QuantumRISC

Work Package 1, Deliverables 1.1 to 1.3:

Use Cases and Requirements

Industrial Use Cases and Requirements for the Deployment of Post-Quantum Cryptography

Version	1.1
Project Coordination	Fraunhofer Institute for Secure Information Technology
Date	August 24, 2020

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

Authors

- Continental AG:
  - David Noack
  - Steffen Sanwald
  - Marc Stöttinger
- Elektrobit Automotive GmbH:
  - Martin Böhner
- Fraunhofer SIT:
  - Norman Lahr
- Hochschule RheinMain:
  - Thorsten Knoll
  - Steffen Reith
- MTG AG:
  - Evangelos Karatsiolis
- Ruhr-Universität Bochum:
  - Georg Land
- Technische Universität Darmstadt:
  - Juliane Krämer
  - Marcel Müller

Project Coordination
Dr. Ruben Niederhagen
Fraunhofer Institute for Secure Information Technology
Advanced Cryptographic Engineering
Rheinstr. 75
D-64295 Darmstadt
Germany

Phone +49 6151 869 135
Mail ruben.niederhagen@sit.fraunhofer.de

Contents

1 Introduction

1.1 Motivation for Research on Post Quantum Cryptography

Research in the development of powerful quantum computers has made enormous progress in recent years. Figure 1.1 depicts the estimated advancing in the development of quantum computers with the expectation that in 2035 large-scale universal quantum-computer exists. In combination with the existence of efficient algorithms for factorization and solving the discrete logarithm, i.e., Shor's algorithm [Sho97], widespread asymmetric cryptographic schemes are currently considered to be broken soon. Hence, signature schemes like ECDSA and DSA or key exchange algorithms like DH are not considered state-of-the-art and safe to use then, because these schemes are based on asymmetric cryptosystems like ECC or RSA. All of them are vulnerable from a cryptoanalytic perspective then.

Less affected by the existence of large scale universal quantum computers are symmetric cryptographic schemes like ciphers or message authentication schemes. Indeed, Grover's algorithm [Gro96] is an efficient search algorithm that can be executed on quantum computers, but its effect on reducing the search space for symmetric cryptography schemes is less severe. In general, the security of symmetric schemes is halved if an attacker can run an attack using Grover's algorithm on a large scale universal quantum computer. A straightforward countermeasure to this threat is to double the key length of the symmetric schemes to achieve the original intended classical security level. For instance, schemes using AES-128 today should switch to AES-256 to ensure the same resistance to cryptoanalytic attacks in the future as they have today.

This future threat scenario puts one cornerstone, the cryptographic algorithms, of IT security at risk and threatens them to become unusable in the mid to long term. Future products, systems, and infrastructure rely on IT security. Hence embedded systems are affected too. Due to the digitalization of products and services in almost all areas, the connectivity of the components used is increasing. Therefore, the need for advanced cyber security techniques to protect them is high. Especially real-time systems with high connectivity like automotive vehicles utilize cryptographic algorithms to protect them against cyber attacks on asymmetric cryptographic schemes.

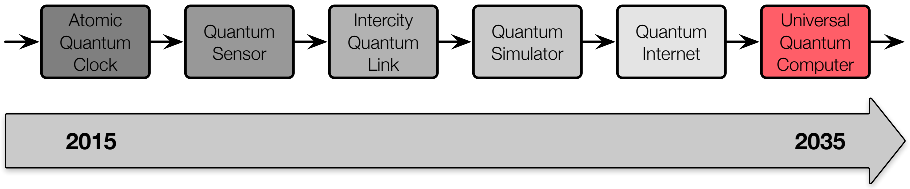

Figure 1.1: Predicted steps towards a universal quantum computer (inspired by [Pre16]).

The National Institute of Standards and Technology (NIST) already started a standardization contest to identify successor algorithms to replace the standardized cryptographic algorithms of RSA and ECC in 2016. However, due to the nature and properties of the families of the novel proposed cryptographic algorithms, there is no single successor algorithm for signature schemes, encryption, and key-exchange to replace RSA or ECC. Up to now, the NIST plans to standardize one algorithm of each of the five existing families. Each family has its advantages and disadvantages due to its properties, and thus not every algorithm might be applicable for every use case. Therefore, the use of case-based evolution is necessary for the right proposal of post-quantum cryptographic scheme and parameter selection of the algorithm.

1.2 Document Structure

This document manifests the first three deliverables of the work package WP1:

- D1.1 - Definition of Use Cases
- D1.2 - Requirements on Post-Quantum Cryptographic Schemes
- D1.3 - Requirements on Hardware Platforms

Each delivery is documented in Chapter 3, Chapter 4, and Chapter 5. Due to the dependencies of these three deliverables, we decided to put them into one report to provide a better overview on the content of the deliverables and present the dependencies among them clearly.

Chapter 2 lists an overview of frequently used abbreviations and terms in the document that are crucial to foster a common understanding of the content of this report.

Chapter 3 provides a collection of use cases and a proposal for a systematical way to present their potential, risks, and chances. Because the threats are common to various branches of industry, special care has been taken that the use cases are selected and presented in an agnostic manner, i.e., not only the interests of the automotive industry are taken into account. Further, so called cluster use cases are defined to group use cases with identical security and model requirements.

Chapter 4 details the different selection criteria on the post-quantum cryptographic schemes. It is obvious that a systematic enumeration of relevant use cases and analysis of those use cases enable the derivation of requirements and selection criteria for post-quantum cryptographic schemes. In addition, the requirements based on the selected use cases also influence the parameter selection of the different post-quantum cryptographic algorithms. At this early stage, of course, the selection criteria can only be very roughly captured and will need to be adapted over time. Thus, a refinement of the selection criteria will have a strong influence on the use cases presented here (and vice versa).

Chapter 5 discusses the required hardware platforms and specifications that are considered for the execution of such post-quantum cryptographic schemes. Not only performance requirements on the hardware platforms will be discussed, but also architectural aspects as well as basic hardware components to accelerate the execution of post-quantum cryptographic schemes in general.

2 Terminology

This section provides an overview of abbreviations and terms used in this document.

Abbreviation	Name	Explanation
CA	Certification Authority	An entity in a PKI that issues digital certificates.
CAN	Controller Area Network	CAN is a serial bus system used in the automotive and automation domain. The bus data is transmitted with a differential signal for robustness reasons.
DMIPS	Dhrystone million instructions per second	A benchmark to estimate the performance of a controller.
ECU	Electronic Control Unit	ECU is the general term for an electronic component in the in-vehicle network with certain functions required by the overall architecture of the vehicle for the correct usage of the vehicle.
HSM	Hardware Security Module	Additional component offering security services to the device or system: In the automotive domain an HSM is an additional IP core inside a chip or microcontroller that offers a security enclave with cryptographic services to the main processor of the chip or microcontroller.
HTTP	Hypertext Transfer Protocol	A communication protocol specifying the messages between servers and clients. It is specified in [FR14].
LDAP	Lightweight Directory Access Protocol	A communication protocol specifying the communication of clients with directory services. It is specified in [Ser06].
OBD	On-Board Diagnosis	The Onboard Diagnosis is a vehicle-specific diagnosis system consisting of standardized diagnosis protocols and interfaces.
OSCP	Online Certificate Status Protocol	A communication protocol used by clients to ask servers whether one or more certificates are revoked. It is specified in [San+13].
PKI	Public-key Infrastructure	An infrastructure including policies, rules, protocols, and instances in hardware and software for handling digital certificates and keys.

PQC	Post-Quantum Cryptography	Cryptographic algorithms that are resilient against attacks exploiting large scale quantum computers.
TCG	Trusted Computing Group	An industrial standardization body for establishing open standards for the trusting computing platform.
TPM	Trusted Platform Module	An additional security chip with security features specified by the TCG to provide security features to the host system.

Table 2.1: Overview of the terminology used in this report.

3 D1.1 - Definition of Use Cases

This chapter provides a comprehensive overview of use cases that rely on asymmetric cryptography and thus need post-quantum cryptographic equivalents as motivated in the introduction. Typically, use cases describe the detailed interaction of a user with the system to accomplish a specific goal, and the entirety of use cases define the features to be implemented for that system. The goal of our list of use cases, however, is not to depict a single and complete system, but to collect a wide range of use cases that might be realized in separate systems, and to assess the practical relevance of post-quantum cryptography in industrial applications. For this reason, use cases that exclusively use symmetric cryptography are omitted. Thus, steps of a use case that neither concern nor are affected by the use of asymmetric cryptography can be kept simpler and more generalized. Consequently, different use cases that share similar mechanics and requirements are combined in cluster use cases.

In the following, we first describe the abstraction model used to make the use cases comparable and derive the requirements needed for an adequate selection of algorithms and parameters in Chapter 4.

3.1 Abstraction Model

Due to the strong focus on security and cryptographic application, it makes sense to use security goals and threats as a frequent basis to describe and compare use cases. It is not feasible to define a system model that would cover all potential use cases. The single characteristic common to all use cases is that there are at least two distinct communication entities within different system environments that are engaged in communication. Therefore, we introduce a simple abstraction model that represents the data control flow and data processing based on a simple sender-receiver model (SRM) for communication. Each use case is evaluated by applying the threat model to the SRM to derive (or specify) security goals. These security goals, in combination with the non-functional characteristics for the involved entities, provide the relevant requirements for algorithm selection.

3.1.1 Simple Sender Receiver Model

The simple sender receiver model (SRM) consists of a sender, a receiver, and a communication channel between them that is used for data exchange. The relation between the entities is modeled using the n-m notation, which provides the following relations: 1-1,1-n and n-m. The relevant threats to a given model are depicted in rectangle brackets $[\cdots]$. Non-functional requirements for a use case are listed below each entity. Figure 3.1 visualizes all possible information that could be given for an SRM, but concrete SRMs differ amongst use cases.

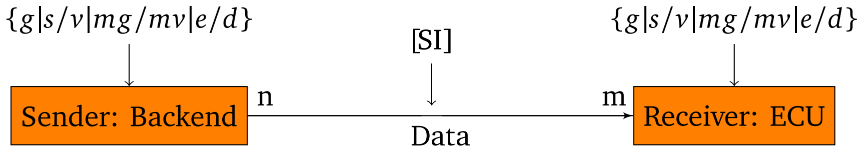
{g|s/v|mg/mv|e/d}
[SI]
{g|s/v|mg/mv|e/d}
Sender: Backend
n
m
Receiver: ECU
Data

Figure 3.1: Simple sender receiver model for abstract visualization of all relevant characteristics.

Abbreviation	Threat	Security Goal
S	Spoofing	Authenticity
I	Information Disclosure	Confidentiality

Table 3.1: Threat Model with security goals and corresponding threats.

The curved brackets over the active entity in the model (the receiver and the sender) indicate the cryptographic operations they perform on the outgoing or incoming data. Considering signature schemes a signature generation operation is depicted $\{s\}$ and a signature verification operation $\{v\}$. For message authentication codes (MAC) generation is represented by $\{mg\}$ and verification by $\{mv\}$. Eventually, $\{e\}$ stands for an encryption and accordingly $\{d\}$ for the decryption operation. Keys generated on a device as part of the use case are represented with $\{g\}$. Next to the name of the active entities is a label to indicate the corresponding component in the use case. For instance, *Sender: Backend* means that the sender in this model is a Backend as practical comportment represented in the use case.

3.1.2 Threat Model

The threat model used to describe the security goals and the corresponding threats for each use case in the given sender-receiver model can be derived by considering the data exchanged as the primary asset to be protected. Hence, as depicted in Table 3.1, the two relevant security goals are authenticity and confidentiality, and spoofing or information disclosure are the respective threats.

Authenticity ensures that the sending entity can prove she is whom she claims to be. Spoofing, the threat towards authenticity, is successful if an attacker can convince the communication partner to be another identity. Authenticity is usually implemented using signature schemes. Confidentiality ensures that only the receiving entity, to which a message was intended for, can read that message. Accordingly, if an attacker can read the information that should have been confidential, the corresponding threat information disclosure was realized. Confidentiality is usually implemented using encryption. Both security goals would be immediately compromised if an attacker gained access to the corresponding secret keys for either signature generation or decryption. Hence, integrity is inherently a vital security goal for any key material. However, since it has no impact on the selected cryptographic algorithms, and we evaluate use cases at the communication level rather than the system level, we do not consider it in our threat model. We, therefore, start from the general requirement of integrity protection of key material. Integrity on the communication level is inherited by authenticity. Furthermore, replay attacks (threat to the security goal of freshness) can be a threat to particular use cases, but would be mitigated on the protocol level, have no impact on PQ-resilience and thus are also not considered.

3.1.3 Non-Functional Requirements and Properties

Some aspects of the use cases are essential for the characterization but not reflected in this abstraction model. These aspects are non-functional requirements as well as properties of the use case or the involved entities, respectively, that have an impact on security and execution. All properties are evaluated independently for each participant of a use case and in the scope that imposes the most limiting or demanding factor. As an example, in a typical scenario of a backend ministering a large number of client devices, the single execution of the use case is the limiting scope for each device. In contrast, the total number of devices and parallel executions of the use case is the limiting scope for the backend. The potential impact of the properties of our requirements is listed in Table 3.2.

ID	Requirements	Explanation	Defined set of constraints for each requirement
1	Latency	How much time do the cryptographic operations have for executing all necessary tasks of the use case?	high (1hrs), medium (few minutes), low (<5 s), very low(<1 s), very very low (<10 ms)
2	# Executions over product life time	Is there an upper limit for the cryptographic operations performed with the same key pair during the life time of the entity?	limited, unlimited
3	# Key pairs	How many different key pairs exist in parallel independent from data dissipation?	≤10, ≤1000, ≤100.000, ≤1.000.000, >1.000.000
4	Data dissipation	What is the data exchange or activity scenario between the sender(s) and receiver(s)?	one-to-one, one-to-many, many-to-one, many-to-many
5	Life time of a cryptographic artifact	How long shall a signature or a cipher text considered to be secure?	≤1 hour, ≤1 day, ≤1 week, ≤1 months, ≤1 year, ≤5 years, >5 years
6	Current security level	How many classic security bits does this use case require?	≤64 bits, ≤80 bits, ≤128 bits, ≤192 bits, ≤256 bits
7	Size of processed data	How large is the processed input data of the cryptographic operation?	≤8 B, ≤16 B, ≤32 B, ≤64 B, ≤1 KB, ≤1 MB, ≤100 MB, ≤ 1 GB, >1 GB
8	Physical accessibility	Could an attacker get physical access to the operation environment to perform side-channel analysis?	restricted, accessible
9	Computational power	What is the computation power of the entity?	deeply embedded (≤250 DMIPS), embedded (≤700 DMIPS), industry PC (≤2.000 DMIPS), IT (≤20.000 DMIPS), Server (>20.000 DMIPS)
10	RAM availability	How much RAM is available on the entity?	deeply embedded (≤32 KB), embedded (≤256 KB), industry PC (≤1 MB), IT (≤1 GB), Server (>1 GB)
11	Storage availability	How much non-volatile memory is available on the entity?	deeply embedded (≤2 MB), embedded (≤4 MB), industry PC (≤32 MB), IT (≤1 GB), Server (>1 GB)

Table 3.2: Non-functional requirements of a use case with a corresponding set of constraints.

3.2 Use Cases: Authenticated One-to-Many

This use case cluster regards an authenticated one-to-many relation between one sender and multiple receivers (cf. Figure 3.2). In all use cases of this section, the sender (server) is always a powerful resource, either a cloud backend or a certification authority (CA). In contrast, the properties of the receivers are use case dependent. Data is transmitted from the sender to the receiver, but not necessarily in parallel or within certain time bounds. These use cases require any receiver to verify the authenticity of the transmitted data. Consequently, common preconditions are that the public key of the sender has been rolled out securely and is tamper-protected at the receiver.

Table 3.3 shows ranges between the worst case and the best case of the requirements of all sub use cases. In all sub use cases, the sender has the computational properties of a server and runs in a trusted operation environment. In contrast, the receiver runs in the untrusted field with varying computation properties between deeply embedded and regular IT systems. The latency on both sides varies between low and medium. The number of executions over the product lifetime varies between less than 128 and 1024 executions on the sender and between less than 128 and an unpredictable number on the receiver side. The lifetime validity of the cryptographic artifacts is rather long and does not require more frequent updates than one per year.

Exemplary use cases are downloading (updated) software securely to an ECU (Section 3.2.1), feature activation, where already installed software on the ECU is enabled during runtime (Section 3.2.2), and publishing certificate revocation lists securely (Section 3.2.3).

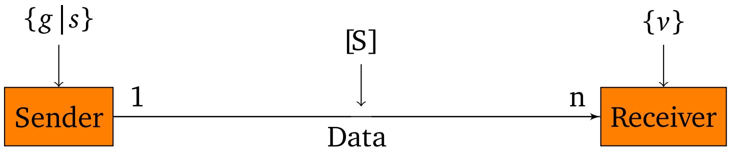

Figure 3.2: Communication model of the use case authenticated one-to-many.

Properties	Sender	Receiver
Latency	Low : Medium	Low : Medium
# Executions over product lifetime	limited : unlimited	limited : unlimited
Size of processed data	≤64B : >1GB	≤64B : >1GB
Physical accessibility	restricted	accessible
Computational power	Server	Deep. Embedded : IT
RAM availability	Server	Deep. Embedded : IT
Storage availability	Server	Deep. Embedded : IT
# Key pairs	≤10	
Data dissipation	one-to-many	
Life time	≤1 year : >5 years	
Current security level	≤128 bits : ≤256 bits	

Table 3.3: Requirement ranges of the use cases authenticated one-to-may.

3.2.1 Secure Download

The secure download mechanism enforces that only authenticated software is flashed to an ECU. Asymmetric signatures prevent that an attacker can flash manipulated software to an ECU. The signature is deployed, along with the software, to be flashed. Therefore, it is signed with a private key in a trusted environment (e.g., the backend). The corresponding public key used for the verification must be enrolled in the ECU and stored tamper-protected in secure storage.

For flashing software, the ECU usually boots into a dedicated bootloader. There, it receives the software from a defined source (e.g., the communication unit or the onboard diagnosis interface) and downloads the software to the memory of the ECU and adjusts the boot flags. The secure software download mechanism is part of the bootloader and performs a signature verification before flashing the software. Only if the signature verification succeeds, the bootloader flashes the software [WS09].

Threat Model

The transmitted data contains the code and configuration data of the ECU and must only be accepted if coming from an authenticated backend. The according SRM is shown in Figure 3.3, where the backend generates the signature, and the ECUs are verifying it, thus ensuring authenticity.

Properties and Requirements

For this use case, the properties and requirements are derived for implementing secure software download in the automotive domain on an ECU and shown in Table 3.4.

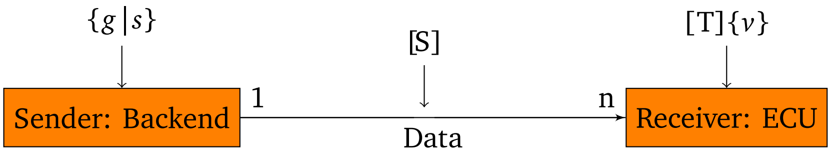

Figure 3.3: Communication model of the use case secure software download.

Properties	**Backend**	**ECU**
Latency	Medium	Low
# Executions over product lifetime	limited	limited
Size of processed data	≤1MB: >1GB	≤1MB: >1GB
Physical accessibility	restricted	accessible
Computational power	Server	Deeply embedded
RAM availability	Server	Deeply embedded
Storage availability	Server	Deeply embedded
# Key pairs	≤10	≤10
Data dissipation	one-to-many	
Life time	>5 years	
Current security level	≤192 bits : ≤256 bits	

Table 3.4: Requirements of the use case secure software download.

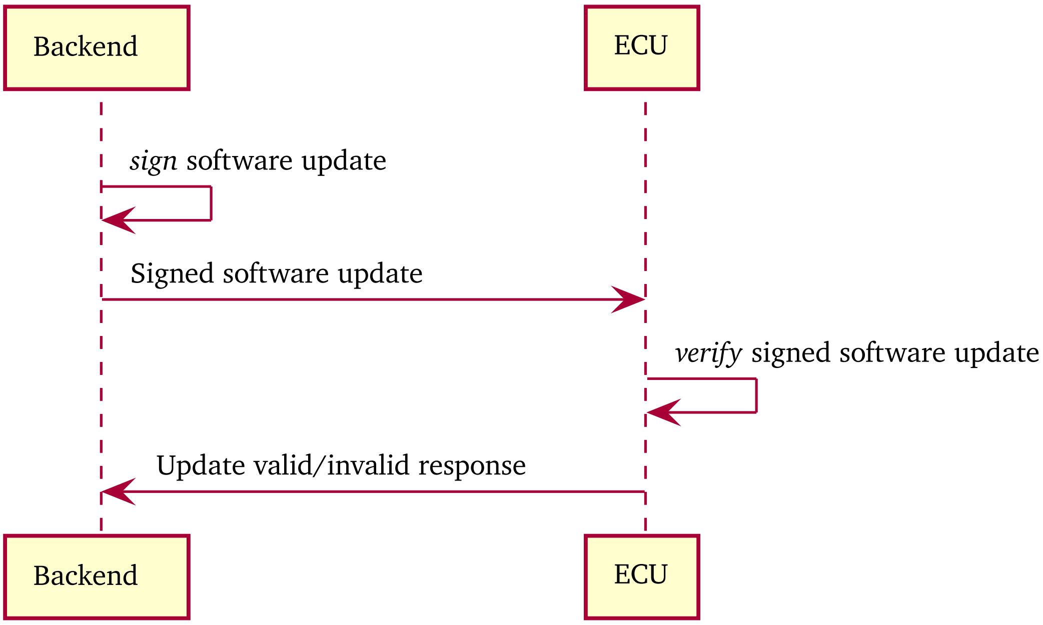

Figure 3.4: Sequence diagram of the secure software download use case.

3.2.2 Feature Activation

With increasing connectivity and digitalization of vehicles, functionality is increasingly controlled by software and as a consequence can also be controlled remotely. This enables new business models, such as pay-on-demand services for different functional features of a device. For example, a manufacturer of sports cars could offer to enable additional power in 30-minute packets. In general, the features that can be requested on demand are realized in software and the corresponding software section gets activated or deactivated by request from the backend. Usually, the initially deployed software of a device/ECU already contains all functionalities, but only a certain set is activated due to a policy set.

When the user buys or rents a feature on demand, and after a feature activation acknowledgement from the backend, the policy setting in the ECU gets updated via a specific message. This message can be transmitted to the ECU via a physical connection or via a remote connection. The most common approach is that the policy change is realized by a certificate that is signed by a backend entity. In this setup, the number of signatures generated by a dedicated private public key pair is finite and predictable over the life time of the vehicle. The certificate shall contain some sort of time stamp to enable the check for freshness. In the case of a time-limited feature activation, the feature shall be deactivated after a certain period of time, which either requires a secure source of time or a tamper-resistant timer in the vehicle, or the explicit deactivation by another message from the backend after the elapsed time.

In addition, the activation of features does not need a one-to-one activation and in certain scenarios the feature activation might be broadcast to many identical devices. One example for this case was the range extension feature Tesla activated when hurricane Irma hit Florida. Tesla extended the range of all Tesla models in the Florida region for a certain period of time to support the evacuation, [Lip17]. Figure 3.6 depicts the process steps of performing an activation of an additional feature of a device that is already deployed into field operation.

Threat Model

Mapped to the sender-receiver model visualized in Figure 3.5, the sender represents the backend, the ECU is the receiver and is required to verify the authenticity of the activation message to mitigate spoofing, which in this case would typically be financially motivated.

Properties and Requirements

Table 3.5 depicts the non-security related requirements for implementing the use case. For this use case, the properties and requirements are derived for implementing feature activation in the automotive domain on an ECU.

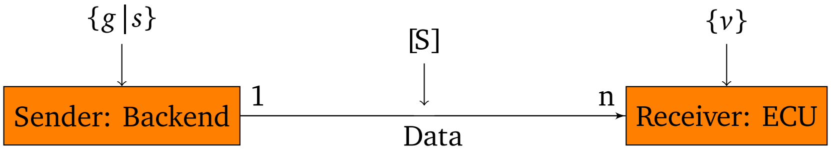

Figure 3.5: Communication model of the use case feature activation.

Properties	**Backend**	**ECU**
Latency	Low	Low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤1MB	≤1MB
Physical accessibility	restricted	accessible
Computational power	Server	Embedded
RAM availability	Server	Embedded
Storage availability	Server	Embedded
# Key Pairs	≤10	≤10
Data dissipation	one-to-many	
Life time	<1 year	
Current security level	≤128 bits	

Table 3.5: Requirements of the use case feature activation.

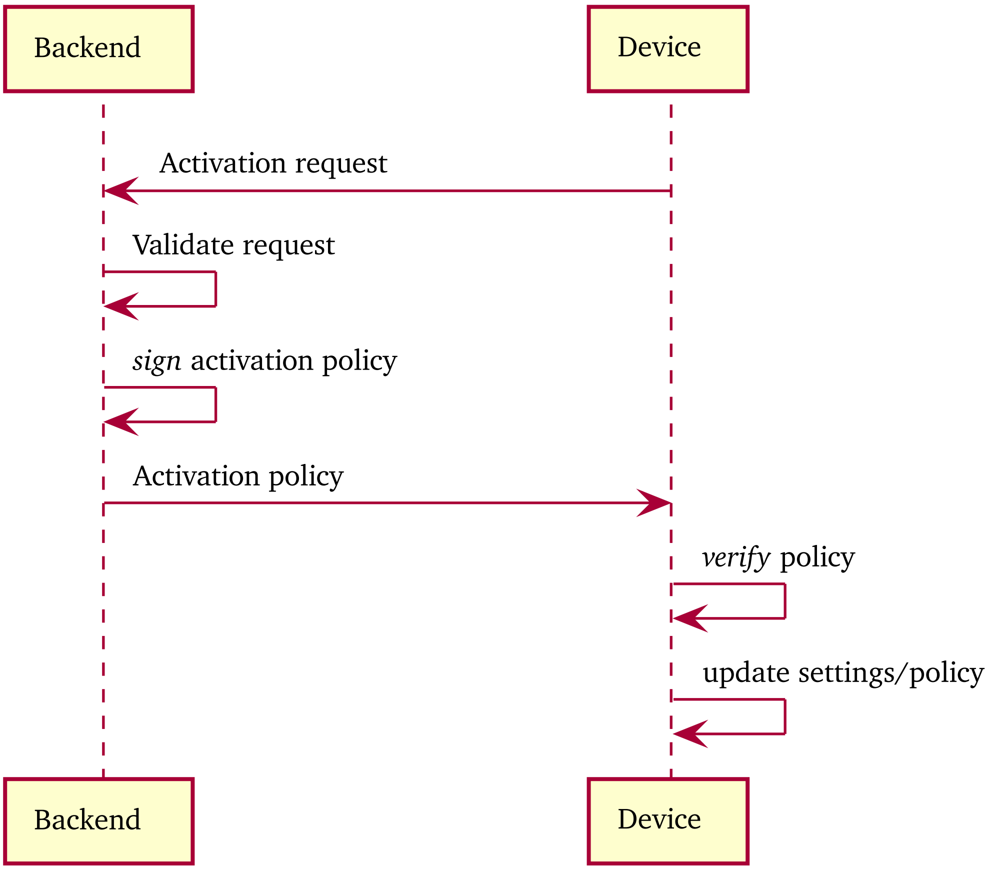

Figure 3.6: Sequence diagram of the use case feature activation.

3.2.3 Certificate Revocation List LDAP/HTTP

In this use case, the participants of a PKI download a certificate revocation list (CRL) from an LDAP or HTTP server. A CRL is a list signed by the CA that contains certificates that are revoked. Before using a certificate, the users check whether this certificate is included in the CRL or not. If it is included, it is revoked and deemed invalid by the users.

Threat Model

Even though communication is triggered by an end-entity requesting the list from the server, we consider the CA to be the sender, since it signs the CRL, as shown in Figure 3.7. The LDAP/HTTP Server is considered as part of the communication channel. The threat to this use case is an attacker managing to provide a manipulated CRL, excluding certificates that have been revoked, thereby spoofing the CA.

Properties and Requirements

Table 3.6 depicts the non-security related requirements for implementing the use case.

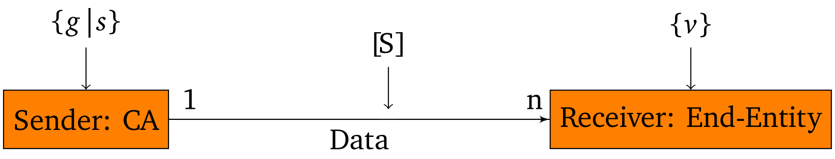

Figure 3.7: Communication model of the use case feature activation.

Properties	**CA**	**End-Entity**
Latency	Medium	Medium
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤1MB	≤1MB
Phyisical accessibility	restricted	accessible
Computational power	Server	IT
RAM availability	Server	IT
Storage availability	Server	IT
# Key pairs	≤10	≤10
Data dissipation	one-to-many	
Life time	>5 years	
Current security level	≤128 bits	

Table 3.6: Requirements of the use case certificate revocation list LDAP/HTTP.

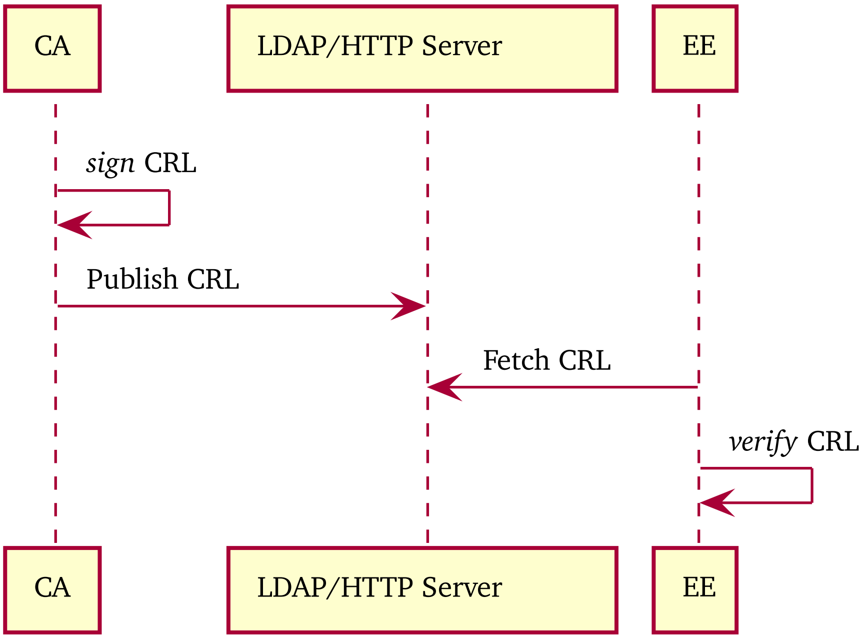

Figure 3.8: Sequence diagram of the use case certificate revocation list LDAP/HTTP.

3.3 Use Cases: Encrypted & Authenticated One-to-Many

In addition to the previous cluster, the following use cases also require confidentiality of message exchange in a one-to-many relation between sender and receivers (cf. Figure 3.9). In all use cases of this section, the sender is always a powerful server. In contrast, the properties of the receivers are use case dependent. Data is transmitted only from the sender to the receiver and is always encrypted in addition to being signed. The receivers shall be able to verify the origin and integrity of the transmitted data, which is realized by an asymmetric cryptographic signature. The sender generates the signature over the payload data using its private key and sends the data along with the signature to the receivers. On reception of the message, the receivers can verify the signature using the corresponding public key.

As the use cases all describe encrypted data, a symmetric key needs also to be produced in general. In the case of a server encrypting data for later use, this key is generated on the server. In cases where a secure channel is to be opened, a key exchange algorithm needs to be used.

Table 3.7 compares the worst case and the best case of the requirements of all sub use cases. In all sub use cases, the sender has the computational properties of a server and runs in a trusted operation environment. In contrast, the receiver runs in the untrusted field with varying computational properties between deeply embedded and standard IT systems. The latency on both sides varies between low and medium. The number of executions over the product's life time varies between less than 128 and 1024 executions on the sender and between less than 128 and an unpredictable amount on the receiver side. The lifetime validity of the cryptographic artifacts of the use cases is less than one year with the exception of the use case in Section 3.3.3.

Exemplary use cases are confidential configuration (Section 3.3.1), encrypted software updates (Section 3.3.2), flashing via on-board diagnosis (Section 3.3.3) and distributing encrypted software to target devices in the field without attaching the secret for decryption (Section 3.3.4). Whereas sharing the secret to a defined later time enables us to decouple the distribution from the actual release of the software, for example, to balance the server load over multiple days/weeks.

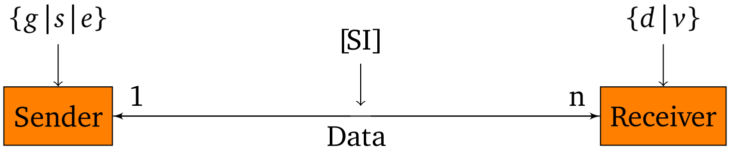

Figure 3.9: General communication model of encrypted and authenticated one-to-many use cases.

Properties	**Sender**	**Receiver**
Latency	Low: High	Low: Medium
# Executions over product lifetime	limited:unlimited	limited:unlimited
Size of processed data	≤64B: >1GB	≤64B: >1GB
Physical Accessibility	restricted	accessible
Computational power	Server	Deep. Embedded: IT
RAM availability	Server	Deep. Embedded: IT
Storage availability	Server	Deep. Embedded: IT
# Key pairs	>1.000.000	≤10
Data dissipation	one-to-many	
Life time	≤1 year	
Current security level	≤192 bits : ≤256 bits	

Table 3.7: Requirement ranges of the use cases encryption & authenticated one-to-may.

3.3.1 Confidential Configuration

Servicing a device containing private information should be done in a way such that the private data can only be accessed by authorized clients. This confidential configuration mechanism goes two ways, an attacker should not be able to see which data is accessed nor the data itself once it is relayed back. The data is usually accessed by way of a remote server, or a local diagnosis tool. The onboard device would need to be able to verify that whoever is accessing it is authorized to do so. Such an authorization could stem from a temporary certificate signed by an authorized authority. The onboard device would then check whether the certificate is still valid and initiate a transfer using it as proof. This means that the server/diagnosis tool needs to be able to generate certificates, and send them to be signed to the correct authority. The authority thus needs to be able to sign an arbitrary amount of times.

Threat Model

Mapped to the sender-receiver model visualized in Figure 3.10, the sender represents the device that requests confidential data from the client, which may be a server or even a deeply embedded device. The communication may pass through multiple hops, without revealing any information of what had been transmitted. Hence, the security goals are authenticity and confidentiality for the transmitted data.

Properties and Requirements

The use case is generic over which sender-receiver pair it operates on. Consequently, direct requirements cannot be formulated. However, the possible range of values is presented in Table 3.8.

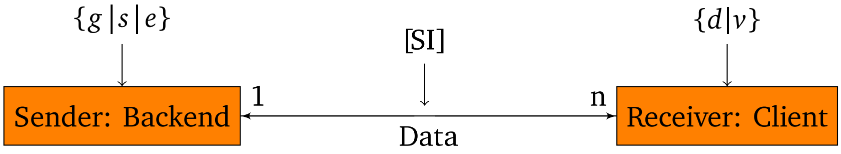

Figure 3.10: Communication model of the use case confidential configuration.

Properties	**Backend**	**Client**
Latency	Medium	Low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤1GB	≤1GB
Physical accessibility	accessible	accessible
Computational power	Server: Embedded	Server: Deeply embedded
RAM availability	Server: Embedded	Server: Deeply embedded
Storage availability	Server: Embedded	Server: Deeply embedded
# Key pairs	>1.000.000	≤10
Data dissipation	one-to-one	
Life time	≤1 year	
Current security level	≤256 bits	

Table 3.8: Requirements of the use case confidential configuration.

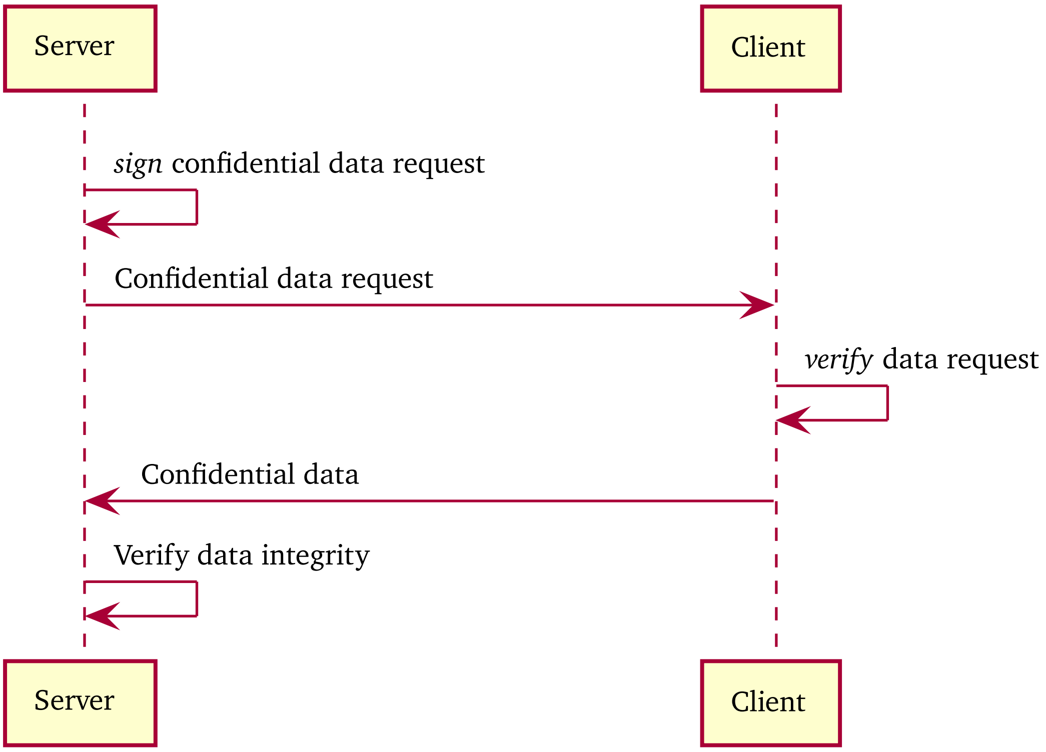

Figure 3.11: Sequence diagram of the use case confidential configuration.

3.3.2 Encrypted Software Update

Updating software is not always done by the developer of that software. Instead, it may be distributed to intermediaries who then are tasked to update relevant devices: the clients. This kind of update helps in cases where a central server would have too high a load, or when the expected environment of the update does not have the required connectivity (be it connection at all, or download speed).

Such an update needs to be encrypted at rest, and therefore it differs from a 'simple' update. The update needs to also be verifiably tamper-proof and authenticated. Hence the client can verify that the update is indeed valid.

Threat Model

Mapped to the sender-receiver model visualized in Figure 3.12, the sender represents the server publishing an update. It signs and encrypts it with a key that is accessible to the client only. The result is then made available for download to intermediaries, who then store the update until it is no longer required. At a later time, the intermediary sends the downloaded update to the client device, which then verifies the integrity and authenticity of the update before applying it. Thus, ensuring the authenticity and confidentiality of the update.

Properties and Requirements

The use-case is generic over which sender-receiver pair it operates on. Consequently, direct requirements cannot be formulated. However, the possible range of values is presented in Table 3.9.

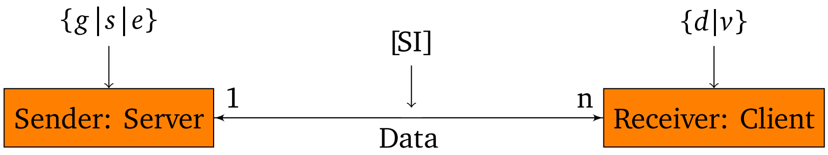

Figure 3.12: Communication model of the use case confidential configuration.

Properties	**Server**	**Client**
Latency	N/A	N/A
# Executions over product lifetime	limited	limited
Size of processed data	>1GB	>1GB
Physical accessiblity	accessible	accessible
Computational power	Server: Embedded	Embedded: Deeply embedded
RAM availability	Server: Embedded	Embedded: Deeply embedded
Storage availability	Server: Embedded	Embedded: Deeply embedded
# Key pairs	>1.000.000	≤10
Data dissipation	one-to-one	
Life time	≤1 year	
Current security level	≤256 bits	

Table 3.9: Requirements of the use case encrypted update.

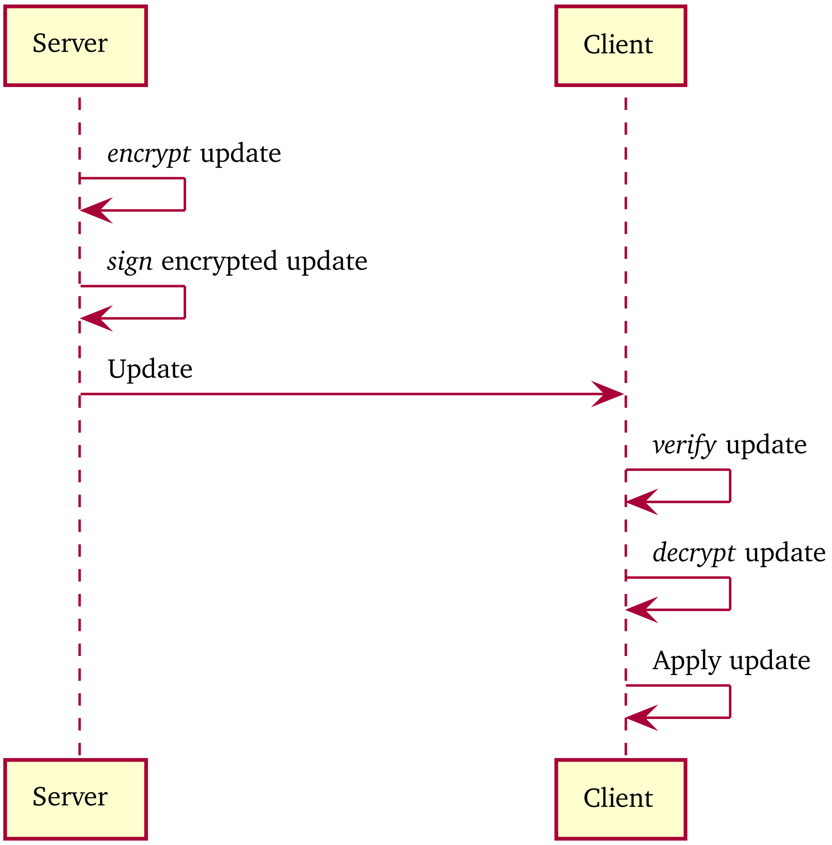

Figure 3.13: Sequence diagram of the use case encrypted update.

3.3.3 Flashing via Onboard Diagnosis

Closely related to the secure software download outlined above is the EVITA use case for (diagnosis) flashing per on-board diagnosis (OBD). The EVITA specification describes the procedure of flashing new firmware to an ECU as follows: "A car owner takes his car to the area of a service station. To start the diagnosis session, the car has to be activated. The ECU initializes its software and starts the diagnosis function, called diagnosis server. In this state, the diagnosis server is in the default mode (this is defined as a session in [ISO13]). The service station employee connects his diagnosis tool to the on-board diagnosis interface in the vehicle. Such a connection is established by plugging a cable to the diagnosis connector, which is different from car to car. A diagnosis request is then sent via the Communication Unit CU (on-board diagnosis interface) to the ECU. The ECU authenticates the diagnosis tool and checks data integrity. If the request is successful, the ECU opens a programming session. The service station employee begins his diagnosis by checking the ECU type and firmware version. Assuming the ECU type is known, a comparison is also made to figure out the need for an update of the version. The diagnosis tool then sends the encrypted packets of the new firmware to the ECU, which stores it in the RAM. The new firmware is decrypted at ECU level and flashed in the ROM packet wise. The date of the update is written in the ECU. Finally, the programming session is closed by sending an EcuReset request to the ECU." [Eur09].

Threat Model

This use case description focuses on the firmware enrollment from the diagnostic tool to the ECU, which requires authenticity to ensure that no malicious firmware updates can be installed and confidentiality to protect the IP of the firmware. The communication unit (diagnosis tool) is abstracted as part of the communication channel, as the backend has to encrypt the update package.

Properties and Requirements

Table 3.10 depicts the non-security related requirements for implementing the use case. The given categorization is derived for implementing flashing via on-board diagnosis in the automotive domain on an ECU.

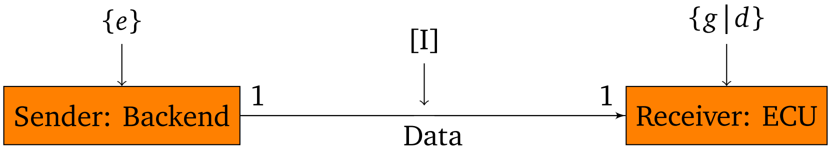

Figure 3.14: Communication model of the use case flashing via onboard diagnosis.

Properties	Server	ECU
Latency	High	Low
# Executions over product lifetime	limited	limited
Size of processed data	≤1MB: ≤1GB	≤1MB: ≤1GB
Physical accessibility	restricted	accessible
Computational power	Industry PC	Deeply embedded
RAM availability	Industry PC	Deeply embedded
Storage availability	Industry PC	Deeply embedded
# Key pairs	≤10	≤10
Data dissipation	one-to-one	
Life time	≤5 year	
Current security level	≤192 bits	

Table 3.10: Requirements of the use case flashing via onboard diagnosis.

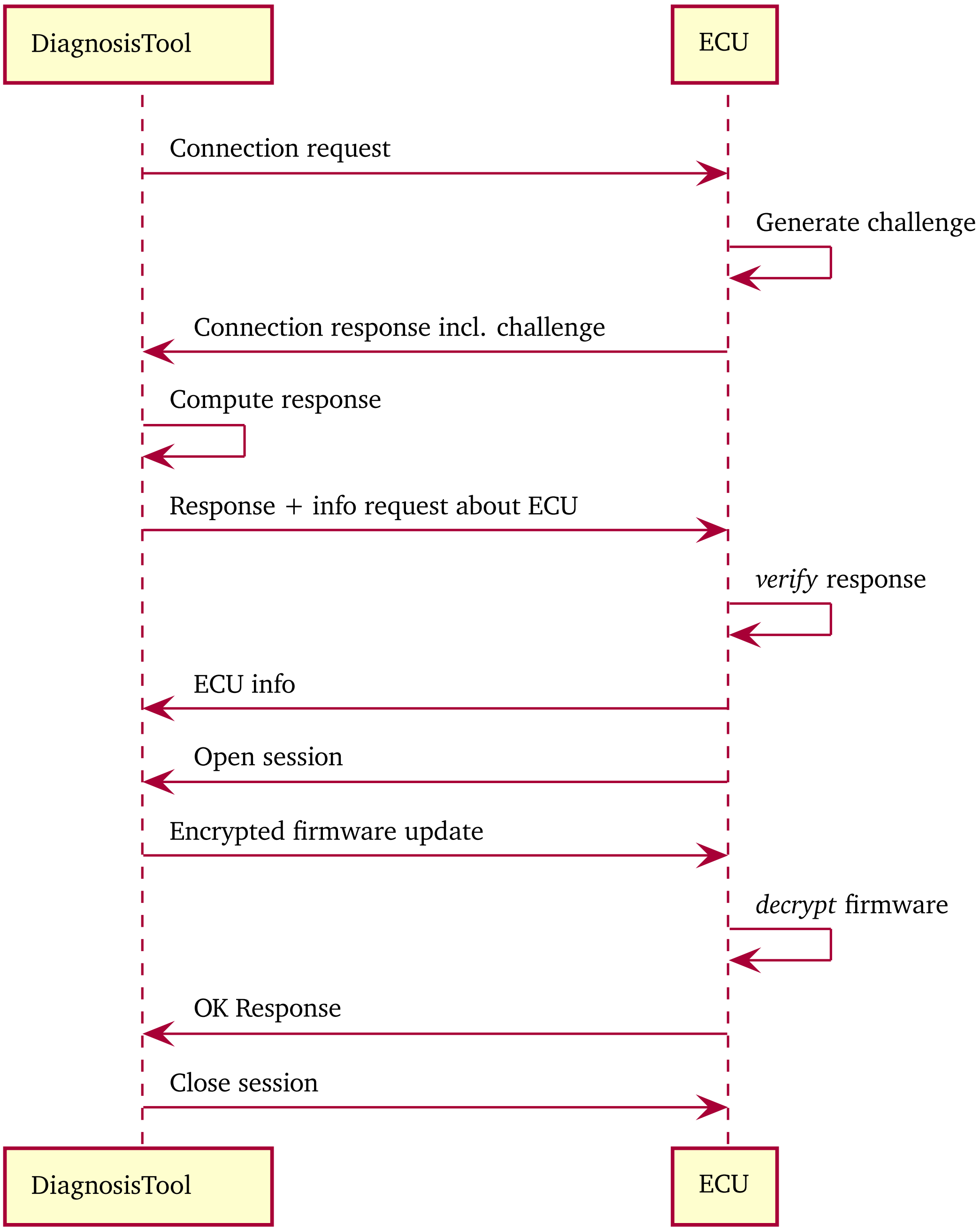

Figure 3.15: Sequence diagram of the flashing via onboard diagnosis use case.

3.3.4 Preloading Updates

Routine updates are often scheduled to be available to all clients at the same time. However, downloading the update to all clients at the same time can quickly exceed the available bandwidth. Limitations in download speed or even delaying downloads for different user groups are detrimental to customer satisfaction. Silently distributing updates prior to the scheduled time can mitigate these limitations and allow for parallel activation. To ensure authenticity updates must be signed, while encrypting them with a symmetric key, ensures that they also cannot be installed or inspected beforehand. For activation only the decryption key needs to be distributed, which requires significantly less bandwidth. Signing the decryption key ensures that the key comes from the same trusted source as the update. As a precondition the relevant root certificates must be enrolled to the devices to verify and trust the signature. Additonal encryption of the decryption key is not necessary since confidentiality of the update is only required until the key is distributed. Figure 3.17 visualizes this workflow.

Threat Model

In terms of the SRM (cf. Figure 3.16), the sender is always a backend distributing the software update, which needs to be authentic. This is ensured by a dedicated signed encryption key and a signature on the update itself. Different possible device classes ranging from server to deeply embedded can occur as a receiver, and depending on their capabilities, key pairs for encryption must either be generated on the device or securely flashed in a trusted environment.

Properties and Requirements

The use case is generic over which server-client pair it operates on. Consequently, direct requirements cannot be formulated. However, the possible range of values is presented in Table 3.11.

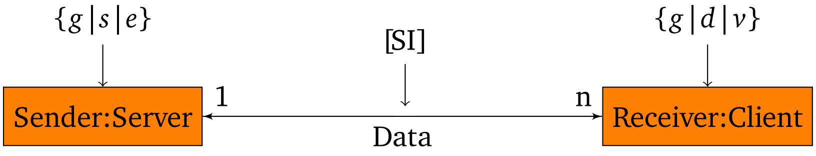

Figure 3.16: Communication model of the use case confidential configuration.

Properties	Server	Client
Latency	High	Medium
# Executions over product lifetime	limited	limited
Size of processed data	≤1MB: ≤1GB	≤1MB: ≤1GB
Physical accessibility	restricted	accessible
Computational power	Server	Server: Deeply embedded
RAM availability	Server	Server: Deeply embedded
Storage availability	Server	Server: Deeply embedded
# Key pairs	> 1.000.000	≤10
Data dissipation	one-to-many	
Life time	≤1 year	
Current security level	≤192 bits	

Table 3.11: Requirements of the use case preloading updates.

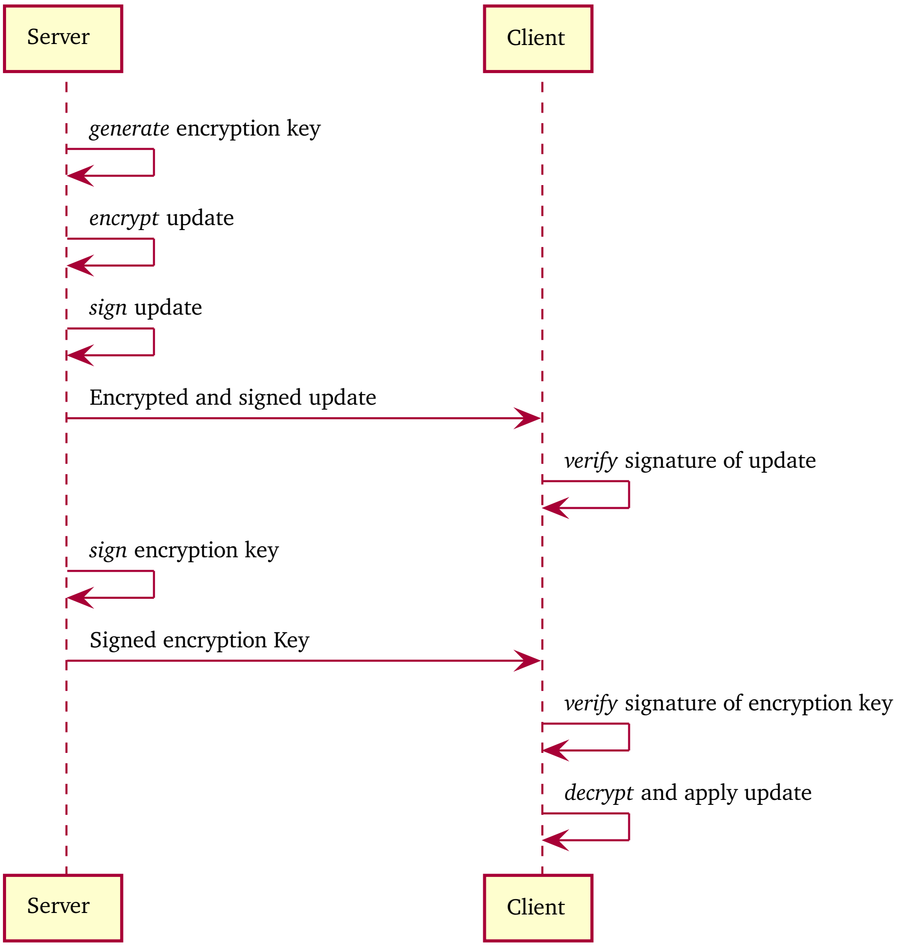
Server
Client
generate encryption key
encrypt update
sign update
Encrypted and signed update
verify signature of update
sign encryption key
Signed encryption Key
verify signature of encryption key
decrypt and apply update
Server
Client

Figure 3.17: Sequence diagram of the use case update preloading.

3.4 Use Cases: Authentication Check with Low Latency

This cluster section consolidates use cases for authentication checks with the constraint of low latency. In particular, the target is to verify the integrity of firmware/software images and files of embedded systems to detect manipulations by an attacker. While Section 3.4.1 verifies the integrity during the boot process, Section 3.4.2 verifies the integrity during the runtime of the system. As both use cases pursue a similar objective, both can be implemented with the same symmetric and asymmetric cryptographic operations. To cover both characteristics in the PQ-resilience, Section 3.4.1 is described with asymmetric operations and Section 3.4.2 is described with symmetric operations.

Regarding common security goals in the abstracted SRM in Figure 3.18, the sender always aims to protect the integrity and authenticity of the data, and the receiver verifies it. Depending on the architecture, the sender can be either on the same system as the receiver or outside. In the latter case, the authenticity of the data can be used and verified on multiple systems.

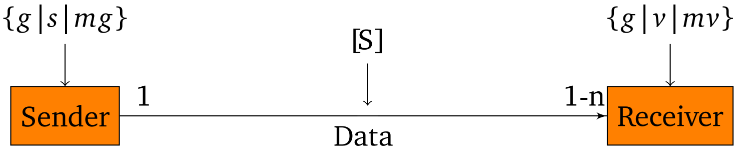

Figure 3.18: Communication model of the abstract use case of authentication check with low latency.

Properties	Device	Device
Latency	Very low	Very low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	<1KB: >1GB	<1KB: >1GB
Physical accessibility	accessible	accessible
Computational power	Embedded	Embedded
RAM availability	Embedded	Embedded
Storage availability	Embedded	Embedded
# Key pairs	≤10	≤10
Data dissipation	one-to-one	
Life time	<1 months	
Current security level	≤192 bits	

Table 3.12: Requirements of the cluster use case authentication check with low latency.

3.4.1 Secure Boot

Secure boot is a mechanism to ensure the software integrity of a device. Before code instructions are executed, the secure boot mechanism cryptographically verifies the integrity and authenticity of the code block in memory. The goal is to gain trust in the integrity of the executed software. Moreover, manipulation of software can be detected during the boot phase of the ECU to prevent its execution. Modern ECUs can contain huge software stacks and are often booted in multiple stages for performance reasons. The secure boot workflow has to verify each boot stage before passing control to it. The trust anchor is a tamper protected immutable portion of code, which is the first code executed after ECU reset and has to be trusted. The trust anchor verifies the first boot stage and, after succeeding, passes control to it. Forming a chain of trust, where each boot stage trusts its predecessor, each boot stage verifies the integrity of its successor until the ECU is booted completely [Kan+19]. There are three secure boot verification schemes: hash-based, MAC-based, and signature-based verification. Considering the relevance of PQC, this use case solely focuses on signature-based verification, which relies on asymmetric cryptography. In this verification scheme, the binary blob containing all executable code instructions and configuration data is deployed to the target ECUs along with a cryptographic signature over the blob. The signature is securely generated with a private key in a protected backend and is often enveloped in a certificate. The corresponding public key is deployed to the target ECUs during production and stored tamper protected [Kan+19]. During the boot phase of the ECU, the secure boot workflow comprises of two main operations: a) computing the hash value over the binary blob in the memory of the ECU and b) comparing the output hash value against the given hash value of the according signature in memory and verifying the signature with the public key. If the verification fails, secure boot shall enforce active countermeasures, e.g., not booting the ECU [Kan+19]. Figure 3.20 depicts the process steps of performing a secure boot on an ECU. By using signature verification on each boot stage, the secure boot workflow checks the authenticity and inherently also integrity of the software, starting from the trust anchor.

Besides asymmetric signature verification, the secure boot can be implemented with symmetric MAC verification, too. This approach is often used to minimize the latency of the integrity verification. In the symmetric approach, the MAC tags are often generated and verified on the same machine using a device-specific key[HiS09].

Threat Model

The threat model included in the SRM for the use case secure boot with asymmetric cryptographic algorithms are depicted in Figure 3.19. In the model, the sender represents the backend, which generates the signature over the binary blob of the ECU code and configuration using its private key. Thus, the sender has the security goals of integrity, authenticity, and confidentiality. The transmitted data is the binary blob containing the code and configuration data of the ECU along with the signature. The data can be distributed to the ECU in several ways, e.g., secure download, production, OTA, and, thus, are omitted in Figure 3.20. The data need to be transmitted tamper-protected and often requires freshness, e.g., it should not be possible to deploy an older version of the ECU firmware. Each receiver represents an ECU that performs the actual secure boot workflow. The signature verifications of each boot stage lead to the security goal of integrity. For some scenarios, the signed firmware or software is additionally encrypted to prevent disclosure of information. In this case, the transmitted data (the firmware or software) needs to be encrypted by the sender and decrypted by the receiver. Hence, for this scenario, the receiver has the additional security goal confidentiality to protect the key for the decryption process, the firmware or software.

Properties and Requirements

Table 3.13 depicts the non-security related requirements for implementing the use case. For this use case, the properties and requirements are derived for implementing secure boot in the automotive domain on an ECU. The different parameters for the properties are defined in Table 3.2. An essential requirement for this use case is latency, as the car is not operable before the ECU has completely booted. Of course, car manufacturers try not to strain the patience of their customers excessively here.

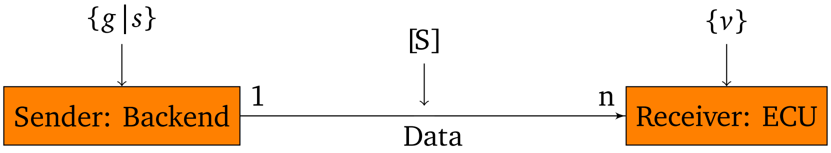

Figure 3.19: Communication model of the use case secure boot.

Properties	Backend	ECU
Latency	Very low	Very low
# Executions over product lifetime	limited	unlimited
Size of processed data	≤1KB: >1GB	≤1KB: >1GB
Physical accessibility	restricted	accessible
Computational power	Server	Embedded
RAM availability	Server	Embedded
Storage availability	Server	Embedded
# Key pairs	≤10	≤10
Data dissipation	one-to-many	
Life time	>5 years	
Current security level	≤192 bits	

Table 3.13: Requirements of the use case secure boot.

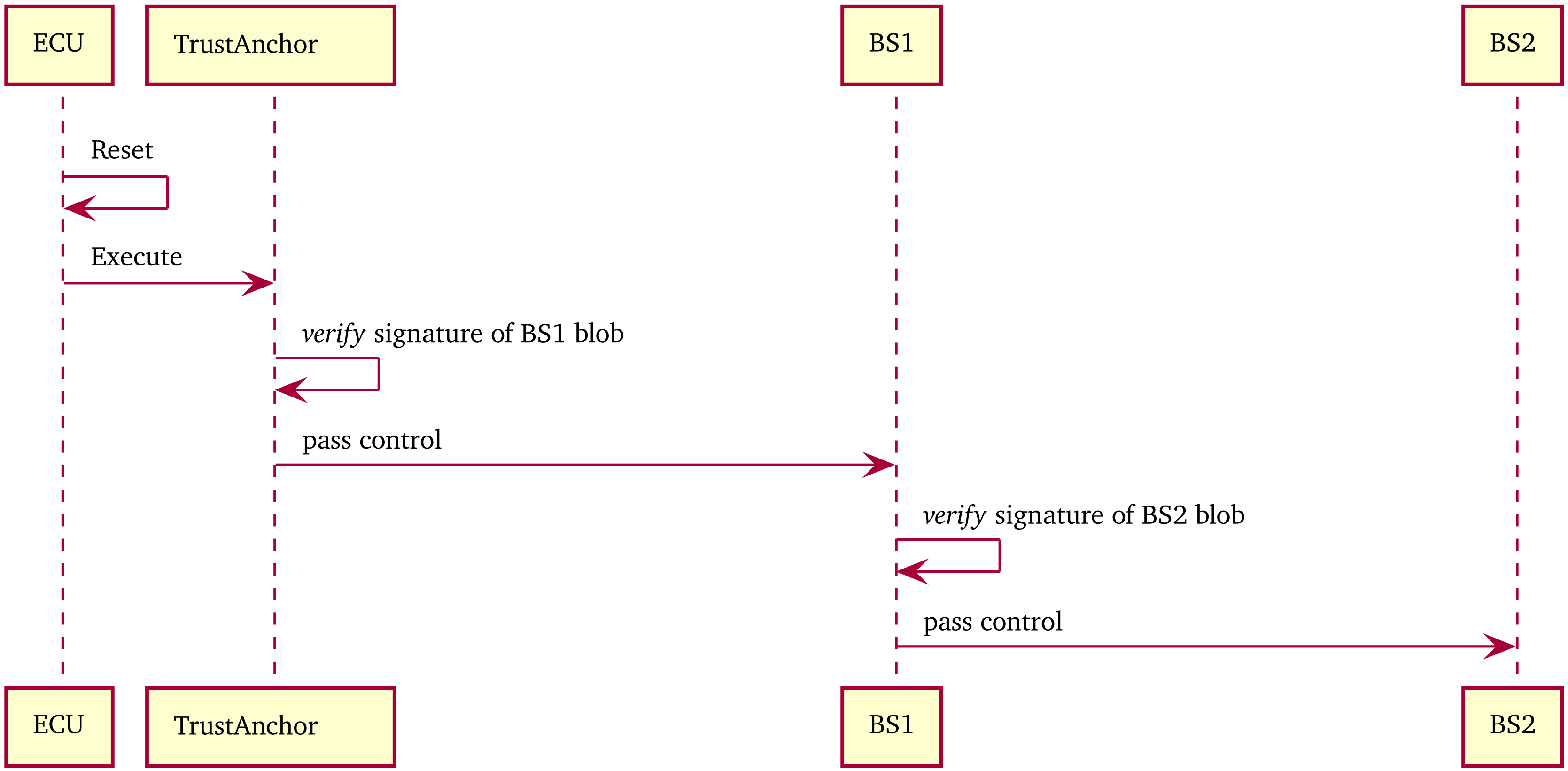

Figure 3.20: Sequence diagram of the secure boot use case.

3.4.2 Manipulation Detection

Manipulation detection extends secure boot to the files of an operating system. This use case targets the runtime detection of offline manipulation of files in a filesystem. For each integrity protected file, a metafile is generated, which contains attributes about the file. One of these attributes is a hash-based message authentication code (HMAC) of the protected file. Before each read access during runtime, the MAC tag is computed over the protected file and compared against the reference tag in the corresponding metafile. If the tag does not match, manipulation can be concluded, and the read access will be prohibited. At each write access to the protected file at runtime, the MAC tag is generated newly for the modified file and updated in the metafile. The symmetric HMAC key is generated locally on the device, is stored in a secure storage, and only accessible during runtime in a trusted environment. Prominent open-source examples of the manipulation detection on filesystems are DM-Integrity and the Extended Verification Module of the Integrity Measurement Architecture (IMA) of Linux [BPM18; KH17].

Figure 3.22 visualizes an exemplary write and read operation with enabled manipulation detection as a sequence diagram. To facilitate the integration of the manipulation detection, a virtual driver handles the MAC generation and verification, which is transparent to the application and represented as ‘transparent manipulation detection layer (TMDL)’. The driver obtains the symmetric key from a trusted environment to generate and verify the MAC tag and is never available to the application.

Considering the performance and capabilities of embedded systems, this use case only regards the HMAC based implementation of manipulation detection. The IMA provides signature-based verification of the file integrity as an alternate option to the MAC verification. This approach allows signing files outside of the system to prohibit changes of those within the system [KH17]. Regarding the SRM, this alternate approach is similar to the SRM representation of secure boot in Section 3.4.1.

Threat Model

Figure 3.21 visualizes the SRM of the use case manipulation detection. The MAC tags are generated and verified locally on the system, and the key never leaves the system. Therefore, the sender and the receiver both represent the same device. The sender represents the process of generating the MAC tag to protect the integrity and authenticity of the files.

The receiver represents the process of the MAC verification to detect violations of the integrity of the files. Since both procedures require the secret symmetric key, both entities derive the property of confidentiality.

Properties and Requirements

Table 3.14 depicts the non-security related requirements for implementing the use case. For this use case, the properties and requirements are derived for implementing manipulation detection on an embedded system. The different parameters for the properties are defined in Table 3.2.

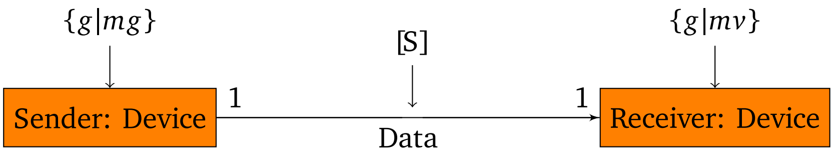

Figure 3.21: Communication model of the use case manipulation detection.

Properties	**Device**	**Device**
Latency	Very low	Very low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤1KB: >1GB	≤1KB: >1GB
Physical accessibility	accessible	accessible
Computational power	Embedded	Embedded
RAM availability	Embedded	Embedded
Storage availability	Embedded	Embedded
# Key pairs	≤10	≤10
Data dissipation	one-to-one	
Life time	≤1 months	
Current security level	≤192 bits	

Table 3.14: Requirements of the use case manipulation detection.

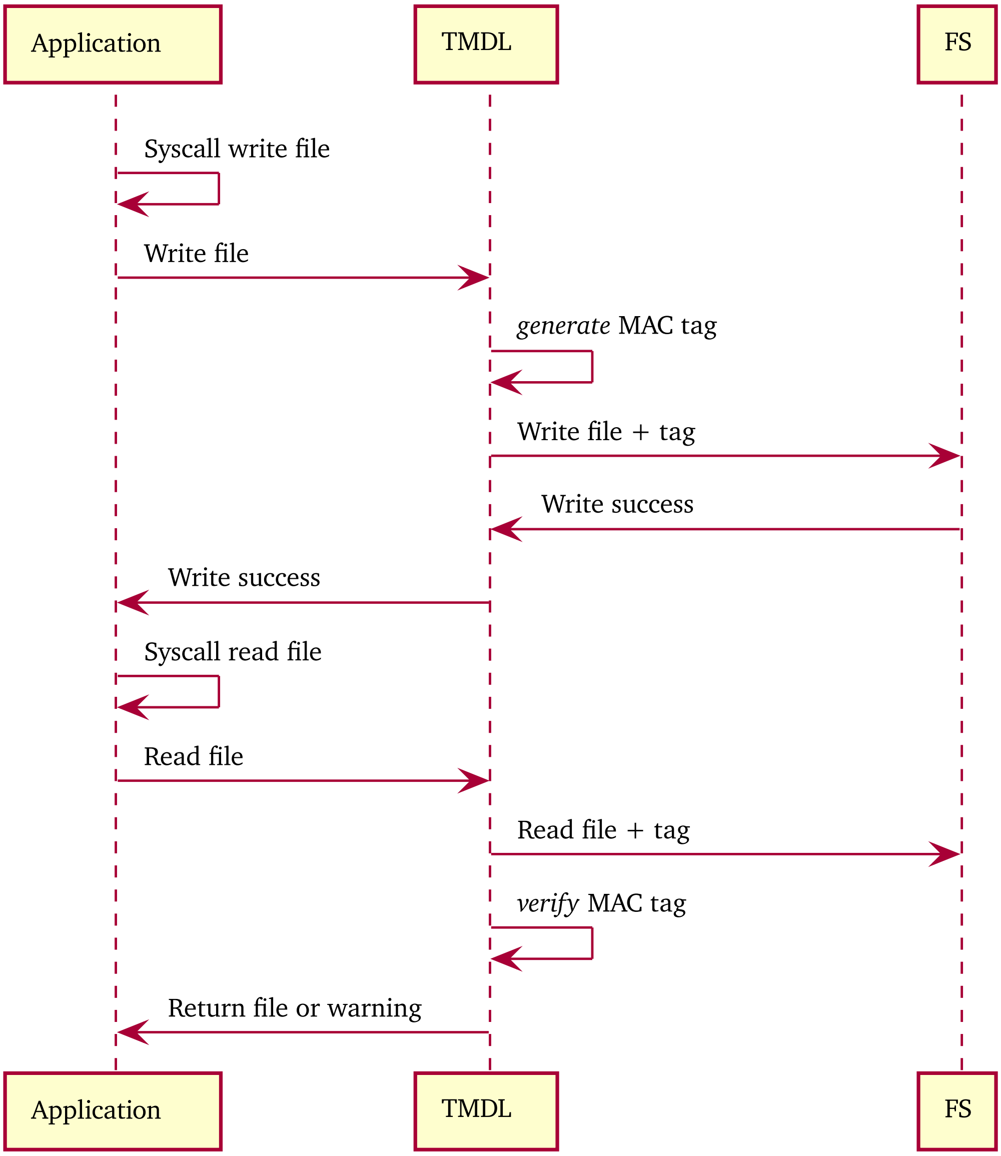

Figure 3.22: Sequence diagram of the use case manipulation detection.

3.5 Use Cases: Challenge Response Authentication

This use case cluster focuses on challenge response methods based on a verify-prover-model using freshness to prevent replay attacks. The verification of the authentication can be established by three different cryptography schemes:

1. signature-based schemes,
2. MAC-based schemes, or
3. encryption-based schemes.

In the abstract sender receiver model (cf. Figure 3.23) of the use cases, the sender always possesses a private secret key to authenticate towards the receiver, which knows the corresponding key for verifying the sender's authenticity. In case of a signature-based scheme or a public encryption scheme the public key needs to be rolled out to the receiver prior to the use case.

For symmetric schemes (MAC-based or symmetric encryption based schemes) the sender and receiver need to hold a pre-shared secret key. Usually in case of hard resource constraints symmetric schemes are favored, if the loss in security strength is acceptable. However, for this case both entities, the sender and the receiver, need additional measures to maintain integrity and prevent information disclosure with respect to the secret key material. The freshness needed for a robust challenge response scheme is typically achieved on protocol level, e.g., by adding a nonce, rather than by cryptographic means. All use cases have at least many-to-one relation between multiple senders and one receiver, or multiple senders which can authenticate towards multiple receivers.

The main differences between the use cases are the available computational power and memory of the entities.

Table 3.15 shows the ranges between the worst case and the best case of the requirements of all sub use cases.

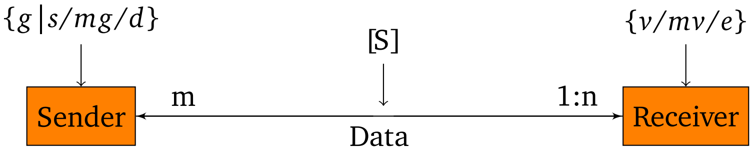

Figure 3.23: Communication model of the use case challenge response authentication using signatures.

Properties	**Sender**	**Receiver**
Latency	Low : Medium	Low
# Executions over product lifetime	Unlimited	Unlimited
Size of processed data	≤32B: <1MB	≤32B: <1MB
Physical accessibility	resctицted	accessible
Computational power	Embedded : Server	Deep. Embedded : Server
RAM availability	Embedded : Server	Deep. Embedded : Server
Storage availability	Embedded : Server	Deep. Embedded : Server
# Key pairs	≤10: ≤100.000	≤10 : ≤100
Data dissipation	one-to-many, many-to-many	
Life time	>5 years	
Current security level	≤80 bits : ≤256 bits	

Table 3.15: Requirement ranges of the use cases challenge response authentication.

3.5.1 Secure Access Control

The receiver represents an automotive ECU that restricts access to certain services and data of the ECU such as flashing. Only authorized entities shall be able to unlock these operations to access the data/services. Typical entities are manufacturers in field returns or car service stations to enroll (updated) firmware.

There are multiple ways to implement such a authorization check via a challenge response scheme:

- signature-based verification scheme
- MAC-based verification scheme
- public key encryption scheme
- symmetric encryption scheme

The signature-based scheme with a private and public key pair works as follows: A public-private key pair is generated in a secure environment (e.g., the backend) and the public key is deployed to the receivers (e.g., during production). If an accessing sender entity wants to unlock the receiver, it sends an "Unlock Request" to the receiver. The receiver generates a random seed with enough entropy and a secure bit-length and then sends it as "challenge" to the sender. The seed shall ensure the freshness to protect the unlock against replay attacks. The sender can proof the possession of the valid private key by signing the challenge seed with the private key, which corresponds to the public key stored in the receiver. The sender sends the generated signature as "response" back to the receiver, where the signature gets verified. The receiver verifies the signature with the public key against the original seed. If these match, the receiver can grant the sender access to the services/data. The first alternative is to exchange the signature scheme with a MAC verification scheme in terms of a MAC generation on sender side and MAC verification on receiver side instead of the respective signature operations. The overall flow of the protocol is in general identical to the signature-based scheme.

In case an encryption operation is used for the authorization verification in the challenge response protocol, the overall flow is identical with exception of the specific cryptographic operation changes. Hence, the challenge generated by the receiver (ECU) is a encrypted random number. The correct response from the sender to this challenge is the correct decrypted random number. This random number is than send in plain back to the receiver for verification. Also, in this setting the public key operations are performed on the ECU with a public key, while the private key operations a performed on a trusted tool. In case a symmetric encryption based scheme is used in the challenge response scheme the secret keys needs to be pre-shared between sender (tool) and receiver (ECU). The general challenge response protocol with abstract process steps is depicted in Figure 3.25.

Threat Model

The threat model of the use case secure access control leverages symmetric or asymmetric cryptographic algorithms, as shown in Figure 3.24, to ensure that only authorized entity is gaining access.

Additional security operation like certificate generation and verification might be required for some implementations of this use case. This is for instance the case, if a remote diagnostic access is requested, where a backend is connecting to the ECU as a tool via internet or other infrastructure based long distance communication channel.

Properties and Requirements

Table 3.16 depicts the non-security related requirements for implementing the use case. For this use case the properties and requirements are derived for implementing secure unlock in the automotive domain on an ECU. In case symmetric cryptographic algorithms are used for verification of the authorization for access control the number of key pairs are growing, because device unique key pairs are required. Hence, the key pairs on the ECUs stay low, but are unique, while on tool side multiple keys might be stored for generation of the correct response.

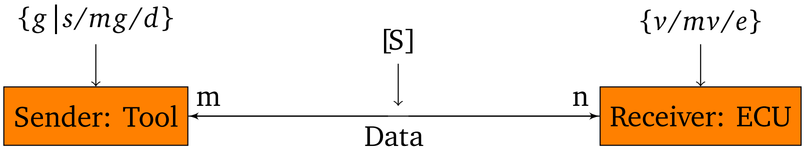

Figure 3.24: Communication model of the use case secure access control.

Properties	Tool	ECU
Latency	Low	Low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤32B: ≤1MB	≤32B: ≤1MB
Physical accessibility	restricted	accessible
Computational power	Server	Deep. Embedded : Embedded
RAM availability	Server	Deep. Embedded : Embedded
Storage availability	Server	Deep. Embedded : Embedded
# Key pairs	≤10 : ≤1.000.000	≤10 : ≤1.000.000
Data dissipation	many-to-many	
Life time	>5 years	
Current security level	≤80 bits : ≤128 bits	

Table 3.16: Requirements of the use case secure access control.

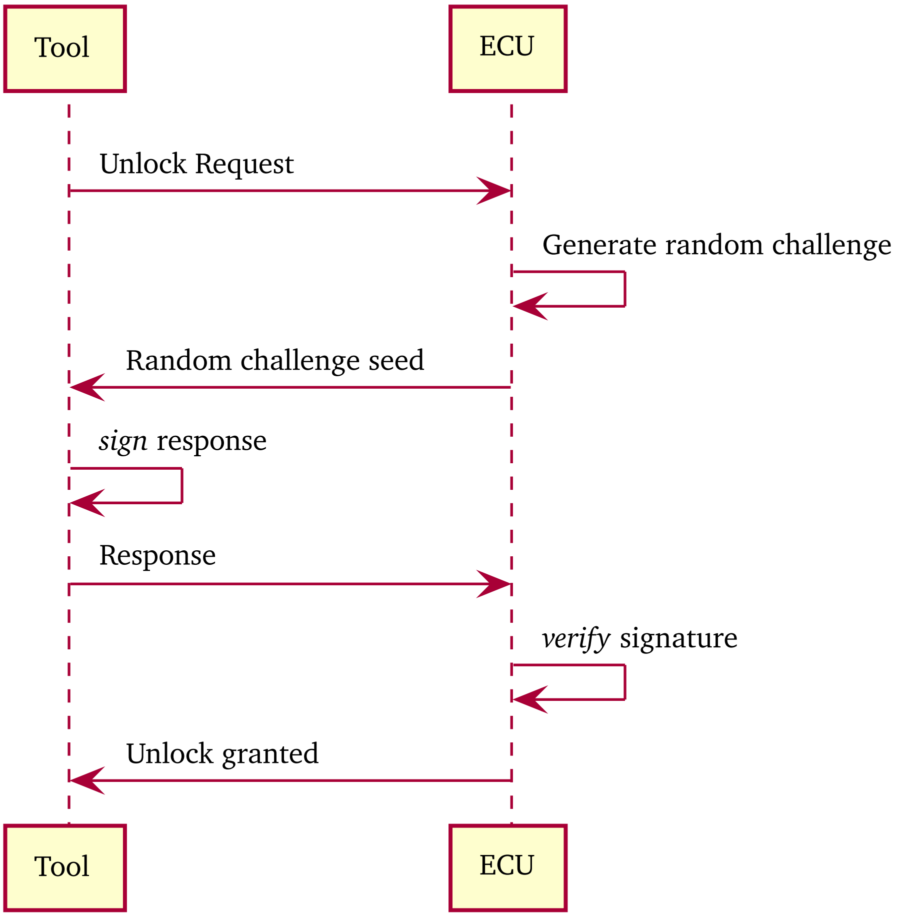

Figure 3.25: Sequence diagram of the secure unlock use case.

3.5.2 (Remote) Attestation

In this use case, a remote verifier wants to ensure that a specific application of a target platform is in the intended state and detects possible manipulations. The target platform is denoted as prover, which attests the integrity of the system to the remote verifier (cf. Figure 3.26).

The verifier requests an attestation from the prover and transmits nonce as freshness value along with the request. After receiving the request, the prover platform computes a hash over the target application to measure its integrity. Then, the hash is signed with the platform's private key along with the nonce. The outputted signature is usually encapsulated in a certificate and transmitted to the verifier. The verifier validates the signature and compares the resulting hash against the expected hash from a local database for valid application states.

In a typical setup e.g., from the trusted computing group, the hash computation over the application and the signature generation is executed in a trusted environment such as a trusted platform module (TPM), cf. [Bar06a].

Threat Model

The threat model included in the SRM for the use case (remote) attestation is depicted in Figure 3.26. The primary security goal of this use case is the authenticity of the sender (the prover). Therefore, the verifier requested an attestation response together with freshness values to avoid replay attacks. Sometimes, the entire history of the different states of the prover is incorporated in the sign response to the prover. This is needed to ensure that the system's authenticity is valid from the initiation of the system of the prover to the current measurement of the response. A pre-condition this use case is that the public key of the prover is securely exchanged with a verifier because the verifier needs the public key of the prover to verify its response, either a signed message or a signed certificate deeding on the implementation of the scheme. The verifier can request the state of multiple provers concurrently and, thus, has an m-1 relation to the prover.

Properties and Requirements

Table 3.17 depicts the non-security related requirements for implementing the use case.

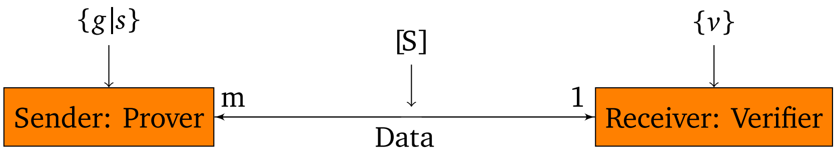

Figure 3.26: Communication model of the use case (remote) attestation.

Properties	**Prover**	**Verifier**
Latency	Medium	Low
# Executions over product lifetime	Unlimited	Unlimited
Size of processed data	≤64B: ≤1MB	≤64B: ≤1MB
Physical accessibility	untrusted	trusted
Computational power	Embedded	Server
RAM availability	Embedded	Server
Storage availability	Embedded	Server
# Key pairs	≤10	≤100
Data dissipation	many-to-one	
Life time	>5 years	
Current security level	≤256 bits	

Table 3.17: Requirements of the use case (remote) attestation.

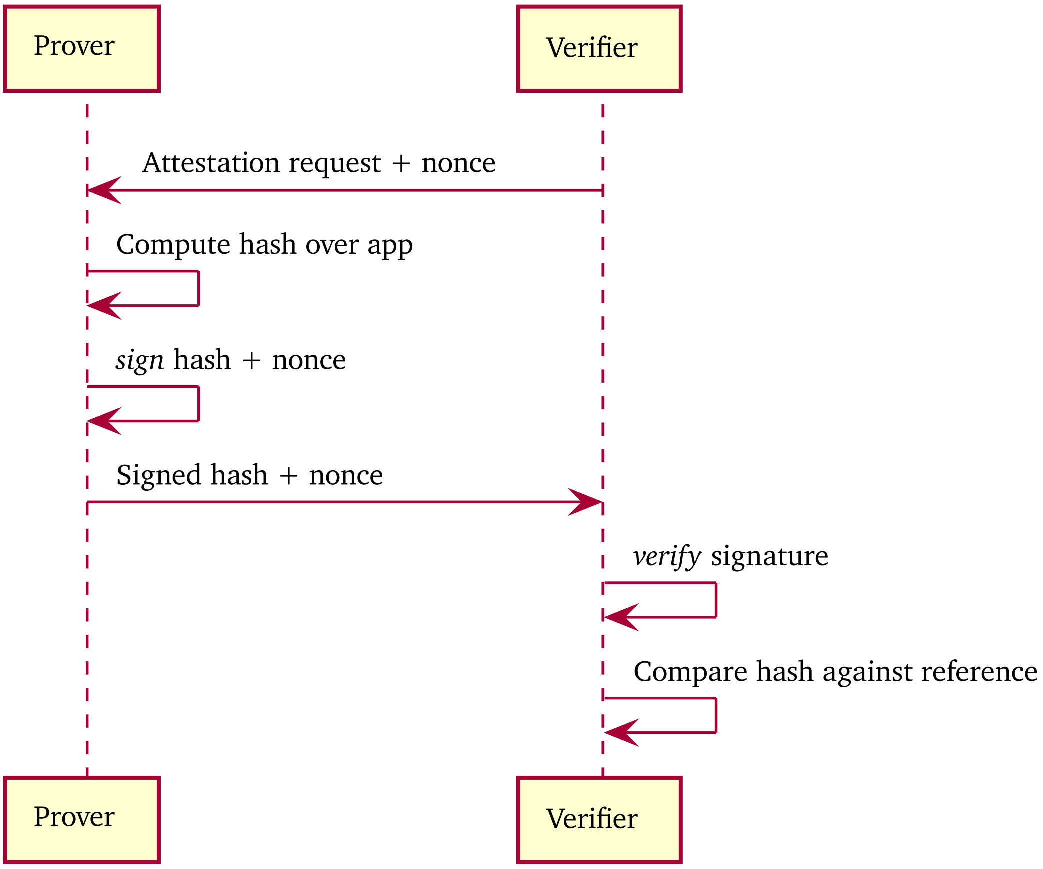

Figure 3.27: Sequence diagram of the (remote) attestation use case.

3.6 Use Cases: Signature Freshness

The use cases within this cluster all share the necessity of having a fresh signature. Thus, it must be ensured that the transmitted data is both authenticated *and* fresh. The corresponding communication model is shown in Figure 3.28.

In general, one party generates a signature or a MAC together with a freshness value, e.g. a counter, a timestamp, or a nonce and the other party verifies the fresh authenticity token. Figure 3.28 represents the example where the sender generates the signature or MAC and the receiver verifies the token. This scheme is applied for the use cases secure time distribution (Section 3.6.1), vehicle-to-vehicle communication (Section 3.6.3) and sensor data transmission (Section 3.6.4). For the use case status request OCSP (Section 3.6.2), the cryptographic operations are switched, i.e. the sender verifies the signed feedback message from the receiver.

Authenticity is usually achieved by the verification of an asymmetric signature. For this, the data is signed by the sender with her private key and verified by the receiver using the sender's public key. The easiest way to achieve data or signature freshness is letting the signer sign a random nonce (beforehand chosen by the verifier) *together with* the payload. If the random nonce is sufficiently long, no adversary can forward older data without breaking the signature scheme. Another way is to sign a timestamp together with the payload. This approach enables the verifier to check the freshness directly. This however, requires synchronized clocks. Finally, a third way to ensure freshness is to sign a sequence number together with the payload. In this case, both signer and verifier need to agree on a starting sequence number via a secure channel and hold their respective current sequence number as a counter. Upon reception, the verifier can check whether the sequence number has been increased and is within a reasonable range to verify that no old data has been replayed.

Table 3.18 shows the ranges between the worst case and the best case of the requirements of all use cases in this cluster.

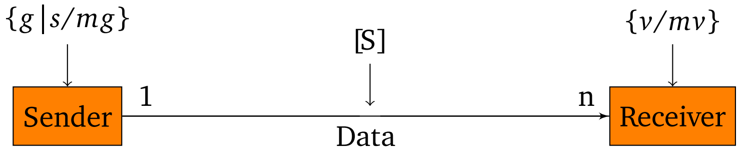

Figure 3.28: Communication model of the use cases signature freshness.

Properties	**Sender**	**Receiver**
Latency	Very very low : Low	Very very low : Low
# Executions over product lifetime	limited : unlimited	limited : unlimited
Size of processed data	≤1KB:≤1MB	≤1KB:≤1MB
Physical accessibility	untrusted:trusted	untrusted:trusted
Computational power	Deeply embedded:server	Industry PC:Server
RAM availability	Deeply embedded:server	Industry PC:Server
Storage availability	Deeply embedded:server	Industry PC:Server
# Key pairs	≤10:<1.000.000	≤1.000
Data dissipation	one-to-many, many-to-many, many-to-one	
Life time	≤1 day	
Current security level	≤128 bits	

Table 3.18: Requirements of the use case cluster signature freshness.

3.6.1 Secure Time Distribution

Any device that requires an accurate time needs to synchronize occasionally. By using a signature, the data authenticity and integrity of the distributed time can be ensured. When requesting synchronization, the End-Entity (EE) appends a random nonce. The Backend's signature is computed over the nonce and the payload using the private key in a trusted environment. The corresponding public key, which is used for verification, is stored inside the EE in a tamper-protected storage. Only if the signature verification is successful and data freshness is ensured by checking the nonce, the synchronization data is used.

The whole process is visualized in Figure 3.30. Note that to prevent an attacker delaying the signed synchronization data, the EE must check that the backend responds within an appropriately short time.

Threat Model

When mapping this use case to the SRM in Figure 3.29, the sender is the backend or a trusted time server and the receiver is the EE that requires time synchronization. A secure time distribution can be crucial to protocols whose security relies on synchronized clocks. Additionally, denial of service or forcing of faults (underflows, etc.) could be achieved with a tampered time. To do so, an attacker can

- spoof the backend's identity to send her own data or
- tamper the backend's data or
- tamper the data sent to the receiver or
- tamper the EE's data reception.

Additionally, the attacker could replay old data even if the protocol is secured against the above four threats. Thus, freshness needs to be ensured as well. The backend must be able to perform re-keying and can revoke keys – both to maintain long term security. The receiving EE needs to verify the backend's certificate.

Properties and Requirements

Table 3.19 shows the requirements for an implementation of this use case. Note that the receiving EE usually could also work with only one key pair. However, if more than one trusted time server shall be available, one key pair must be stored for each of them.

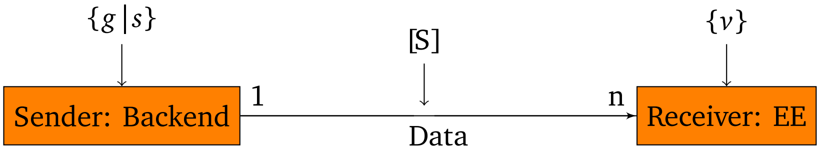

Figure 3.29: Communication model of the use case secure time distribution.

Properties	**Backend**	**EE**
Latency	Very low	Very low
# Executions over product lifetime	limited	limited
Size of processed data	<1KB	<1KB
Physical accessibility	trusted	untrusted
Computational power	Server	Industry PC
RAM availability	Server	Industry PC
Storage availability	Server	Industry PC
# Key pairs	≤10	≤1.000
Data dissipation	one-to-many	
Life time	≤1 day	
Current security level	≤128 bits	

Table 3.19: Requirements of the use case secure time distribution.

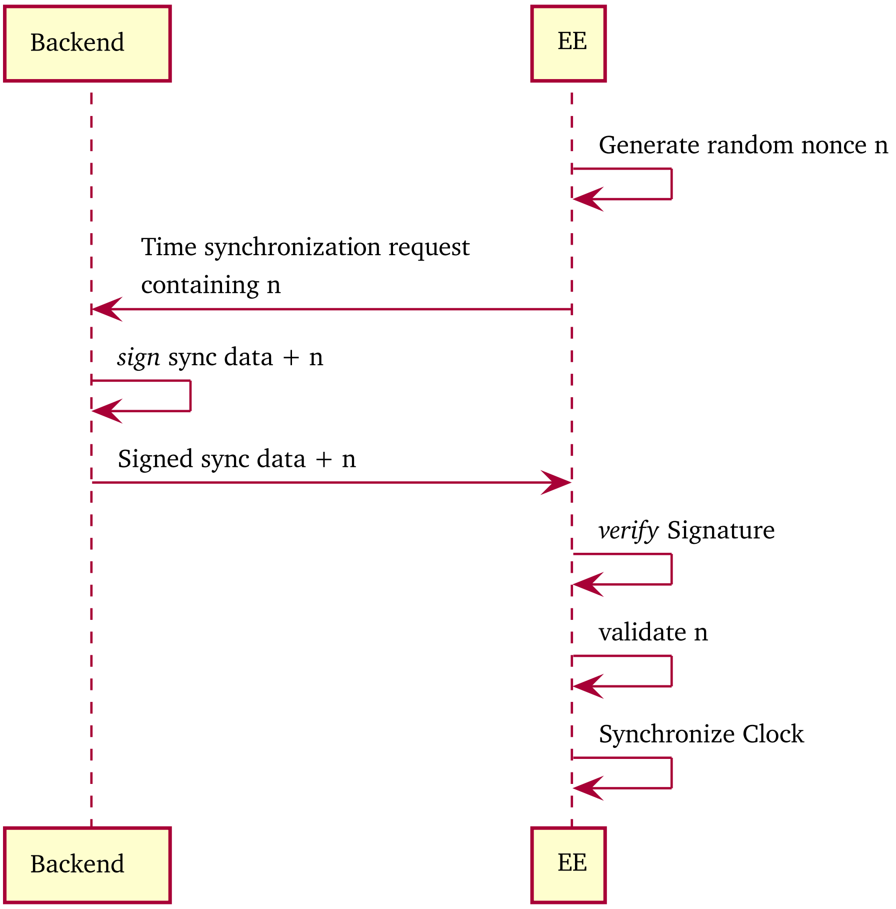

Figure 3.30: Sequence diagram of the use case secure time distribution.

3.6.2 Status Request OCSP

Before using a certificate, entities must validate whether the respective certificate is revoked or not. One method to perform this is to query the CA for a signed response about the status of this certificate. A protocol that supports querying the CA about the status of the certificate is OCSP [RFC6960]. One possible value for the status of the certificate is *good*, which indicates that the certificate is not revoked. Another value is *revoked*, which indicates that this certificate is revoked and thus not valid.

Threat Model

The communication model of this use case is depicted in Figure 3.31. Note that it differs slightly from Figure 3.28 as the sender also verifies. The authenticity and freshness of the response on a request from the CA to the end-entity need to be protected. Hence, a signature scheme together with a nonce as freshness on protocol level hinders replay attacks and forged responses from a malicious entity or imitation of the certification authority (CA) as a communication partner. Furthermore, the CA needs the ability to generate short term certificates to document the validity of other certificates. An additional access control scheme is sometimes put in place before processing every request to filter unauthorized requests.

Properties and Requirements

Table 3.20 shows the non-security related requirements for implementing the use case.

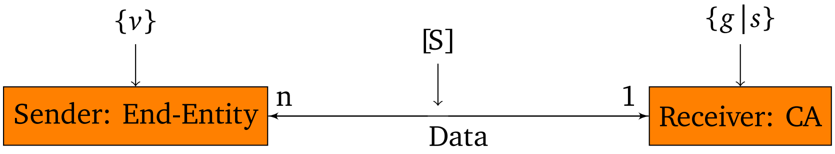

Figure 3.31: Communication model of the use case status request OSCP.

Properties	**End-Entity**	**CA**
Latency	Low	Low
# Executions over product lifetime	unlimited	limited
Size of processed data	≤1MB	≤1MB
Physical accessibility	untrusted	trusted
Computational power	IT	Server
RAM availability	IT	Server
Storage availability	IT	Server
# Key pairs	>1.000.000	>1.000.000
Data dissipation	many-to-one	
Life time	≤1 day	
Current security level	≤128 bits	

Table 3.20: Requirements of the use case status request OSCP.

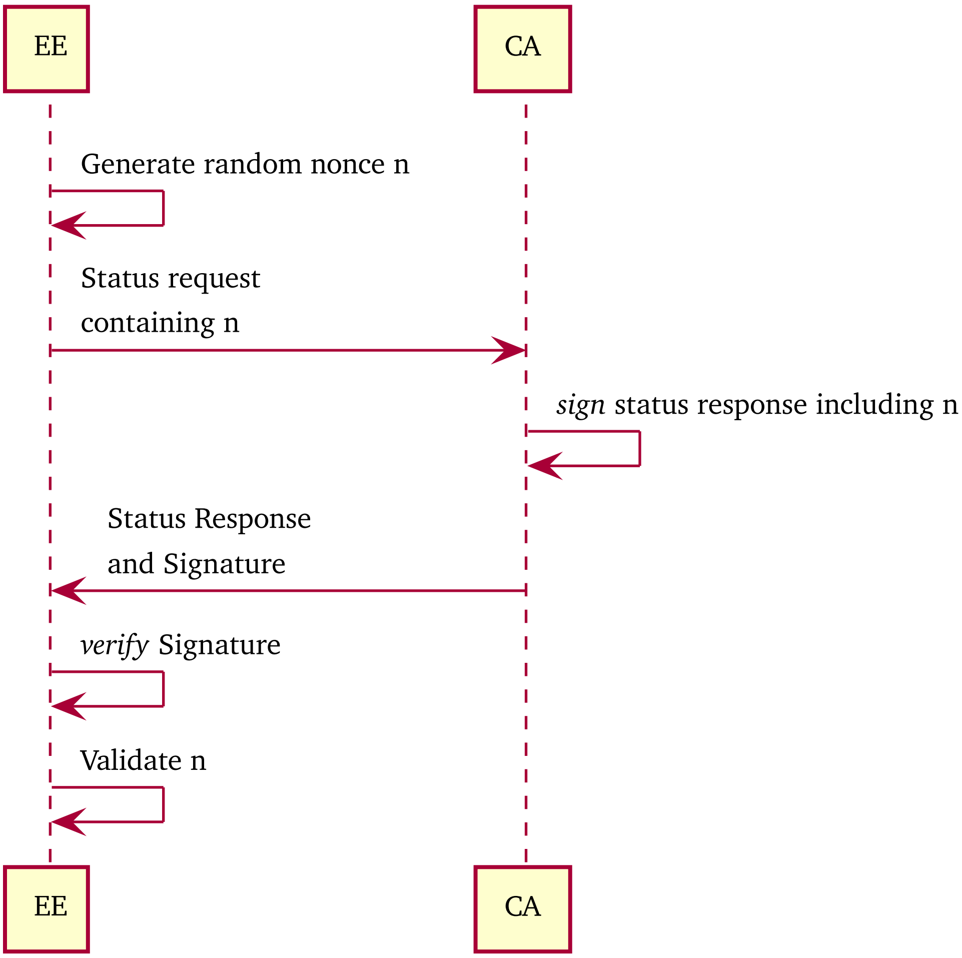

Figure 3.32: Sequence diagram of the use case status request OCSP.

3.6.3 Vehicle-to-Vehicle Communication

Vehicle-to-vehicle communication is an upcoming technology that is supposed to increase traffic safety. To achieve this, vehicles shall collect data such as surrounding conditions (weather, road conditions, traffic density), warnings (accidents) and vehicle-inherent data (e.g. position and velocity) to share them via a wireless channel to other vehicles. The receiving entities must be ensured that the data is authentic and fresh.

Figure 3.34 shows the use case. To ensure the freshness, a timestamp is added to the payload, which requires precise and at least a roughly synchronized time in all participating vehicles.

Threat Model

For this use-case, the sender in Figure 3.33 is the vehicle that shares data, and the receivers are the vehicles that receive the data. In this scenario, it is essential that the vehicle-to-vehicle communication cannot be tampered. Attackers could cause severe traffic problems and even risk the passenger's health and life if it would be possible for them to inject their malicious data (e.g. wrong accident warnings).

To do so, an attacker can

- spoof a vehicle's identity to send her own data or
- pretend to be a vehicle even though she is not or
- tamper a vehicle's data or
- tamper the data sent via the wireless channel or
- tamper a vehicle's data reception.

Additionally, the attacker could replay old data even if the protocol is secured against the above threats. Thus, freshness needs to be ensured as well. Depending on the computational power of the sensor, different cartographic authentication schemes might be used to ensure low latency. The sending vehicle (Vehicle A) must be able to generate keys, compute signatures, revoke keys, and re-keying on top of sending authentic messages to maintain its status as a valid member in the certificate and trust chain of the public key infrastructure. Vehicle B (the receiving vehicle) needs to be able to verify the signatures of the received messages.

Properties and Requirements

Table 3.21 shows the requirements for an implementation of this use case. It is important to stress that even though the data dissipation of this use case is many-to-many, a vehicle always signs data once but needs to verify data many times. Thus, the receiver latency is given with *very very low*, but the sender's latency may be only *very low*. If the sent data is not crucial to safety, the latency requirements may be chosen less strictly.

The processed data usually should be below 1KB. However, if images shall be shared (e.g. for special traffic signs), the data size could grow to multiple 100KB.

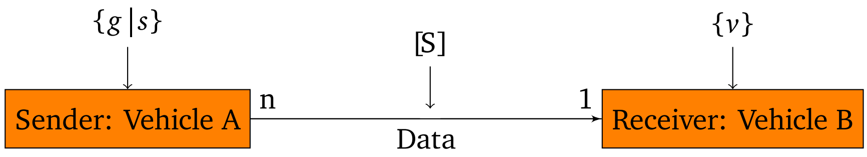

Figure 3.33: Communication model of the use case vehicle-to-vehicle communication.

Properties	**Vehicle A**	**Vehicle B**
Latency	Very low	Very very low
# Executions over product lifetime	Unlimited	Unlimited
Size of processed data	≤1MB	≤1MB
Physical accessibility	untrusted	untrusted
Computational power	industry PC	
RAM availability	industry PC	
Storage availability	industry PC	
# Key pairs	≤10	≤1.000
Data dissipation	many-to-many	
Life time	≤1 day	
Current security level	≤128 bits	

Table 3.21: Requirements of the use case vehicle-to-vehicle communication.

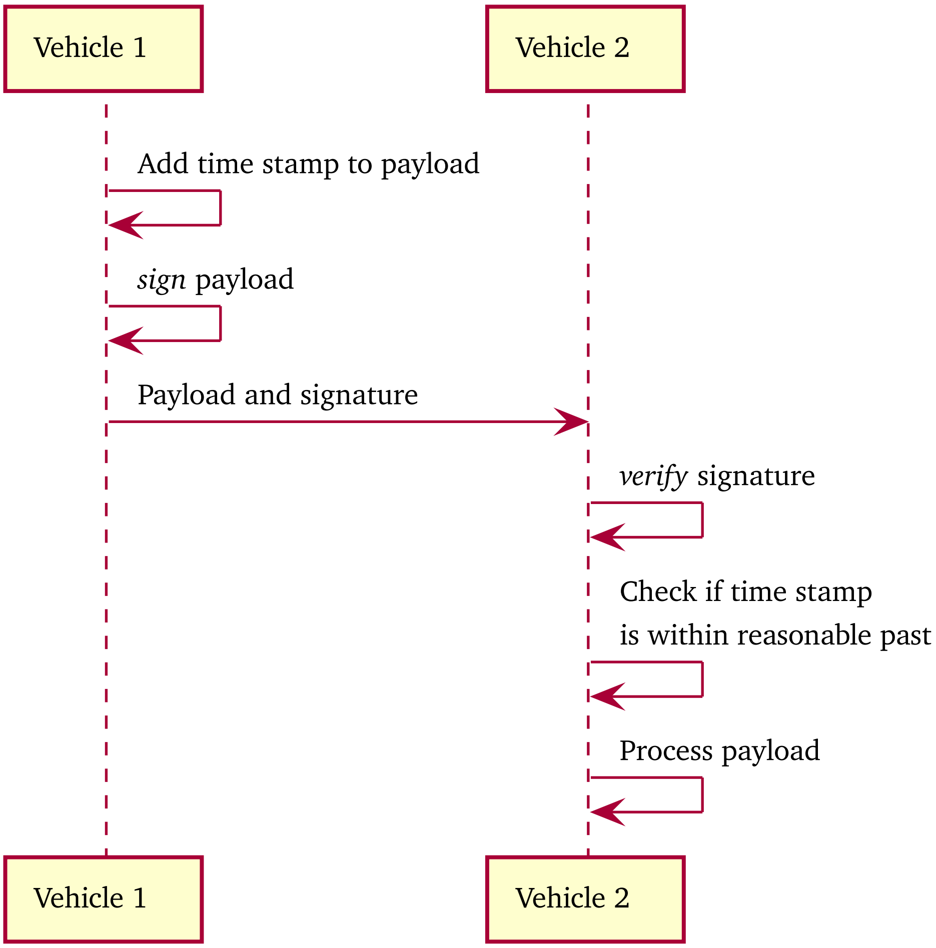

Figure 3.34: Sequence diagram of the use case vehicle-to-vehicle communication.

3.6.4 Sensor Data Transmission

Sensor data is what a system perceives of its surroundings or its state. Thus, it is crucial to secure the data in any scenario where an attacker can alter them maliciously. For this, a scheme to generate an authentication token - either a signature or a MAC over the payload - is applied. The sensor generates the token over its data before transmitting it to the ECU. Also, data freshness must be ensured in some cases.

Figure 3.36 depicts the use case. A sequence number is added to the sensor's payload data. After verifying the signature, the ECU checks whether the sequence number is reasonably higher than the last received one to hinder replay attacks.

Threat Model

In Figure 3.28, the sender is the sensor and the receiver is the host ECU. As already pointed out, sensor data is a system's perception, and thus, tampering this data may significantly influence the results or actions the system produces. For example, a tampered distance measurement of a car's radar may let this car initiate an emergency braking, which poses a safety threat to traffic participants. As depicted in Figure 3.35, an attacker can

- spoof a sensor's identity to send her own data or
- tamper a sensor's data or
- tamper the transmitted data or
- tamper the ECU's data reception.

Additionally, the attacker could replay old data even if the protocol is secured against the above threats. For example, if it would be possible to inject old distance measurements that contain a short result (from a low-velocity situation) into a situation where the vehicle moves with high velocity, the exact above scenario of an emergency braking could be provoked. Thus, data freshness needs to be ensured as well.

Depending on the computational power of the sensor, different cryptographic authentication schemes might be used to ensure very low latency. Besides that, this use case has no additional requirements for any additional security operation, because it just provides a basic security measure to ensure authenticity with minimal latency.

Properties and Requirements

Table 3.22 shows the requirements for an implementation of this use case. Note that the size of processed data mainly depends on the sensor type and could be even larger than given here for the case that cameras are included in this use case as well.

Also, the latency requirement may be less strict for systems that are not relevant for safety. However, it is essential to note that often, one ECU receives data from multiple sensors, which leads to a restriction of the ECU's latency.

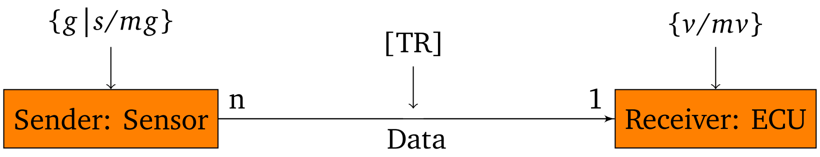

Figure 3.35: Communication model of the use case sensor data transmission.

Properties	**Sensor**	**ECU**
Latency	Very very low	Very very low
# Executions over product lifetime	Unlimited	Unlimited
Size of processed data	<1KB	<1KB
Physical accessibility	untrusted	untrusted
Computational power	Deeply embedded	Embedded
RAM availability	Deeply embedded	Embedded
Storage availability	Deeply embedded	Embedded
# Key pairs	≤10	≤1.000
Data dissipation	many-to-one	
Life time	≤1 day	
Current security level	≤128 bits	

Table 3.22: Requirements of the use case sensor data transmission.

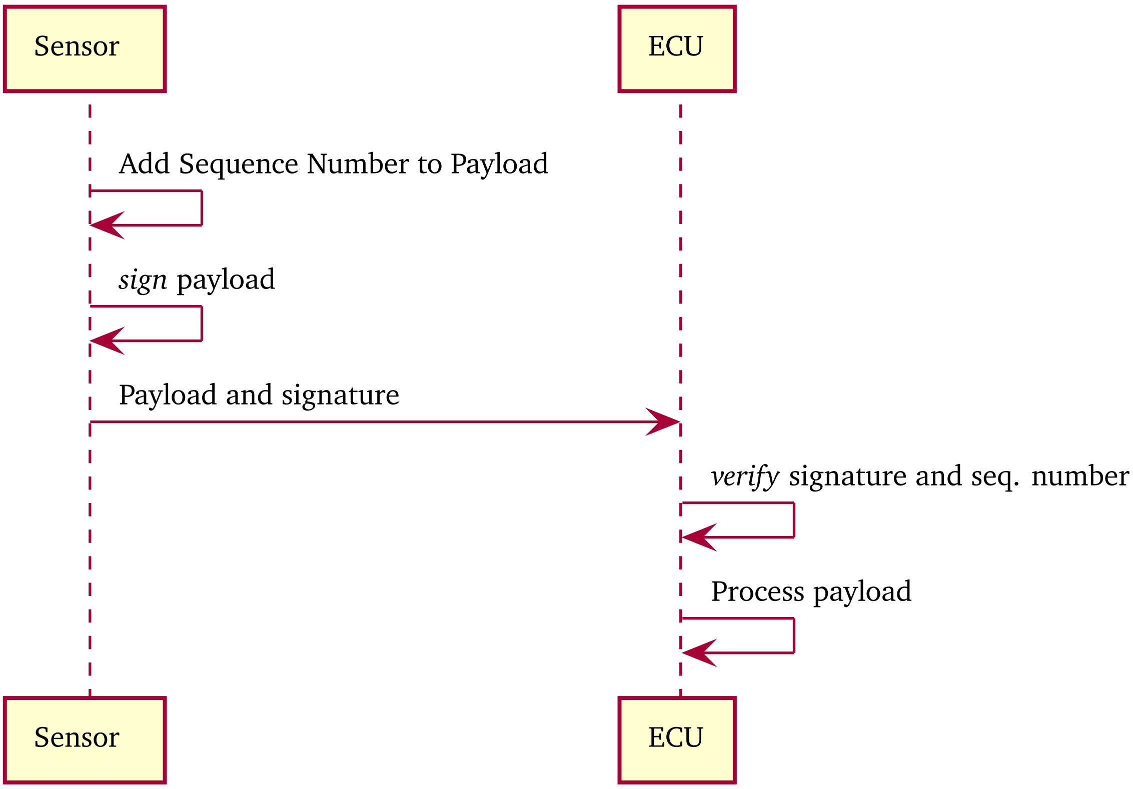

Figure 3.36: Sequence diagram of the use case sensor data transmission.

3.7 Use Cases: Key Pair Generation

This cluster of use cases considers the generation of key pairs both for resource-constrained devices as well as server systems. The public key is, in most cases, certified by a backend or a CA. In the context of PQC, this is an interesting and challenging family of use cases because the involved cryptographic operation is mostly affected by quantum computers. All use cases consider operations like key pair generation and signature calculation when issuing certificates, secure delivery of key material is essential, and a large number of devices is affected.

In Section 3.7.1 the key pair is generated on a device, and the CA is providing a certificate to integrate the public key of the device into the trust chain of the PKI. In this use case, the sender needs to perform a key generation and a signature verification for each generated key pair. In Section 3.7.2, we describe the use case of generating the key pair at the PKI side. In this second use case, the sender (CA in the backend) performs a key generation and signature generation.

Table 3.23 shows the ranges between the worst case and the best case of the requirements of all sub-use cases.

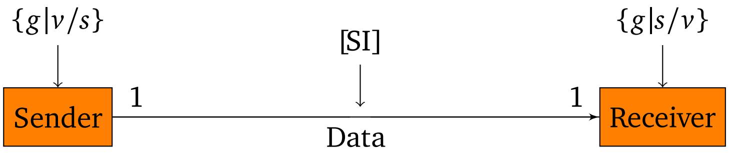

Figure 3.37: Communication model of the use case key generation on device.

Properties	Sender	Receiver
Latency	Very Low	Very Low
# Executions over product lifetime	limited	limited
Size of processed data	≤16KB	≤16KB
Physical accessibility	untrusted:trusted	untrusted:trusted
Computational power	Embedded:Server	Deep. emb.:Server
RAM availability	Embedded:Server	Deep. emb.:Server
Storage availability	Embedded:Server	Deep. emb.:Server
# Key pairs	≤10: >1.000.000	>1.000.000: ≤10
Data dissipation	one-to-one	
Life time	>5 years	
Current security level	≤128 bits	

Table 3.23: Requirement ranges of the key pair generation use cases.

3.7.1 Key Generation on Device

Some devices require one or maybe multiple device unique keys. The private key of the device's unique private and public key pair shall not leave the device due to security policies. However, the public key of this pair needs to be integrated into the chain of trust of the corresponding public key infrastructure. Therefore, during production the public key needs to be signed by the corresponding private key of the CA. In some cases, the public key needs to be integrated into the trust chain of the corresponding CA, when the device is already deployed in the field.

The CA may check the quality of the public key by trying to find weaknesses before signing it. Usually, this is done by performing some tests about the mathematical properties of the key, see [Bar06b]. If the quality of the key is insufficient, the CA does not integrate the public key into the trust chain and thus does not present a certificate to the requesting device. Also, the CA might check if the requesting device is in possession of the private key corresponding to the public key that has been requested to get signed. This procedure is called proof-of-possession (PoP). Such a proof requires that the device performs an operation using the private key. The CA can verify the result of this operation. PoP is usually performed by signing the request with the private key. The CA validates this signature, and if the signature is correct, then it is assured of the existence and possession of the private key by the EE. If signing is not possible, then the CA either creates an encrypted challenge that the end-entity needs to decrypt by using the private key, or it encrypts the certificate. The CA does not issue a certificate if it fails to verify the possession of the corresponding private key by the device.

Threat Model

The primary threat is that the public key of the device is not signed with the intended and corresponding private key of the CA. Therefore, it needs to be ensured that the one-to-one communication is correctly set up. During production, this can be ensured due to a physical connection and a trusted operation environment. For the case of a re-keying in the field, an additional secure communication channel needs to be established to ensure the authenticity of the sending and receiving data between the device and the backend. Therefore, a secure session-based secure channel can be leveraged, as explained in Section 3.8.1.

Properties and Requirements

Table 3.24 depicts the non-security related requirements for implementing the use case. For this use case, the properties and requirements are derived for implementing key generation on device in the automotive domain on an ECU.

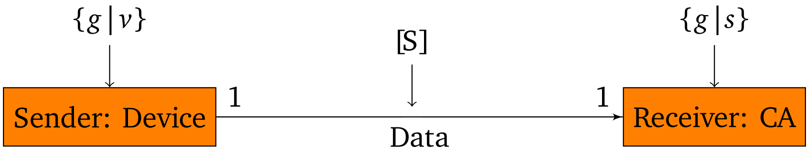

Figure 3.38: Communication model of the use case key generation on device.

Properties	**Device**	**CA**
Latency	Very Low	Very Low
# Executions over product lifetime	limited	limited
Size of processed data	≤16KB	≤16KB
Physical accessibility	untrusted	trusted
Computational power	Embedded	Server
RAM availability	Embedded	Server
Storage availability	Embedded	Server
# Key pairs	≤10	>1.000.000
Data dissipation	one-to-one	
Life time	>5 years	
Current security level	≤128 bits	

Table 3.24: Requirements of the use case key generation on device.

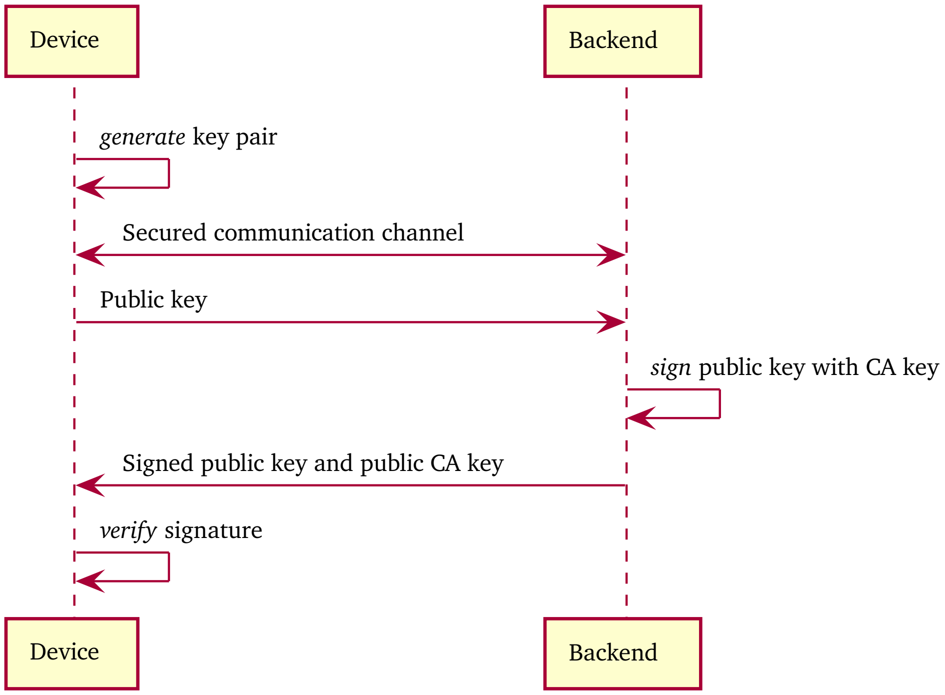
Device
Backend
generate key pair
Secured communication channel
Public key
sign public key with CA key
Signed public key and public CA key
verify signature
Device
Backend

Figure 3.39: Sequence diagram of the key generation on device use case.

3.7.2 Key Generation with PKI and Deployed on Device

This use case is very similar to the *key generation on device* use case. The only difference is the sender and receiver instances. The direction of the unidirectional secure transport channel just changed, and the instances just swapped. The backend (usually a CA) becomes the sender, and the device becomes the receiver. In this use case, the private and public key is generated in the backend of the CA and transferred securely to the device by leveraging an additional secured transportation channel. This use case also covers the generation of secret keys for the device in the backend. Also, the generated secret keys in the backend need to be transferred securely by an additional secured transportation channel to the device. The security properties of the secure transport channel are authenticity, integrity, and confidentiality. These properties are essential to protect the secret and private keys or other confidential credentials.

Next to the keys (symmetric and asymmetric) generated for the device also the public key of the signing CA is transferred. The public key of the CA is required to verify the public key of the device that can be embedded in a certificate. This certificate is then provisioned by the CA. The process steps of this use case are depicted in Figure 3.41.

Threat Model

The threat model of this use case has an additional threat compared to the previous threat model of this use case in this cluster. Since the private and public key pair is generated outside the device, the private key needs to be transferred from the CA to the device in a confidential manner. Therefore, the sender and receiver model has an additional security goal to the transferred data compared to the previous one, as depicted in Figure 3.40.

The additional security operation for certificate generation and verification is required for the current use case. In some cases the public key for the device is embedded in a certificate.

Properties and Requirements

Table 3.25 depicts the non-security related requirements for implementing the use case. For this use case, the properties and requirements are derived for implementing key generation on the server-side and deployment on the device.

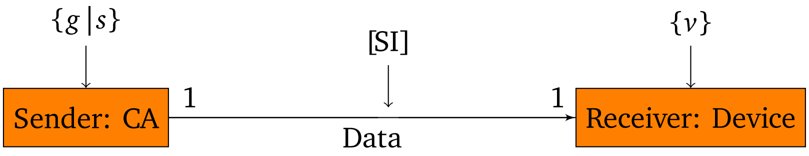

Figure 3.40: Communication model of the use case key generation with PKI and deployed on device.

Properties	**CA**	**Device**
Latency	Very Low	Very Low
# Executions over product lifetime	limited	limited
Size of processed data	≤1MB	≤1MB
Physical accessibility	trusted	untrusted
Computational power	Server	Deeply embedded
RAM availability	Server	Deeply embedded
Storage availability	Server	Deeply embedded
# Key pairs	>1.000.000	≤10
Data dissipation	one-to-one	
Life time	>5 years	
Current security level	≤128 bits	

Table 3.25: Requirements of the use case key generation with PKI and deployed on device.

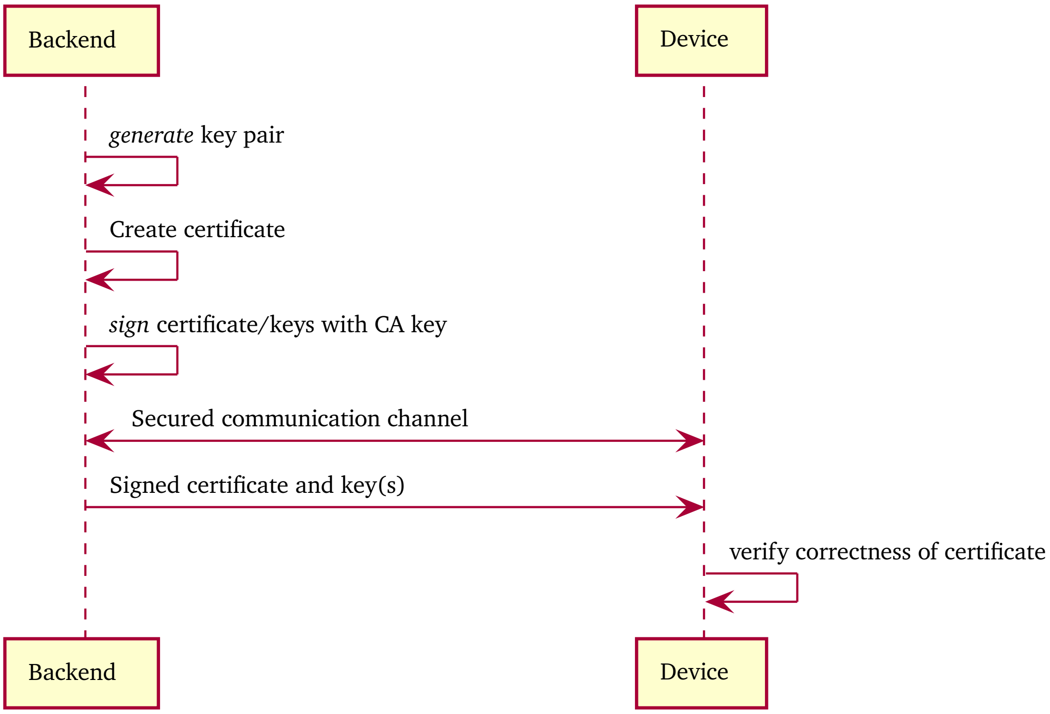
Backend
Device
generate key pair
Create certificate
sign certificate/keys with CA key
Secured communication channel
Signed certificate and key(s)
verify correctness of certificate
Backend
Device

Figure 3.41: Sequence diagram of the key generation with PKI and deployed on device use case.

3.8 Use Cases: Secure Channel Establishment

This section clusters use cases that enable the establishment of a secure channel. They provide an authentic and confidential communication channel between two parties. In general, a secure channel is required for multiple data exchange scenarios, e.g., to collect status information, maintain remote devices, or transfer privacy-related data. In contrast to the use cases described in Section 3.3 this cluster covers logical one-to-one connections. The connections are managed as sessions so that an established channel sustains for some time and messages are continuously transmitted bidirectionally. Whereas a secure channel is established between two parties there is typically a single server that handles connections to many clients as shown by Figure 3.42. Thus, the server in the backend requires appropriate resources. The two use cases Session-based Secure Channel and Password-authenticated Key Establishment belong to this use case cluster. The first use case discusses a secure channel establishment and is detailed in Section 3.8.1. The second use case is a scheme for access control and secure channel establishment by exploiting a key-exchange scheme for an authenticated and encrypted channel, see Section 3.8.2.

Both parties must be authentic and the exchanged data has to be confidential, authentic, and fresh. So the parties mutually authenticate using digital signatures as well as certificates or passwords. To ensure freshness during the exchange of data, the nonce is introduced on the protocol level. The data is authenticated by symmetric MACs or implicitly using authenticated encryption. The confidentiality of the exchanged data is realized by symmetric encryption. The symmetric encryption uses a shared session key which is initially established using an asymmetric key encapsulation or key exchange mechanism.

Table 3.26 shows the ranges between the worst case and the best case of the requirements of all sub use cases. In all sub use cases, the clients (initiator) has the computational properties of a deeply embedded to an IT device and runs in a insecure operation environment, while the back-end systems or stations run in an insecure or controlled field with varying computation properties between embedded and server systems. The latency on both sides varies between very very low to very low. The number of executions over the product live time is higher than 1024, and an unpredictable amount of processed data per message varies between several bytes to more than 1 GB. The lifetime validity of the cryptographic artifacts of the use cases are less than one year to more than five years.

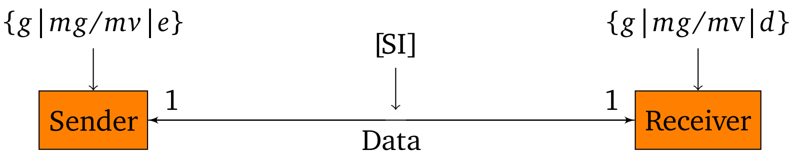

Figure 3.42: Communication model of the cluster Secure Channel Establishment.

Properties	**Sender**	**Receiver**
Latency	very very low:very low	very very low:very low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤64B:>1 GB	≤64B:>1 GB
Physical Accessibility	restricted	accessible
Computational power	embedded:Server	deeply embedded:IT
RAM availability	embedded:Server	deeply embedded:IT
Storage availability	embedded:Server	deeply embedded:IT
# Key pairs	≤10:≤1000	≤10:≤1000
Data dissipation	one-to-one	
Life time	≤1 year:>5 years	
Current security level	*:≤128 bits	

Table 3.26: Requirements of the use case Secure Channel Establishment.
*Depends on the password length.

3.8.1 Session-based Secure Channel

Maintaining and controlling many devices that are connected to a backend system over an IP-based local, wide-area, or mobile network requires a secure channel to ensure confidential and authentic communication. Typical use cases are, for instance, the management of a car or truck fleet, the gathering and aggregation of sensor data, or the remote control of a power plant. The needed services that provide these functionalities are well scalable using IP-based communication protocols.

Transport Layer Security (TLS) enables the protection of the message payload. The protocol specification provides mutual certification-based authentication and session-based secure communication. If both the payload of messages and the metadata must be protected against misuse a Virtual Private Network (VPN) should be used. Several clients connect to a central VPN-gateway. Each connection is established individually but the gateway can route network packages so that a protected network is build up. There are multiple variants of VPN on different ISO/OSI levels, e.g., it could be implemented using Transport Layer Security (TLS).

Threat Model

The utilized physical connection is an untrusted communication medium on which transferred data could be eavesdropped, forged, injected, or manipulated by intermediate nodes.

In general, both parties must be authentic, and the exchanged data has to be confidential, authentic, and fresh. So the parties mutually authenticate using digital signatures as well as certificates or passwords. The data is authenticated by symmetric MACs or implicitly using authenticated encryption. The confidentiality of the exchanged data is realized by symmetric encryption. The symmetric encryption uses a shared session key, which is initially established using an asymmetric key encapsulation or key exchange mechanism. For each session, a new nonce is generated, or a counter is incremented to establish freshness on protocol level. The communication model including the cryptographic operation are shown in Figure 3.43 and the sequence diagram in Figure 3.44. In most cases, the backend side is more trusted than the client-side, hence the backend side is most often controlling the communication channel. Therefore, the backend able to trigger re-keying procedures or revoke keys or certificates.

Properties and Requirements

Table 3.27 depicts the non-security-related requirements for implementing the use case. For this use case, the properties and requirements are derived for implementing a session-based secure channel with a VPN or TLS in the scenarios described above.

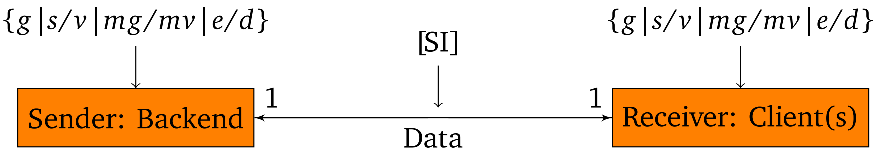
$\{g\,|\,s/v\,|\,mg/mv\,|\,e/d\}$
[SI]
$\{g\,|\,s/v\,|\,mg/mv\,|\,e/d\}$
Sender: Backend
1
1
Receiver: Client(s)
Data

Figure 3.43: Communication model of the use case Session-based Secure Channel.

Properties	**Backend**	**Client(s)**
Latency	very very low:very low	very very low:very low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤64B:>1 GB	≤64B:>1 GB
Physical accessibility	untrusted:trusted	untrusted
Computational power	Server	embedded:IT
RAM availability	Server	embedded:IT
Storage availability	Server	embedded:IT
# Key pairs	≤1000	≤1000
Data dissipation	one-to-one	
Life time	≤1 year	
Current security level	≤128 bits	

Table 3.27: Requirements of the use case Session-based Secure Channel.

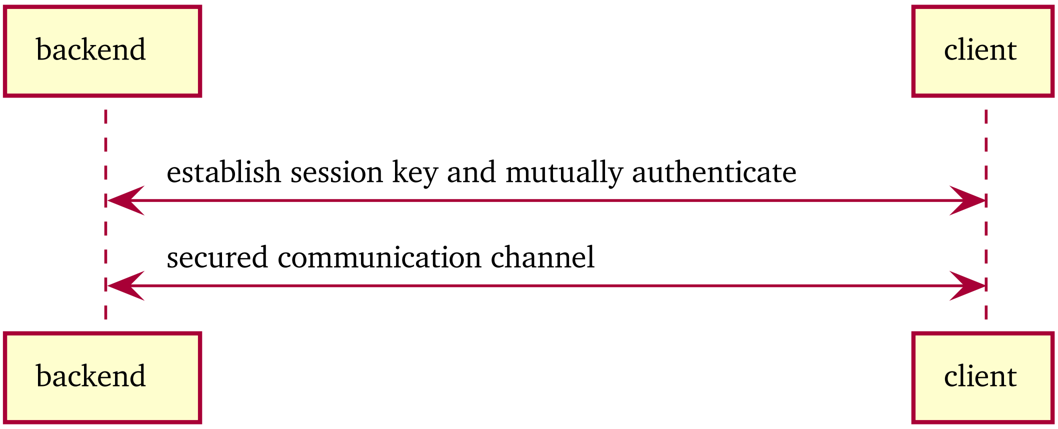
backend
client
establish session key and mutually authenticate
secured communication channel
backend
client

Figure 3.44: Sequence diagram of the Session-based Secure Channel use case.

3.8.2 Password-authenticated Key Establishment

Small Bluetooth connected gadgets or modern chip-based passports, for instance, are embedded or even deeply embedded devices with very tight constraints and do not have an extensive or any user interface at all. However, these devices often require a secure channel to communicate with their counterparts or even serve as an additional security factor in multi-factor authentication to establish a secure channel Password-authenticated Key Establishment (PAKE) protocols provide a high entropy key derivation from a low entropy input, e.g., a password or a Personal Identification Number (PIN). These protocols can be used for session key derivation for embedded devices. A famous protocol of this family is Password Authenticated Connection Establishment (PACE) which is applied in machine readable travel documents, for instance. It uses a mapping function to derive Diffie-Hellman group parameters from a password, PIN, or machine readable data, for agreeing on a session key by the execution of the Diffie-Hellman scheme. The communication model including the cryptographic operations is shown in Figure 3.45. The exemplary use case flow depicted in Figure 3.46 shows how a client would initiate a key exchange derived from a password. Note that an important constraint and prerequisite is that terminals need to have access to databases maintaining the user passwords/secrets in order to decrypt the random number.

Threat Model

The utilized physical connection is an untrusted communication medium on which transferred data could be eavesdropped, forged, injected, manipulated, or replayed by intermediate nodes. Both parties share a secret or password, respectively. This password must not be disclosed or tampered. In particular, PACE uses a MAC, symmetric encryption, a cryptographic hash function, and the Diffie-Hellman scheme (in Elliptic Curve groups) and nonces for freshness.

Properties and Requirements

Table 3.27 depicts the non-security-related requirements for implementing the use case. For this use case, the properties and requirements are derived for implementing a key session agreement using a PAKE protocol.

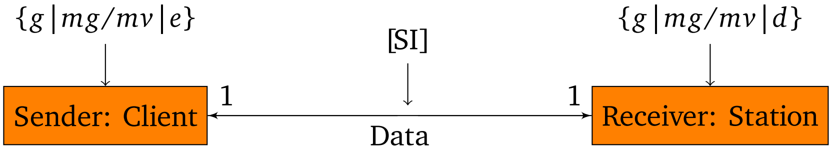

Figure 3.45: Communication model of the use case Password-authenticated Key Establishment.

Properties	Client	Station
Latency	very very low:very low	very very low:very low
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤1KB:≤1MB	≤1KB:≤1MB
Physical accessibility	untrusted	untrusted
Computational power	deeply embedded:embedded	embedded:IT
RAM availability	deeply embedded:embedded	embedded:IT
Storage availability	deeply embedded:embedded	embedded:IT
# Key pairs	≤10	≤10
Data dissipation	one-to-one	
Life time	>5 year	
Current security level	Depends on the password length	

Table 3.28: Requirements of the use case Password-authenticated Key Establishment.

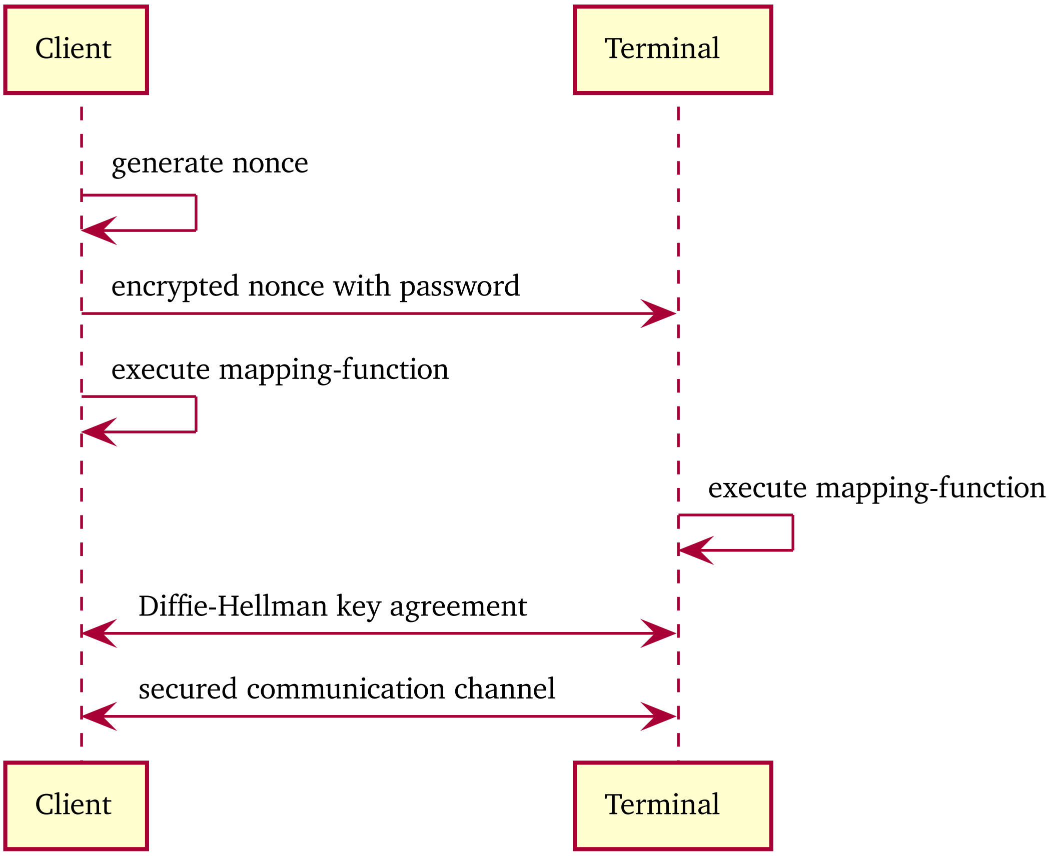

Figure 3.46: Sequence diagram of the Password-authenticated Connection Establishment use case.

3.9 Use Cases: Distributed Data Access

Due to the importance and value of all kinds of data, government agencies and other stakeholders (e.g. insurance companies) will want to have access to it. However, this is not always desirable for the user. Instead, the owner of a vehicle wants to retain data sovereignty, although several government agencies could still jointly access this information. For this reason, the issues of a distributed and consensual access to such data must be urgently considered. It must be regulated which parties are granted access, i.e. data should only be accessible if several participants decide so together. If a consensus is achieved, at least two parties can gain access to the data. This is impossible for one party alone. See Figure 3.47 and Figure 3.48 how to generate the share. The procedure to regain the secret is given by Figure 3.49 and Figure 3.50. In this use case the trusted party is a ECU inside the car and the paricipants are the stakeholder having interests in gaining access to the critical data inside the car.

Unfortunately, such a use case is not implemented at the moment in the automotive industry. However, the use case and its utilization of secret sharing is from an academic perspective useful and might apply to the to use cases shortly described in Section 3.9.1 and Section 3.9.2. This new concept could solve some implementation issues of the use cases in Section 3.9.1 and Section 3.9.2 concerning the architecture level and protocol level. This secret sharing concept might become beneficial if the architecture in which the use case is implemented does not contain secure storage.

The security goals for the secret sharing approach are hard to link to the traditional security goals that are related to cryptographic primitives. This requirement is derived mainly from the protocol level, where the first step of the secret sharing approach is implemented in general.

Requirements

See Table 3.29 for the non-security related requirements needed for implementing this use case. It should be noted that access to protected data does not have to be particularly fast. However, the data must be protected reliably over an unusually long period, as it may contain critical information.

Properties	Other Party	Trusted Party
Latency	Low	Low
# Executions over product lifetime	limited	limited
Size of processed data	≤1KB	≤1KB
Physical Accessibility	accessible	restricted
Computational power	Embedded	Embedded
RAM availability	Embedded	Embedded
Storage availability	Embedded	Embedded
# Key pairs	≤100	≤100
Data dissipation	one-to-one	
Life time	>10 years	
Current security level	≤128 : ≤256 bits	

Table 3.29: Requirements of the use case delegated access.

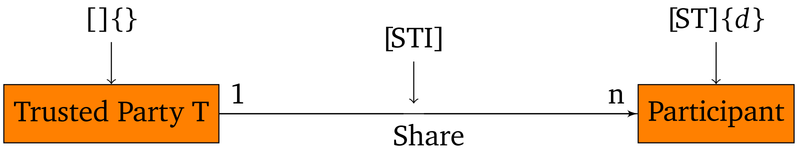

Figure 3.47: Communication model to generate and distribute shares to n participants.

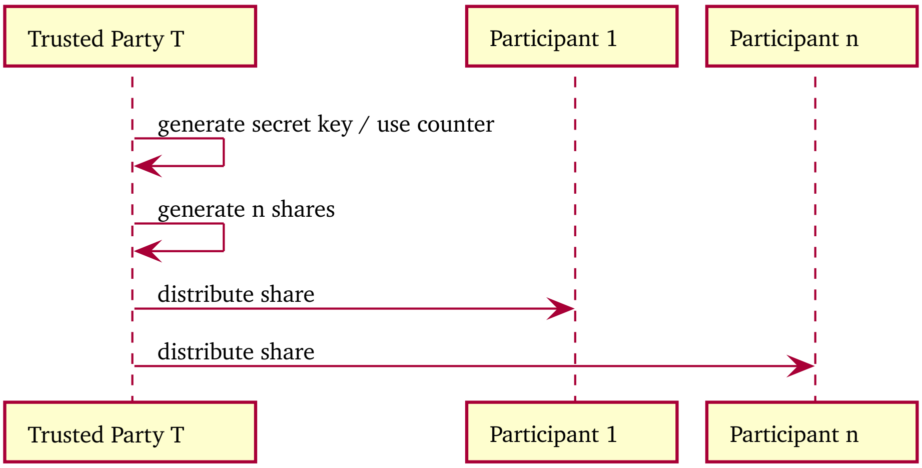

Figure 3.48: Generating and distributing shares among n participants.

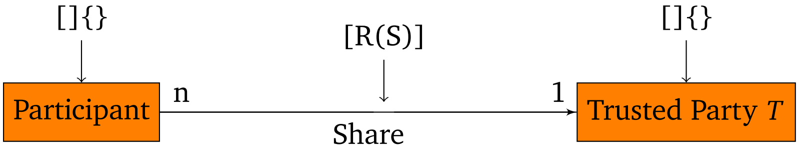

Figure 3.49: Communication model for regaining a secret by collecting n shares.

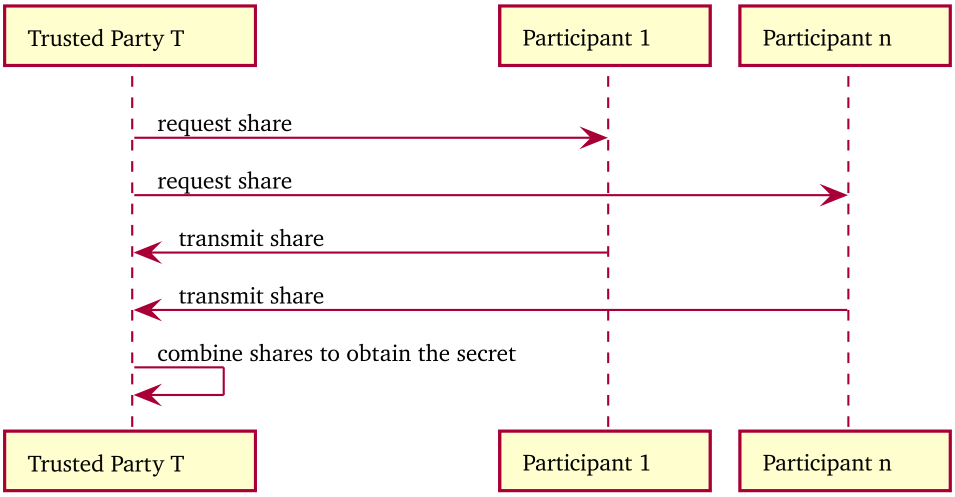

Figure 3.50: Combining shares to regenerate the secret.

3.9.1 Delegated Access Control

A black box is a device that records all relevant data during a crash. Possibly, in the event of an accident, the owner of a vehicle may not want to provide this crash information to other parties (e.g. the lawyer of an opponent). But if at least two participants of the members of the group manufacturer, insurance, police, and the owner itself work together, the secret data can be revealed. Another example of data with special protection needs is motion profiles. Modern cars have activated permanent GPS tracking, which results in large amounts of location data worth protecting. Hence, the secret itself or a key providing access to the secret has to be divided into several parts.

3.9.2 Secure Logging

Surprisingly, this use-case occurs relatively frequently and has interesting applications. Therefore, it can be used for e.g., the construction of tachometers, operating hours counters, or for the safe storage of a number of performed production steps. From the security perspective, it is irrelevant what type of resource is counted, and therefore counter-values can reliably be stored. In this use-case, we assume that no secure storage is available. Otherwise, we would have a trivalent solution.

The basic idea is based on the following fact. If a counter reading is stored distributed over several devices, a potential attacker must carry out several attacks, which makes the attack more expensive and, with appropriate planning, economically unattractive. Only if a sufficient number of these make their shares available can the secret be reconstructed by the trusted party.

Furthermore, devices with better protection can get several shares (and thus gain in importance), and simple ECUs manage only a few shares. By such a measure, the costs of a possible attack can be finely balanced without significantly increasing the hardware costs (tamper-resistant storage).

4 D1.2 - Requirements on Post Quantum Schemes

In this chapter, we list selection criteria for the selection of proper cryptographic algorithms for the use cases that are listed in Chapter 3. The selection criteria aim to demonstrate the implications and considerations of replacing the current algorithms with PQC algorithms in realistic use cases (from an industrial view). All the criteria listed in this section represent a first rough approximation, which have to be refined later. Although they are certainly not detailed enough to evaluate the cryptographic algorithms reliably, they can be used in this work package to structure and evaluate the use cases systematically.

4.1 Efficiency and Resources Footprint

Run-time and memory requirements are traditional key parameters for evaluating an algorithm. These resources are particularly necessary for embedded devices that have very few resources. For example, in the case of signatures, the time for generating the keys, signature generation, and verification is essential. The use cases show which of these parameters are relevant. If the verification has to be performed by a small embedded device, this step should require limited resources. Hence, the use cases should provide information about these facts as well. Similarly, the size of a private and corresponding public key, a signature or the overhead of ciphertexts should be known.

The impact of the signature and keys sizes directly translates to requirements concerning memory. Storing keys has an impact on the size of non-volatile memory, such as ROM. The dynamic memory component size, e.g., RAM, is impacted by other algorithm and implementation-specific criteria. In general, the block size of processing cipher- or plain texts, as well as the signature size and message size, have direct impact on the required dynamic memory consumption. But also the implementation of specific parameters might have an impact on non-volatile memory consumption. To reflect this impact on the memory, the use case descriptions in Chapter 3 provide a typical platform-specific memory for the application of the use case. Hence, the platform characteristics provided in Table 3.2 specify upper or lower boundaries for consideration of memory utilization.

In general, requirements on computing power (DMIPS) and latency of a controller are used to reflect the performance and execution speed of the different algorithms on the controller. Thus, Table 3.2 also contains the boundary specification of these two requirements for specific platforms. For each use case description, the typical expectation values on the latency and performance shall be provided.

The definition of the use case requirements now allows deriving more specific requirements on potential algorithms used. It is important to note here, that these numbers

Device Type	Storage	RAM
Deeply Embedded	64KB	10KB
Embedded	256KB	50KB
IT	512KB	128KB
Server	5MB	500KB

Figure 4.1: Labels for different device categories.

only represent what amounts to an estimate on potential capacities and are kept artificially low. Due to the larger resource requirements of post-quantum algorithms, it may not be possible to fulfill all of the following requirements. Nonetheless, they are useful in so far as that they give an anchor around which future investigations and improvements can focus on. From a technological point of view, capabilities also get large and more affordable with time, and as such should be able to afford larger sizes in the future.

The requirements that follow are broken down by use case. They use the categories listed in Figure 4.1 for the different device types and otherwise follow the notation from Table 3.2. Ranges of resource requirements between A and B are denoted as $A : B$.

Secure Download (Section 3.2.1)

Primitive Used		Signature		
Signature		Private Key Generation	Signing	Verification
Time (Latency)	Backend	Medium		
	ECU			Low
Storage	Backend	Server		
	ECU			Deeply Embedded
RAM	Backend	Server		
	ECU			Deeply Embedded

Feature Activation (Section 3.2.2)

Primitive Used		Signature		
Signature		Private Key Generation	Signing	Verification
Time (Latency)	Backend	Low		
	ECU			Low
Storage	Backend	Server		
	ECU			Embedded
RAM	Backend	Server		
	ECU			Embedded

Certificate Revocation List (Section 3.2.3)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time (Latency)	CA	Medium		▨
	End Entity	▨	▨	Medium
Storage	CA	Server		▨
	End Entity	▨	▨	IT
RAM	CA	Server		▨
	End Entity	▨	▨	IT

Confidential Configuration (Section 3.3.1)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	Backend	▨	▨	▨
	Client	▨	▨	▨
Storage	Backend	Embedded : Server		
	Client	Deeply Embedded : Server		
RAM	Backend	Embedded : Server		
	Client	Deeply Embedded : Server		

Primitive Used		Key Exchange	
Key Exchange		**Private Key Generation**	**Public Key Generation / Exchange**
Time	Backend	▨	▨
	Client	▨	▨
Storage	Backend	Embedded : Server	
	Client	Deeply Embedded : Server	
RAM	Backend	Embedded : Server	
	Client	Deeply Embedded : Server	

Encrypted Update (Section 3.3.2)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	Backend	▨	▨	▨
	Client	▨	▨	▨
Storage	Backend	Embedded : Server		
	Client	Deeply Embedded : Server		
RAM	Backend	Embedded : Server		
	Client	Deeply Embedded : Server		

Primitive Used		Key Exchange	
Key Exchange		Private Key Generation	Public Key Generation / Exchange
Time	Backend		
	Client		
Storage	Backend	Embedded : Server	
	Client	Deeply Embedded : Server	
RAM	Backend	Embedded : Server	
	Client	Deeply Embedded : Server	

Flashing via Onboard Diagnosis (Section 3.3.3)

Primitive Used		Signature		
Signature		Private Key Generation	Signing	Verification
Time	Tool			
	ECU			
Storage	Tool	IT		
	ECU	Deeply Embedded		
RAM	Tool	IT		
	ECU	Deeply Embedded		

Preloading Updates (Section 3.3.4)

Primitive Used		Signature		
Signature		Private Key Generation	Signing	Verification
Time	Tool	Low		
	Client			Low
Storage	Tool	IT		
	Client			Deeply Embedded : Server
RAM	Tool	IT		
	Client			Deeply Embedded : Server

Secure Boot (Section 3.4.1)

Primitive Used		Signature		
Signature		Private Key Generation	Signing	Verification
Time	Backend	Very Low		
	ECU			Very Low
Storage	Backend	Server		
	ECU			Embedded
RAM	Backend	Server		
	ECU			Embedded

Manipulation Detection (Section 3.4.2)

The use case ”Manipulation Detection” uses an HMAC as its security algorithm. As the underlying algorithm itself is symmetric, it is so far not heavily impacted by quantum computers. Nonetheless, the situation could change, and thus a summary of its constraints is presented.

Primitive Used	HMAC
Time	Very Low
Storage	Backend
RAM	Backend

Access Control

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	Backend	Low : Medium		
	Client	Low : Medium		
Storage	Backend	Server : Embedded		
	Client	Server : Deeply Embedded		
RAM	Backend	Server : Embedded		
	Client	Server : Deeply Embedded		

Primitive Used		Key Exchange	
Key Exchange		**Private Key Generation**	**Public Key Generation / Exchange**
Time	Tool	Low	
	ECU	Low	
Storage	Tool	Server : Embedded	
	ECU	Server : Deeply Embedded	
RAM	Tool	Server : Embedded	
	ECU	Server : Deeply Embedded	

Remote Attestions (Section 3.5.2)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	Prover	Medium		▨
	Verifier	▨		Low
Storage	Prover	Embedded		▨
	Verifier	▨		Server
RAM	Prover	Embedded		▨
	Verifier	▨		Server

Secure Time Distribution (Section 3.6.1)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	Backend	Very Low		▨
	EE	▨		Low
Storage	Backend	Server		▨
	EE	▨		IT
RAM	Backend	Server		▨
	EE	▨		IT

Status Request OCSP (Section 3.6.2)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	CA	Low		
	End-Entity			Low
Storage	CA	IT		
	End-Entity			IT
RAM	CA	IT		
	End-Entity			IT

Vehicle-to-Vehicle Communication (Section 3.6.3)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	Vehicle A	Very Low		
	Vehicle B			Very Low
Storage	Vehicle A	IT		
	Vehicle B			IT
RAM	Vehicle A	IT		
	Vehicle B			IT

Sensor Data Transmission (Section 3.6.4)

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	Sensor	Very Very Low		
	ECU			Very Very Low
Storage	Sensor	Deeply Embedded		
	ECU			Embedded
RAM	Sensor	Deeply Embedded		
	ECU			Embedded

Key Generation on Device (Section 3.7.1)

Primitive Used		Key Exchange	
Key Exchange		**Private Key Generation**	**Public Key Generation / Exchange**
Time	Device	Very Low	
	CA	Very Low	
Storage	Device	Embedded	
	CA	Server	
RAM	Device	Embedded	
	CA	Server	

Key Generation with PKI (Section 3.7.2)

Primitive Used		Key Exchange	
Key Exchange		**Private Key Generation**	**Public Key Generation / Exchange**
Time	Device	Very Low	
	CA	Very Low	
Storage	Device	Deeply Embedded	
	CA	Server	
RAM	Device	Deeply Embedded	
	CA	Server	

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	CA	Very Low		
	Device			Very Low
Storage	CA	Server		
	Device			Deeply Embedded
RAM	CA	Server		
	Device			Deeply Embedded

Delegated Access & Secure Logging (Section 3.9.1)

Both ”Delegated Access” and ”Secure Logging” do not define the cryptographic primitives, thus only general constraints are given:

Primitive Used	Variable
Time	Low
Storage	Embedded
RAM	Embedded

Session-based Secure Channel (Section 3.8.1)

Primitive Used		Key Exchange	
Key Exchange		**Private Key Generation**	**Public Key Generation / Exchange**
Time	Backend	Very Very Low : Very Low	
	Client	Very Very Low : Very Low	
Storage	Backend	Server	
	Client	Embedded : IT	
RAM	Backend	Server	
	Client	Embedded : IT	

Primitive Used		Signature		
Signature		**Private Key Generation**	**Signing**	**Verification**
Time	CA	Very Very Low : Very Low		
	Device			Very Very Low : Very Low
Storage	CA	Server		
	Device			Embedded : IT
RAM	CA	Server		
	Device			Embedded : IT

Password-authenticated Key Establishment (Section 3.8.2)

Primitive Used		Key Exchange
Key Exchange		**Private Key Generation** **Public Key Generation / Exchange**
Time	Backend	Very Very Low : Very Low
	Client	Very Very Low : Very Low
Storage	Backend	Deeply Embedded : Embedded
	Client	Deeply Embedded : IT
RAM	Backend	Deeply Embedded : Embedded
	Client	Deply Embedded : IT

4.2 Resilience

Cryptographic methods require the correct choice of parameters. It is essential that small improvements in cryptanalytic techniques do not have a significant impact on the security of a scheme. Therefore, it should be possible to select universal parameter sets, which can be used over a more extended period of time and which can be used for standardization. Similarly, the parameters should provide a proper security-level on quantum computers. Also, these parameter sets should be widely used for a long time to increase confidence in an algorithm.

All public-key cryptographic primitives are based on certain hard mathematical problems. For this reason, strong confidence in the underlying mathematical problem is desirable (e.g., due to the duration of the investigation by the research community, the variety of results such as NP-hardness or reductions to other well known problems).

Additionally, the number of citations in established scientific publications (impact factor should be considered, too) and active discussions in relevant forums (e.g., the PQC forum) is a good indicator of the quality of a mathematical problem. Similarly, the existence of proofs and reductions which consider attackers with quantum computers should be carefully taken into account.

The randomness used during a cryptographic operation is a central point. For this reason, the use cases have to consider this issue carefully. It must be examined whether the influence of weak randomness on security and effectiveness is known for the use cases and underlying cryptographic primitives.

4.3 Ease of Implementation

Although an algorithm should be easy to understand and implement, serious problems may arise. Laypersons could realize an algorithm without understanding the underlying mathematics, which could lead to fatal weaknesses (”footgun”). For this reason, the use cases should also be examined and classified to see whether the necessary post-quantum algorithms provide an attack surface in this case.

4.4 Physical Attacks

Side-channel attacks are a widespread tool to break cryptographic primitives. The use cases can give valuable information which of these attacks may or may not be relevant (e.g., because the attacker has no physical access to the device). The necessary mechanisms and use cases should, therefore, also be evaluated according to which side-channel attacks have already been investigated and how long the research community is already working on such attacks. It is important to consider, if such attacks exist, how expensive or efficient the countermeasures are in regard to a given use-case. Similarly, constant-time implementations also need to be evaluated in a given context, and the given security parameters.

An indication for any imposed threat on a device by implementation attacks is the operating environment of the device. For most of the side-channel and fault injection attacks, excluding cache timing attacks, the attacker needs to have physical access to the target or needs to be in very close proximity for a specific time. For the scenario of the upcoming remote side-channel and fault injection attacks (e.g., Rowhammer [Kim+14], Meltdown [Lip+18], Spectre [Koc+19], or Plundervolt [Mur+20]), the previous assumption on close proximity does not completely hold. Hence, implementations of cryptographic algorithms via side-channel and fault-injection attacks might be conducted if the algorithm is executed in an untrustworthy environment. In a trusted environment, implementation attacks are of no concern. Of course, the risk of a mounted implementation attack increases, if the environment becomes not trusted. If the environment is only controlled instead of trusted, mounting is possible but still considered very hard. In this case, an attack can at least be detected. If the environment of operation is considered not secure, countermeasures against implementation attacks should be considered for sure. However, the implementation of countermeasures against implementation attacks like side-channel and fault injection attacks highly depends on the application-specific protection need.

4.5 Additional Remarks

As additional remarks, the use cases should help to outline PQC-specific properties and behavior of the algorithms, e.g., the property of statefulness of some signature algorithm impact architectural and implementation decisions. These facts are important because current signature schemes do not have this restriction. Hence this can have severe impacts on industrial applications. Moreover, it could also be interesting to use only algorithms from one family (e.g., lattice-based PQC) to keep the code base of a practical implementation small.

From an industrial point of view, it is currently wholly open which methods can be used in the long term. For this reason, the use cases should be examined to see whether the underlying primitives have already been standardized by NIST (or similar organizations) or whether this is intended soon.

5 D1.3 - Requirements on Hardware Platforms

5.1 Consideration Aspects

Looking at the vast amount and the diversity of the given use cases, it seems necessary to group the use case requirements into a smaller number of hardware requirement categories. It would not be helpful to end up with a dozen different hardware platforms and, therefore, the same number of different demonstrators. While this categorizing approach fits well for Micro Controller Units (MCU) and Central Processing Units (CPU), a more differentiated look into the choice and the programming of Field-programmable Gate Arrays (FPGA) is needed. A broader view of what the QuantumRISC project wants to achieve has to be taken into account when it comes to FPGAs. For example, the European strategies for digital sovereignty, the future production costs of application-specific integrated circuits (ASIC), or the license models for development- and design tools are of some importance with the choice of FPGA platforms and tools. Additionally, the actual candidates for Post-quantum Cryptography (PQC) are in a wide range of different algorithmic categories and cryptographic primitives.

In the next chapters, we will start with the definition of crypto agility in a new way, focusing on PQC and hardware development, calling it "Hardware crypto agility" and draw a line to the term "HW-SW Co-design." After that, the hardware platforms and the requirement mappings are defined. The later chapters conclude with the considerations about FPGAs, the hardware programming languages, and tooling, merging them into the FPGA requirements.

5.2 Hardware Crypto(graphic) Agility

There are two terms widely used when it comes to cryptographic implementations:

- Crypto(graphic) agility and
- Hardware-/Software Co-design (HW/SW Co-design).

While the first one (Crypto agility) mostly defines the need for easy and quick exchange ability of cryptographic algorithms in an already productive environment, for example when it comes to freshly discovered security vulnerabilities. Crypto agility applies to pure software implementations and as well to the exchange of hardware. In the context of the QuantumRISC project this maps to the implementations for MCUs and CPUs and is not any requirement aspect for the choice of MCU and CPU hardware platforms than more a design choice for Application Programming Interfaces (API) on the software level. It might also come into account with the integration of the parts of the demonstrator.

A common definition of HW/SW Co-design is the simultaneous consideration of software- and hardware parts and dependencies for building a system to optimize them together for meeting the system level objectives. This surely applies strongly to the broad range of use cases in this project. The requirements on MCU and CPU platforms to enable a smooth HW/SW Co-design are shown in the upcoming Section 5.3.1 and are by most derived from the use case requirements with some additions (see Table 5.1 for an example).

One of the bigger parts inside the QuantumRISC project will focus on the development of hardware with FPGAs. Combined with the actual known algorithms for PQC, some additional aspects have to be considered when it comes to the requirements for FPGA hardware development:

- PQC does not feature the "one-solves-it-all" algorithm so far. And it is foreseeable that way more different cryptographic primitives will be involved than in standard Public Key Cryptography (RSA, ECC, etc.).
- Flexible combinations of the needed primitives.
- The rise of open source tooling and FPGA-chip solutions. As with Kerckhoffs's principle the development should contain as less secrets as possible.
- Licensing models and Non-disclosure Agreements (NDAs) for vendor tools and FPGA platforms.
- The European strategies for digital sovereignty. Europe should become more independent in hardware development and production (5G discussion e.g.)
- The common concept of blackbox IP-Cores might not fit.
- Easy exchange of modules through standardisation (e.g., RISC-V ISA)
- Reusability of modules in the sense of hardware libraries. New concepts through Hardware Description Languages (HDL) like SpinalHDL.
- Choice of Softcore implementations (e.g., VexRiscv).

While all of the mentioned aspects could be seen inside the concept of HW/SW Co-design, some of them are on a deeper level, then meeting the system-level objectives for a cryptographic system. Take alone the point of open source tools for getting independent from possible malicious injections done by the toolchain. This mitigation is most likely not a system-level requirement in HW/SW Co-design. Another point outside the scope of HW/SW Co-design is the European strategy for digital sovereignty. One will hardly see this in a technical system specification (maybe coming in the future).

Furthermore, the aspects of exchanging and reusing modular components fit more into the definition of cryptographic agility. Therefore we name this wide-range set of aspects above "Hardware cryptographic agility" to differentiate the wide variety and the broad origins of these requirements. This list is preliminary and might be expanded during the project. In the following chapters, we dive deeper into what they mean for choosing the platforms, toolchains, HDLs, and softcores.

Requirement	**Value**
Use case	Secure Download 3.2.1
Abstraction model part	ECU
Main Task	Verify signature
PQC Algorithm	XMSS - hashbased Signatures
Signature size	? KB
Public key size	? KB
Tree size	? KB
Latency	Low
Computation power	Deeply embedded
RAM	Deeply embedded
Storage	Deeply embedded
Storage hardend	No
Hardware accelerators	SHA256, SHA3
# Key pairs	<1000
Security level	192 bits
Security goals	Verification, Integrity

Table 5.1: Example of a set of derived requirements for a specific use case.

5.3 Hardware Platforms

5.3.1 Microcontroller Units (MCU) and Central Processing Units (CPU)

Most of the requirements for MCUs and CPUs can be derived from the properties and requirements for use cases defined in Section 3.1.3. Each use case contains a table about them (Example: Table 3.5). As not all fields in these tables are needed as requirements for MCUs or CPUs, a subset could be sufficient. Such a possible subset can be seen in the example at the end of this chapter. This set of information should be good to go for the choice of the key features on MCUs and CPUs, for example, the vendor, speed, RAM size, and register width.

Further requirements for MCUs and CPUs are derivable from the thread models and security features, which are tied to the abstraction model, defined in Section 3.1. Again each use case gives an own instantiation of these. From this information, the additional, necessary chip or board features can be defined, for example, hardened storage, connectivity, or accelerators.

The next step in the QuantumRISC project is work package WP2 (Choice and Enhancement of PQC-Algorithms and Protocols). With choosing the PQC algorithms and defining their parameters, another necessary piece of information towards the choice of hardware platforms is defined. Not every PQC algorithm fits into any of the defined use cases. A signature-only algorithm would not make any sense combined with a use case that needs key derivation. Instead of creating a vast, multidimensional matrix with all combinations of the requirements mentioned above, a define-by-need approach for each set of requirements is taken.

An example set of requirements is shown in Table 5.1.

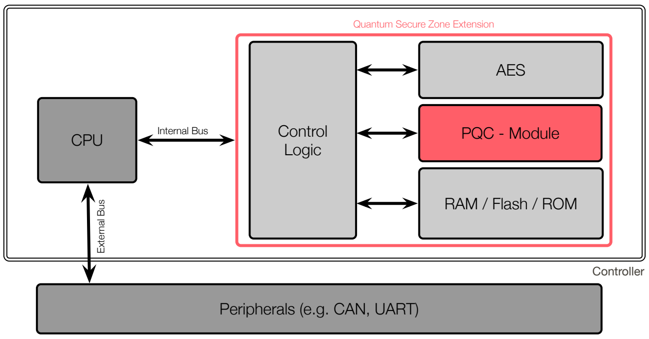

Figure 5.1: A simple hardware architecture for PQC.

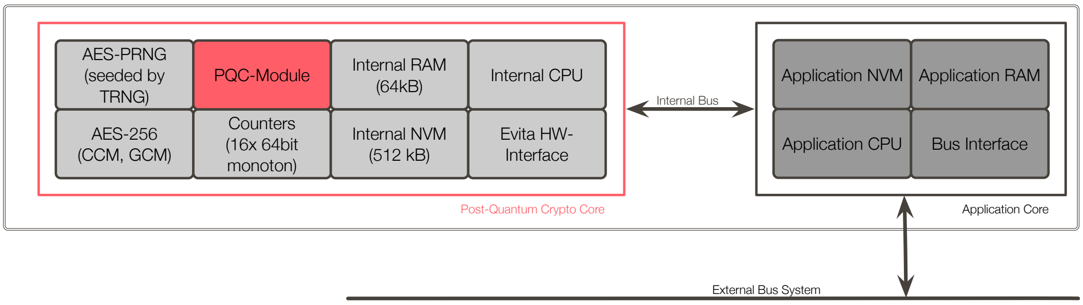

Figure 5.2: A complex hardware architecture for PQC.

5.3.2 Some Architectural Considerations

Currently, two underlying hardware architectures for cryptographic applications are common in the automotive industry. The first simple approach (cf. Figure 5.1) extends the application CPU by basic cryptographic primitives, i.e., both the application and the cryptographic operations work on the same hardware. This approach simplifies attacks, thus weakening the implemented protection measures and should only be considered for applications with low protection requirements. The significant advantage of this approach is the cost since only a small additional effort is required.

5.3.3 Field Programmable Gate Arrays (FPGA)

As described in Section 5.2 (Hardware crypto agility), the requirements for choosing an FPGA platform do not solely depend on the requirements derived from the use cases. Before we map the named aspects as requirements for FPGA platforms, the necessary properties get categorized in Table 5.2.

Based on this categorization, a rough mapping of the considerable aspects to the property categories can be defined as follows:

- Choice of PQC algorithms and parameters get mapped onto
  - Chip properties
  - Board properties

Category	Property
FPGA Chip	*Amount and/or size of* Logic Cells Lookup tables (LUTs) Digital signal processors (DSPs) In- and Output pins BlockRAM
Development Board	*Additional Components as* SD-/DDR RAM Controllers Interfaces
Connectivity	UART, USB, JTAG HDMI, VGA SD Card slot
Measurements	Measuring connectors Shunt resistors
Openness	Closed/Open Tools License models Non disclosure agreements (NDAs)

Table 5.2: Categorizing the FPGA properties into categories.

- Use case and demonstrator requirements get mapped onto
  - Board properties
  - Connectivity properties
- Side-channel analysis and countermeasures get mapped onto
  - Measurement properties
- System security, Malicious implants and injections, digital sovereignty, license models, NDAs get mapped onto
  - Openness

5.4 Softcores on FPGAs

At this point, it should be noted that FPGAs are used in this project as a rapid prototyping platform. It should be clear that all findings can be transferred relatively easily to ASICs, as these have a much better price/performance ratio for mass-market applications. Therefore, the research on SW/HW co-design on FPGAs is very beneficial for the application of the obtained result for the design of SoC or ASIC later on.

A flexible, robust, and open softcore implementation of the RISC-V ISA is required for evaluation, testing, and benchmarking of the algorithms to be selected. The obtained results must be verifiable by a broad scientific community. Therefore, only non-commercial RISC-V implementations with an open-source license are considered.

This approach also allows a cost-saving use in the industry, which improves the usability of the results achieved in the QuantumRISC project. Moreover, the results can be used in the long term.

The RISC-V implementation used in this project should be designed in such a way that any kind of change or improvement is easily possible. Certain parts of a calculation have to take place at different pipeline-steps. For example, long calculations should be distributed evenly over several pipeline-stages, so that the maximum clock frequency of the design is not significantly affected. Hence, softcores that are specially optimized, designed in less powerful hardware description languages or that have a high internal complexity (e.g., super-scalar architectures), cannot be considered. Instead, simple, stable, and "good-natured" implementations should be considered for the pending more in-depth investigations.

It is already apparent that the selected implementation should be extendable in a modular manner. This modularity is of particular importance if two different algorithms are to be investigated in a typical hardware environment. In this case, an existing cryptographic mechanism should not hinder the additional integration of another test candidate. For this reason, it is important that the softcore implementation used is modular and can be easily extended by new modules.

To be able to implement new algorithms quickly and without errors, a powerful hardware description language has to be used. The classic languages such as VHDL or Verilog hardly seem suitable for quickly creating hardware modules that can be flexibly extended, parameterized, and integrated. For this reason, new types of description languages must be used, which allow efficient working. Good examples are Chisel[1] or SpinalHDL[2]. For this reason, however, the selected CPU must also be implemented in the chosen language to efficiently process the planned work packages. Note that previous experience has shown, that those meta hardware description languages produce hardware-modules whose performance is in no way inferior to VHDL or Verilog implementations. Thus, the choice of implementation language is relatively free but has a significant impact on the results of the benchmarks. Hence, this approach leads to implementation with a small footprint on the FPGA and therefore results in low production costs.

5.5 Connectivity

MCUs, CPUs, and FPGAs are widely available as part of evaluation or development boards with onboard peripherals. It is to define which are needed. Again, the use cases can offer a way to set the requirements for the connectivity. Usually, more peripherals mean broader usage possibilities. But as a (or some) demonstrator(s) shall be built, the size of the boards should be taken into account as well. As an example, imagine a model car and the maximum size of parts that can be built into. A DIN-A5 sized board might not fit well into that car.

[1] https://www.chisel-lang.org/

[2] https://spinalhdl.github.io/SpinalDoc-RTD/

HDL name	Base language	From
SystemC	C++	Open SystemC Initiative (OSCI)
Chisel	Scala	UC Berkeley
SpinalHDL	Scala	Charles Papon
MyHDL	Python	Jan Decaluwe
Cλash	Haskell	University of Twente and Myrtle Software Ltd

Table 5.3: Examples of high-level HDLs.

5.6 Hardware Description Languages (HDL)

Programming FPGAs is done with Hardware Description Languages (HDL). They describe the hardware circuits that are built inside an FPGA. The same HDLs are used to describe the circuitry of Application Specific Integrated Circuits (ASIC). Therefore HDLs have wide usage in the development and production of different hardware. Two common and standardized HDL are

- Verilog and
- Very High-Speed Integrated Circuit Hardware Description Language (VHDL).

The aspects of HW crypto agility (Section 5.2) and the requirements regarding the softcore (Section 5.4) defined feature sets for HDLs that can hardly be fulfilled by these two common, but quite inflexible ones. In the past decade, a lot of community effort went into constructing new HDL concepts. The most interesting ones for usage in the QuantumRISC project are keeping a standardized and industry-accepted format while additionally offering some paradigms from classic software design. That means programming in a high-level language (Python, Scala, SystemC), including the benefits of object-orientated programming, functional programming, polymorphism, generic instantiation, recursion, and many more are desired. The high-level code then gets “compiled” (or “transcribed”) to a standard HDL (Verilog, VHDL). This approach leads to short and good human-readable source code while still maintaining the existing standards. Some of these high-level HDLs are listed in Table 5.3.

An extended list of HDLs can be found on the corresponding Wikipedia page[3]. The choice of the HDL for this project has a dependency on the choice of the softcore. For an efficient workflow, the softcore should be written in the same HDL as the future FPGA implementations. Not quite a hard requirement, but the in-house knowledge of the project partners with the chosen HDL has to be considered too. A first poll seems to prefer SpinalHDL and VexRiscv (Softcore).

5.7 Electronic Design Automation Tools

In the last decades, the choice of Electronic Design Automation (EDA) tools for developing FPGA implementations was quite a simple choice. FPGA vendors kept their chip-design and bitstream documentation as closed sources. Therefore the choice of

[3] https://en.wikipedia.org/wiki/Hardware_description_language

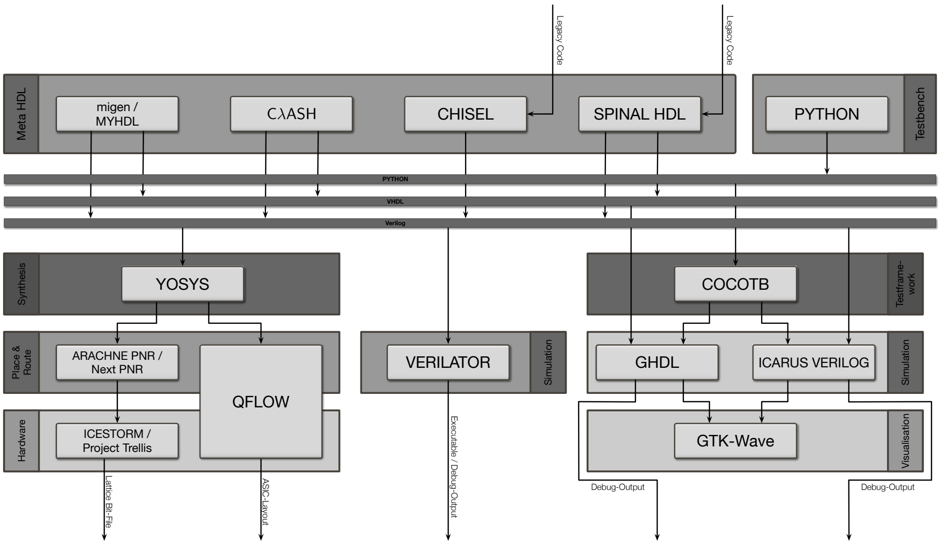

Figure 5.3: Some open-Source tools for EDA.

the FPGA platform was glued to the fitting vendor toolchains. Two main tuples are still dominating the overall FPGA market:

- Vendor Xilinx and the ISE/Vivado toolchain.
- Vendor Altera (now Intel) and the Quartus toolchain.

These vendor toolchains areas closed source as the chips themselves. Migrating projects to a platform of a different vendor are solely possible on HDL levels and forces to build the whole hardware design around it from scratch. In terms of requirements, this leads to a simple conclusion: Choose an FPGA vendor, and you are stuck to the toolchain from the same vendor. The same applies to the other direction. When choosing a vendor toolchain, you got to go with the same vendor for the FPGA.

Since 2013 this scenario relaxed in some aspects. While not going into too much detail about the different stages of the development process for FPGA implementations, the open-source toolchain “Yosys and Icestorm” got published. Since then, some more degrees of freedom opened up in setting requirements for FPGA tooling. While the first supported FPGA chips were reverse-engineered by community effort, right at the writing of this document, the first vendor opened his chip- and bitstream documentation for open source usage. It is foreseeable that during the running time of the QuantumRISC project, more openness comes to FPGA development.

Moreover, Open Source tools with nearly the same capabilities as the vendor tools could be in reach during the project Figure 5.3). This trend could lead to huge steps in terms of security against hardware trapdoors and fault injections. Digital sovereignty overall, and especially in Europe, would profit massively from such options.

5.8 Preliminary Conclusion on Hardware Selection

The criteria described here for selecting the hardware are intentionally not very precise. On the one hand, the still existing uncertainties in the selection of algorithms and, on the other hand, more specific requirements resulting from the industrial application. Further valuable additional information can be found in [Alb+19]. The recent paper [SW20] gives deep insights into possible selection criteria, especially from an industrial point of view.

Bibliography

[Alb+19] Martin R. Albrecht et al. "Implementing RLWE-based Schemes Using an RSA Co-Processor". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, Issue 1 (2019), pp. 169–208. DOI: 10.13154/tches.v2019.i1.169-208. URL: https://tches.iacr.org/index.php/TCHES/article/view/7338.

[Bar06a] J. Christopher Bare. *Attestation and Trusted Computing*. 2006. URL: https://courses.cs.washington.edu/courses/csep590/06wi/finalprojects/bare.pdf (visited on 03/09/2020). Access via https://courses.cs.washington.edu/.

[Bar06b] Elaine B. Barker. *SP 800-89. Recommendation for Obtaining Assurances for Digital Signature Applications*. Tech. rep. Gaithersburg, MD, United States, 2006. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-89.pdf.

[BPM18] Milan Brož, Mikuláš Patočka, and Vashek Matyáš. "Practical Cryptographic Data Integrity Protection with Full Disk Encryption". In: *ICT Systems Security and Privacy Protection*. Ed. by Lech Jan Janczewski and Mirosław Kutyłowski. Cham: Springer International Publishing, 2018, pp. 79–93. ISBN: 978-3-319-99828-2.

[Eur09] Seventh Research Framework Programme (2007-2013) of the European Community. *Specification and evaluation of e-security relevant use cases*. Dec. 1, 2009. URL: https://www.evita-project.org/Deliverables/EVITAD2.1.pdf (visited on 11/13/2019). Access via https://www.evita-project.org.

[FR14] E. Fielding and J. Reschke. "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content". In: *IETF Request For Comments* 7231 (June 2014).

[Gro96] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph].

[HiS09] HiS. *Secure Hardware Extension (SHE) functional specification*. Tech. rep. v1.1. Hersteller Initiative Software, 2009.

[ISO13] ISO. *Road vehicles — Unified diagnostic services (UDS) — Part 1: Specification and requirements*. Standard. Geneva, CH: International Organization for Standardization, Mar. 2013.

[Kan+19] Liron Kaneti et al. "Secure Boot Revisited: Challenges for Secure Implementations in the Automotive Domain". In: *17th Embedded Security in Cars (escar) Europe*. Nov. 2019.

[KH17] Dmitry Kasatkin and Serge Hallyn. *Integrity Measurement Architecture (IMA)*. Tech. rep. 2017. URL: http://downloads.sf.net/project/linux-ima/linux-ima/Integrity_overview.pdf.

[Kim+14] Y. Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 2014, pp. 361–372.

[Koc+19] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1–19.

[Lip+18] Moritz Lipp et al. "Meltdown: Reading Kernel Memory from User Space". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.

[Lip17] Andrew Liptak. *Tesla extended the range of some Florida vehicles for drivers to escape Hurricane Irma*. Sept. 10, 2017. URL: https://www.theverge.com/2017/9/10/16283330/tesla-hurricane-irma-update-florida-extend-range-model-s-x-60-60d (visited on 11/13/2019). Access via https://www.theverge.com.

[Mur+20] Kit Murdock et al. "Plundervolt: Software-based Fault Injection Attacks against Intel SGX". In: *41st IEEE Symposium on Security and Privacy (S&P'20)*. 2020.

[Pre16] The Netherlands EU Presidency. *Quantum Manifesto*. May 17, 2016. URL: http://english.eu2016.nl/documents/publications/2016/05/17/quantum-manifesto (visited on 05/18/2019). Access via https://archive.org/.

[San+13] S. Santesson et al. "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP". In: *IETF Request For Comments* 6960 (June 2013).

[Ser06] J. Sermersheim. "Lightweight Directory Access Protocol (LDAP): The Protocol". In: *IETF Request For Comments* 4511 (June 2006).

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J. Comput.* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397.

[SW20] Marc Stöttinger and Wen Wang. *Post-Quantum Secure Architectures for Automotive Hardware Secure Modules*. https://eprint.iacr.org/2020/026. 2020.

[WS09] Dr. André Weimerskirch and Dr. Kai Schramm. *Using software flashing to secure embedded device updates*. Apr. 13, 2009. URL: https://www.embedded.com/using-software-flashing-to-secure-embedded-device-updates/ (visited on 11/13/2019). Access via https://www.embedded.com.