2016.7503802 (cited on page 85).

Porter, M. and J. Heppelmann [2015]. "How Smart, Connected Products Are Transforming Companies". *Harvard Business Review* 93.10 (2015) (cited on pages IX, XI, 2).

Priller, P., A. Aldrian, and T. Ebner [2014]. "Case study: from Legacy to connectivity - migrating industrial devices into the world of smart services". In: *Emerging Technologies and Factory Automation (ETFA), 2014 IEEE International Conference on*. IEEE. 2014 (cited on pages 2, 7, 14).

Saint-Andre, P. [2011]. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120. Mar. 2011. `http://www.ietf.org/rfc/rfc6120.txt` (cited on page 16).

Shelby, Z., K. Hartke, and C. Bormann [2014]. *The Constrained Application Protocol (CoAP)*. RFC 7252. June 2014. `http://www.ietf.org/rfc/rfc7252.txt` (cited on page 16).

The Open Group [2009]. *SOA Source Book*. TOGAF Series. Van Haren Publishing, 2009. ISBN 9789087535384 (cited on page 13).

Tillotson, J. and S. Lundin [2012]. *The Art of Smart Services*. Volume 2. Qualcomm, 2012 (cited on page 4).

Yokotani, T. and Y. Sasaki [2016]. "Comparison with HTTP and MQTT on required network resources for IoT". In: *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*. Sept. 2016, pages 1–6. doi:10.1109/ICCEREC.2016.7814989 (cited on pages 17, 82).