Automated Post-Quantum Certificate Management for Industrial Internet of Things Infrastructures

Master's Thesis of

Kiron Mirdha

at the Department of Informatics
KASTEL – Institute of Information Security and Dependability

Reviewer:	Prof. Jörn Müller-Quade
Second reviewer:	Prof. Martina Zitterbart
Advisor:	Dr. Sebastian Paul, Robert Bosch GmbH
Second advisor:	M.Sc. Astrid Ottenhues
Third advisor:	M.Sc. Marcel Tiepelt

01. August 2022 – 11. May 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, May 11, 2023

.....

(Kiron Mirdha)

Abstract

The Industrial Internet of Things (IIoT) is characterized by its high interconnectedness enabling data exchange across private and public networks. In order to protect the authenticity of industrial devices and applications against cyber attacks, current best practices typically involve Public Key Infrastructures (PKIs). While PKI solutions are well established in the Web, recent studies suggest that their realization in industrial applications is often insufficient.

Moreover, the long lifespan of IIoT devices necessitates protecting them against future threats, such as attacks aided by quantum computers. Especially the ongoing standardization efforts of post-quantum cryptography (PQC) by the National Institute of Standards and Technology (NIST) motivate research on its applicability in industrial networks.

In this thesis, we reduce the complexity of certificate management for IIoT devices by automating administrative PKI tasks. Furthermore, we addressed the quantum threat by incorporating post-quantum algorithms from NIST's standardization process. Our approach instantiates a use case specific version of the Lightweight Certificate Management Protocol (CMP) Profile for X.509 digital certificates. It considers the requirements of industrial networks and provides an automation concept for the main functions of certificate management: certificate request, renewal, and revocation. We analyzed the authentication of the proposed protocol in the symbolic model using the formal verification tool Verifpal and proofed that the exchanged messages are secure against a Dolev-Yao attacker under the notion of injective agreement. Our impact assessment showed that using the post-quantum signature scheme Dilithium2 instead of Elliptic Curve Digital Signature Algorithm (ECDSA) with the curve P-384 results in shorter execution times at the cost of larger message sizes. In particular, the execution time for generating request messages is reduced by a factor of ~ 7, and for validating their corresponding responses by a factor of ~ 120. Overall, we concluded that Dilithium is a viable post-quantum alternative – even for time-sensitive industrial applications.

Zusammenfassung

Das industrielle Internet der Dinge (IIoT) zeichnet sich durch seine hohe Vernetzung aus, die den Datenaustausch über private und öffentliche Netze hinweg ermöglicht. Zum Schutz der Authentizität von Industriegeräten und -anwendungen vor Cyberangriffen werden in der Regel Public-Key-Infrastrukturen (PKIs) eingesetzt. PKI-Lösungen sind zwar im Internet gut etabliert sind, neuere Studien deuten jedoch darauf hin, dass ihre Umsetzung in industriellen Anwendungen oft unzureichend ist. Darüber hinaus erfordert die lange Lebensdauer von IIoT-Geräten, dass sie gegen künftige Bedrohungen, wie etwa Angriffe mit Hilfe von Quantencomputern, geschützt werden. Insbesondere die laufenden Standardisierungsbestrebungen des NIST für Post-Quanten-Kryptografie motivieren die Forschung zu ihrer Anwendbarkeit in industriellen Netzwerken.

In dieser Arbeit wird die Komplexität des Zertifikatsmanagements für IIoT-Geräte reduziert, indem die administrativen PKI-Aufgaben automatisiert werden. Außerdem wird die Bedrohung durch Quantencomputer durch die Integration von Post-Quanten-Algorithmen aus dem NIST Standardisierungsprozess adressiert. Unser Ansatz instanziiert eine anwendungsspezifische Version des Lightweight Certificate Management Protocol (CMP) Profils für digitale X.509-Zertifikate. Es berücksichtigt die Anforderungen industrieller Netze und bietet ein Automatisierungskonzept für die Hauptfunktionen der Zertifikatsverwaltung: Zertifikatsanforderung, -erneuerung und -widerruf. Die Authentizität des vorgeschlagenen Protokolls wird im symbolischen Modell mit dem formalen Verifikationswerkzeug Verifpal analysiert. Es wird gezeigt, dass die ausgetauschten Nachrichten gegen einen Dolev-Yao-Angreifer unter dem Begriff der injektiven Vereinbarung sicher sind. Eine erste Analyse hat zudem ergeben, dass die Verwendung des Post-Quantum-Signaturalgorithmus Dilithium2 anstelle von ECDSA mit der Kurve P-384 zu kürzeren Ausführungszeiten führt, allerdings auf Kosten größerer Nachrichtengrößen. Insbesondere wird die Ausführungszeit für die Erzeugung von Anforderungsnachrichten um den Faktor ~ 7, und für die Validierung der entsprechenden Antworten um einen Faktor von ~ 120 reduziert. Insgesamt lässt sich daraus schlussfolgern, dass Dilithium eine praktikable Post-Quantum-Alternative ist – sogar für zeitkritische industrielle Anwendungen.

Contents

List of Figures

List of Tables

List of Definitions and Theorems

Acronyms

ACME Automatic Certificate Management Environment 15

API application programming interface 53

BCPQC BouncyCastle Post-Quantum Security Provider 63

CA Certificate Authority 13

CIM Computer Integrated Manufacturing 6

CLI command-line interface ix

CMC Certificate Management over CMS 14

CMP Certificate Management Protocol i

CMS Cryptographic Message Syntax 14

CoAP Constrained Application Protocol 27

CRL Certificate Revocation List 14

CRMF Certificate Request Message Format 25

ECDSA Elliptic Curve Digital Signature Algorithm i

EJBCA Enterprise JavaBeans Certificate Authority 53

EST Enrollment over Secure Transport 15

EUF-CMA existential unforgeability under chosen message attack xi

FTS/OTS few or one time signature 11

HTTP Hypertext Transfer Protocol 7

IEC International Electrotechnical Commission 6

IETF Internet Engineering Task Force 14

IIoT Industrial Internet of Things i

IoT Internet of Things 15

1. Introduction

Today's digital signatures are based on hard mathematical problems such as integer factorization, i.e., Rivest–Shamir–Adleman (RSA), and the discrete logarithm problem, i.e., ECDSA. In 1994, the mathematician Peter Shor developed a quantum algorithm which efficiently solves the hidden-subgroup problem for finite Abelian groups [Sho94]. As a result of this, currently used digital signature algorithms can be broken with a quantum computer. Although a sufficiently large quantum computer has not been built yet, current advancements in the field of quantum computing indicate that it is only a matter of time until a quantum computer can effectively break RSA and ECDSA [Jos+22]. Michele Mosca, an expert on quantum computing, estimated in 2017 that there is a "1/6 chance [of breaking RSA-2048] within a decade and a 1/2 chance within 15 [years]" [Mos18]. This hypothesis is supported by the roadmap of the IBM Quantum processor technology which envisions to start developing large-scale systems beyond 2026 [Gam22]. Considering that IIoT "devices are often expected to stay functional for years or even decades" [GKS19, RFC 8576], quantum technology poses a real threat for IIoT infrastructures when cryptographically relevant quantum computers are available earlier than the development and migration to PQC is completed [Mos18]. Therefore, Niederhagen and Waidner [NW17] conclude in their report that it is already necessary today to incorporate post-quantum secure technology in industrial applications to reduce the future risk.

1.1. Scope of this Thesis

In the thesis, a protocol shall be developed which automatically manages the entire life cycle of post-quantum certificates on IIoT devices. It shall provide a solution to automatically and securely request new certificates, renew them upon expiration and revoke them when needed. To address future security requirements due to the long lifespan of IIoT devices, PQC for digital signatures will be analyzed regarding their feasibility in a resource-constraint IIoT environment and also their status in recent standardization processes. In the scope of this thesis, the proposed protocol shall be analyzed and evaluated w.r.t. its security properties and the performance of the post-quantum signature algorithms compared to currently established ones using a prototype implementation.

Since mutual authentication is considered a requirement for secure IIoT infrastructures, the proposed certificate management protocol will enable manufacturers to securely enhance the networking of their facilities. Additionally, it aims to ease the migration towards upcoming PQC signature standards by enabling post-quantum authentication within existing IIoT production environments.

1.1.1. Industrial Internet of Things

In recent years, industrial control systems have started to migrate from isolated networks and proprietary communication protocols to highly connected networks and IP-based protocols, referred to as IIoT. Gathering data, such as machine status, usage pattern, and production quality, and sharing these sets of data amongst intended peers internally or even externally via public cloud services, IIoT aims to improve the efficiency of industrial operations. This implicates that industrial devices are envisioned to share data about different aspects of industrial operation, such as business strategies, safety- and security-critical information as well as privacy relevant data, across public networks [SLB20; AJS20]. Due to this increased degree of interconnectedness and the sensitive nature of the transmitted data, IIoT infrastructures are even more vulnerable to cyber attacks than current industrial networks [SWW15]. Therefore, security mechanisms are a crucial requirement for IIoT.

One main security goal is to ensure the *authenticity* of devices and applications, i.e., there is a process that assures the correct identities of all communication partners. The current state of the art to provide network wide authenticity are digital certificates deployed via PKIs [Dah+22]. Existing PKI solutions are designed for use with the public Internet and enterprise IT, however, they are not well suited for IIoT networks. In IT networks, certificates are often provided manually to a small number of servers. This works because these systems deploy certificates for server authentication rather than mutual authentication. Different from the web, in IIoT environments the communication takes place to a large extent in a machine-to-machine fashion without any human interaction. In these use cases a server may, for instance, need to authenticate the origin of the received sensor data, hence mutual authentication. This necessitates the use of certificates on both the client and server devices, requiring a substantially larger number of certificates, favoring an automated solution for certificate management. Therefore, in IIoT networks it would be desired to automatically and securely request new certificates, renew and revoke them if necessary.

1.1.2. Research Questions

The goal of the proposed thesis is the development of an automated post-quantum certificate management system for IIoT infrastructures. Following questions shall be addressed in the design and implementation of the proposed approach:

1. How can an automated post-quantum certificate management be constructed for IIoT networks based on existing concepts and protocols?
2. Which security assumptions and goals is the proposed protocol based on and to which extent can they be verified?
3. How does the use of PQC affect the performance of the proposed certificate management system compared to traditional public key algorithms?

The main contributions are a concept for automated post-quantum certificate management (based on an existing certificate management protocol), its prototype

implementation, and a performance evaluation of the use of PQC digital certificates. To ensure the security properties of the used protocol, its security analysis is another major contribution of the thesis. For this, the security assumptions and goals as well as the attacker model of the system will be specified. In a second step, the security properties are verified for the proposed protocol.

1.2. Outline

This thesis is structured into six chapters: After this introduction, the state of the art and related work in the research fields touched by this work are reviewed in Chapter 2. This includes industrial networks, post-quantum digital signatures, certificate management protocols and automated security verification. Based on the conclusions of this chapter, an instantiation of the Lightweight CMP Profile for our industrial PKI architecture is described in Chapter 3. This specification is then used in Chapter 4 to conduct a security analysis using the verification tool Verifpal. The analysis comprises of the security assumptions and goals, the attacker model, the protocol model and the verification results. In Chapter 5, a proof of concept of the proposed post-quantum certificate management solution is presented. It provides a proposal for embedding the PKI architecture into an industrial network and an automation concept for the certificate management operations. Furthermore, the integration of post-quantum algorithm support into an existing protocol implementation is documented here. Then, this prototype is used to assess the impact of post-quantum certificates on memory and execution time. Finally, the results of this thesis are summarized in Chapter 6 and an outlook gives impulses for future work.

2. State of the Art

The subject of this thesis combines multiple fields of research. This chapter provides an overview of all aspects of the topics relevant for the understanding of the contributions in this thesis. First, the industrial networks are introduced as target environment in Section 2.1. Then, the theoretical background for post-quantum digital signatures as well as a comparison of existing signature schemes is presented in Section 2.2. Afterwards, certificate management is explained in Section 2.3 and existing protocols are compared with regard to their suitability in industrial networks. Finally, the topic of computer aided security verification including relevant security definitions and notions is described in Section 2.4 and the use of Verifpal as verification tool is motivated. For each research field, a brief review of related work is provided at the end of each section.

2.1. Industrial Communication Systems and Networks

In manufacturing and industrial facilities, an industrial communication system refers to a large class of automation systems that offer control and monitoring capabilities. These are distributed systems embedded in a large architecture comprising of plant areas with common and shared applications, area-specific control devices and associated field devices, all interconnected via different network equipment, the industrial network. Such networks are increasingly migrating to Ethernet and Internet Protocol (IP) based technologies using both wired and wireless connectivity. At the field level, however, non-IP-based field bus media and protocols are still used. Compared to a business network or the internet the availability of data is much more prioritized in industrial networks. As a result, real-time protocols are used to a greater extent as well as fault-tolerant networks connecting endpoints and servers. Additionally, industrial network architectures need to provide low / consistent latency because the deployed applications and protocols usually depend on deterministic communication and precise timing requirements [KL15b].

The heterogeneous environment and the ubiquitous connectivity can pose a significant security risk if proper design considerations are not taken. Focusing on network attacks, [GT11] divides them into four classes:

1. *Interception attacks:* The attacker attempts to gain unauthorized access to confidential data during the transmission over the network.

2. *Modification attacks:* The attacker attempts to modify messages while they are transmitted over the network.

3. *Fabrication attacks:* The attacker attempts to insert malicious data.

4. *Interruption attacks:* The attacker attempts to disrupt the communication between devices making the data unavailable.

A planned approach to security for industrial networks is the "defense-in-depth" principle relying on multiple security layers to protect valuable assets. This principle is intended to ensure that an attacker or an otherwise triggered incident cannot spread unhindered and cause damage by leveraging a single measure [KL15b].

In the following, a key concept of this approach is described in more detail: security zones and conduits.

2.1.1. Security Zones and Conduits

Many industrial communication systems are structured in a hierarchical way following the Purdue Reference Model for Computer Integrated Manufacturing (CIM) from 1989. Applying this network segmentation based on security requirements the International Electrotechnical Commission (IEC) defined the concept of security zones and conduits as specified in IEC-62443-1-1 [Int09] of the international standard series IEC-62443 "Industrial communication networks - Network and system security":

Definition 2.1.1 (Security zone)**.** *[A security zone describes a] grouping of logical or physical assets that share common security requirements. [...] A zone has a clear border with other zones. The security policy of a zone is typically enforced by a combination of mechanisms both at the zone edge and within the zone. Zones can be hierarchical in the sense that they can be comprised of a collection of sub-zones.*

Definition 2.1.2 (Conduit)**.** *A conduit is a particular type of security zone that groups communications that can be logically organized into a grouping of information flows within and also external to a zone. It provides protection measures for the communication channels within to allow secure communication across zones.*

If implemented correctly, the zone model complies with the *principle of least privilege* and the *principle of least route*. The first principle ensures a reduced attack surface by granting minimal privileges to users and devices. The second principle ensures minimal delay in data communication, which can contribute to lower utilization of transmission paths and communication devices, especially in a real-time network [KL15b].

Figure 2.1 illustrates a three-zone security model for industrial communication applications in a company adopted from Granzer and Treytl [GT11] : The inner control zone (or production zone) hosts the field-level communication systems, which are primarily located at the shop floor. The plant zone (or on-premise/service zone) is typically built upon IP based Local Area Networks (LANs) inside the plant. Finally, the enterprise zone connects multiple plants, remote maintenance sites, etc. In general, it is assumed that devices outside a security zone fulfill a lesser or different level of security, and thus, are not trusted by default. Therefore, all data transfer between

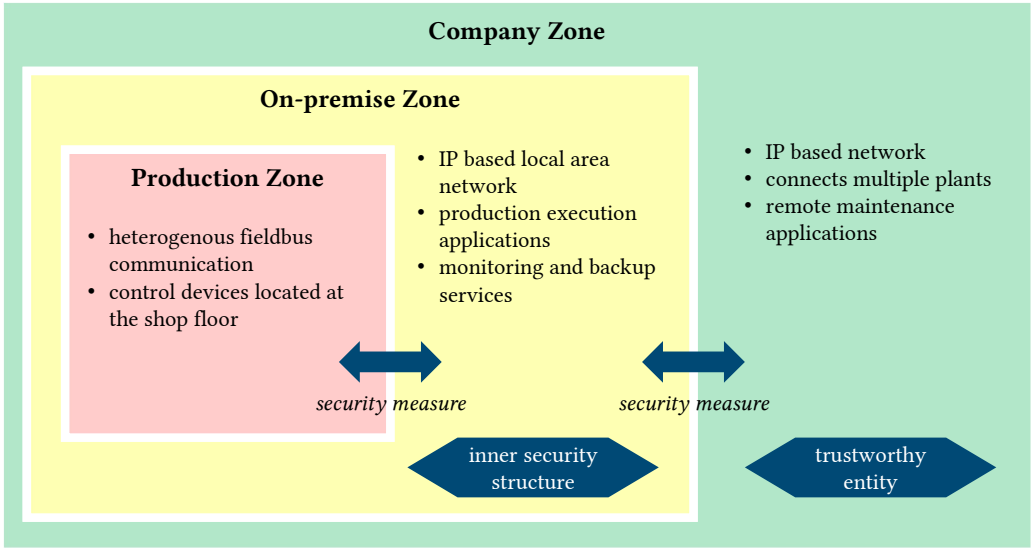


Figure 2.1.: Three-zone security model adopted from Granzer and Treytl [GT11]

zones must have a conduit which provides dedicated security measures. Typical examples are firewalls between the enterprise and service zone or application gateways between the service zone and the production zone.

2.1.2. Machine to Machine Communication

In industrial networks a majority of communication takes place between distributed applications to transfer data or to invoke functions without any human interaction. This is referred to as machine-to-machine communication. Martin Wollschlaeger [Mar11, Section 56.4.3] state that in the automation domain, the use of web services for machine-to-machine communication is increasing. A web service, as defined by W3C [W3C], is "a software system designed to support interoperable machine-to-machine interaction over a network." For this, it offers an interface to describe services in a machine-processable format. Other systems interacting with a web service typically transfer messages using Hypertext Transfer Protocol (HTTP) together with other web-related standards [W3C].

2.1.2.1. Communication Use Cases

Communication which requires communication across security zones can be assigned to three top level use cases: production execution, data streaming, and remote maintenance and support. The first two use cases represent machine-to-machine communication whereas the third involves human interaction.

Production Execution All tasks and processes which directly influence the product and its production. This use case allows conduits to have a permanent, bidirectional connection between the dedicated production execution services and IT systems in the control zone.

Data Streaming Applications such as control monitoring, energy data collection or backup service which involve the continuous provisioning of large amounts of data to a consumer for later use. In these cases, only outgoing communication from the control zone is permitted, the original request by the data consumer excluded.

Remote Maintenance and Support All uses cases in which a human initiates the communication. The communication can be bidirectional, but it is required that the conduit is set up on demand and is limited in time.

In the context of the thesis, the focus is on machine-to-machine communication, i.e., on the use cases production execution and data streaming, which can benefit from an automated certificate management.

2.1.2.2. Secure Industrial Communication Protocols

Based on the expected message pattern the two use cases can be grouped in two types of communication: The production execution use cases represent the client-server model and the data streaming calls for a more data-centric approach, the publish-subscribe model.

In both cases, the Transport Layer Security (TLS) protocol [DR08; Res18], which is already well-established in the IT domain, can be used to establish secure communication channels providing confidentiality, integrity and authenticity based on certificates. TLS is designed for client-server applications and only requires mandatory server authentication. But it is possible that the server also demands clients to authenticate themselves, thus, enabling mutual authentication.

For the publish-subscribe model currently the machine-to-machine communication protocol MQTT, originally an initialism of MQ Telemetry Transport (MQTT) [Ban+19] is widely studied for its use in IIoT networks. Profanter et al. [Pro+19] have shown that it is a lightweight protocol which is able to handle high latency and noisy network conditions. A central broker maintains the entire data set of it communication partners and distributes the information upon subscription. A client can publish messages corresponding to a specific topic (publisher) or receive messages by subscribing to topics provided by the broker (subscriber). MQTT specifies that TLS shall be used to ensure authentication between subscriber and broker as well as between publisher and broker. It should be noted that mutual authentication between a subscriber and a publisher is not possible with MQTT. However, this offers the advantage that a subscriber does not need to validate all publishers it subscribes to. Thus, a performance gain can be achieved compared to the client-server model.

2.2. Post-quantum Digital Signatures

Certificate-based authentication relies on the cryptographic primitive of digital signatures. However, as outlined in Chapter 1 current signature schemes cannot be relied upon once a quantum computer exists that can efficiently break them. Therefore,

post-quantum signature schemes are considered in this thesis and their integration into certificates and certificate management is investigated.

In December 2016, NIST initiated a PQC standardization process which is a competition-like process calling for proposals for quantum-safe [1] digital signatures, Public Key Encryption (PKE) schemes, and Key Encapsulation Mechanisms (KEMs). The standardization committee evaluates the submissions regarding their security guarantees, performance and algorithm and implementation characteristics [NIS22]. In the following, the signature scheme candidates are discussed in more detail. For this, first, the relevant definitions and security notions are specified. Afterwards, the different families of algorithms are briefly described including their representatives in the standardization process. Comparing these candidates regarding their feasibility it is reasoned that Dilithium2 will be used as an example for post-quantum certificate management in IIoT networks.

2.2.1. Definitions and Security Notions

To send an authenticated message, the sender first generates a public-private key pair (pk, sk). The public key pk is then published in a manner so that the receiver can obtain a legitimate copy of it afterwards. When the sender wants to transmit a message m, they compute a signature σ using the secret key sk and send both the message and the signature to the other party. The signature σ is obtained using a signing algorithm *Sign*. The receiver verifies the validity of the signature by running a verification algorithm *Vrfy*, which takes the public key pk, message m, and signature σ as input and determines whether the received signature is valid or not. Formally defined by Katz and Lindell [KL15a, Section 12.2]:

Definition 2.2.1 (Digital signature scheme)**.** *A* (digital) signature scheme *consists of three probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *such that:*

1. *The* key-generation algorithm Gen *takes as input a security parameter* 1^n *and outputs a pair of keys* (pk, sk)*. These are called the* public key *and the* private key*, respectively. We assume that pk and sk each has length at least n, and that n can be determined from pk or sk.*

2. *The* signing algorithm Sign *takes as input a private key sk and a message m from some message space (that may depend on pk). It outputs a signature* σ*, and we write this as* $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$*.*

3. *The* deterministic verification algorithm Vrfy *takes as input a public key pk, a message m, and a signature* σ*. It outputs a bit b, with* $b = 1$ *meaning valid and* $b = 0$ *meaning invalid. We write this as* $b := \mathsf{Vrfy}_{pk}(m, \sigma)$*.*

It is required that except with negligible probability over (pk, sk) *output by* $\mathsf{Gen}(1^n)$*, it holds that* $\mathsf{Vrfy}_{pk}(m, \mathsf{Sign}_{sk}(m)) = 1$ *for every (legal) message m. If there is a function*

[1] The terms "quantum-safe", "quantum-resistant" and "post-quantum" are used synonymously in this thesis.

ℓ such that for every (pk, sk) output by $\mathsf{Gen}(1^n)$ the message space is $0,1^{\ell(n)}$, then we say that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a signature scheme for messages of length $\ell(n)$.

Security of a signature scheme means that an adversary should not be able to produce a forgery even if they obtain signatures of other messages of their choice. For a fixed public key pk generated by a signer S, a *forgery* is a message m with a valid signature σ where the signer did not sign m previously. This security notion is referred to as EUF-CMA and is formally defined by Katz and Lindell [KL15a, Section 12.2] as follows:

Definition 2.2.2 (Signature experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n)$)**.** *Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and consider the following experiment for an adversary $\mathcal{A}$ and parameter n:*

1. *The key-generation algorithm $\mathsf{Gen}(1^n)$ is run to obtain keys (pk, sk).*
2. *Adversary $\mathcal{A}$ is given pk and access to an oracle [for the signature algorithm] $\mathsf{Sign}_{sk}(\cdot)$. The adversary then outputs (m, σ). Let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ asked its oracle.*
3. *$\mathcal{A}$ succeeds if and only if a) [the verification algorithm] $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$, and b) $m \notin \mathcal{Q}$. In this case, the output of the experiment is defined to be 1.*

Definition 2.2.3 (EUF-CMA)**.** *A signature scheme is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:*

$$\Pr\left[\mathit{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1\right] \leq \mathsf{negl}(n).$$

For their candidate signature schemes, NIST primarily considers security proofs in which the attacker uses classical queries to the signing oracle, rather than quantum queries [NIS22]. As previously stated, digital certificates are based on signature algorithms. Thus, in the following, the term "post-quantum certificate" shall refer to digital certificates which are considered secure under the given notion of EUF-CMA.

Additionally to the this security definition, NIST introduces a security strength categories which are defined by the security of "a comparatively easy-to-analyze reference primitive" [NIS22]. The security strength shall reflect how much computational resources an attack requires to break the relevant security definition, i. e., EUF-CMA. Table 2.1 summarizes the five *NIST security levels*, listed in increasing order: The classification is based on the range of security strengths provided by the current NIST symmetric cryptography standards, which are expected to offer significant resistance to quantum cryptoanalysis. In that sense, the security requirement assigned to each security level means, that a signature scheme of a certain level is at least as hard to break as the given symmetric primitive using the specified method. For example, any attack on a level 1 signature scheme "that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e. g., AES-128)" [NIS22].

Level	Security requirement
1	key search on a block cipher with a 128-bit key (e. g., AES-128)
2	collision search on a 256-bit hash function (e. g., SHA-256/ SHA3-256)
3	key search on a block cipher with a 192-bit key (e. g., AES-192)
4	collision search on a 384-bit hash function (e. g., SHA-384/ SHA3-384)
5	key search on a block cipher with a 256-bit key (e. g., AES-256)

Table 2.1.: NIST security levels

2.2.2. Candidates Signature Schemes

In July 2022, NIST completed the third round of the standardization process and decided to standardize the algorithms CRYSTALS-Dilithium [Léo+21], Falcon [Pie+20] and SPHINCS+ [Aum+22] as post-quantum signature schemes [CSR22]. Dilithium and Falcon belong to the family of lattice-based signature schemes, while SPHINCS+ is a hash-based scheme.

Lattice-based cryptography Lattice-based cryptography relies on hard problems on lattices. In linear algebra, a lattice is the set of all integer linear combinations of linearly independent vectors from a basis of $\mathbb{R}^n$. The crucial point here is that the basis for a lattice is not unique, which is utilized in the construction of hard problems. Dilithium depends on the module learning with error (LWE) problem and module short integer solution (SIS). Falcon is constructed with SIS over NTRU, a ring based public-key cryptosystem [HPS98]. According to Bernstein and Lange [BL17] lattice-based algorithms are relatively efficient to implement, but "more research is required to gain confidence in the security of lattice-based cryptography."

Hash-based cryptography Hash-based cryptography relies on hash trees or Merkle trees and few or one time signature (FTS/OTS) which use secure cryptographic hash functions, i. e., they are required to be preimage and collision resistant. When the signer signs a message with FTS/OTS the corresponding private key becomes a leaf in the Merkle tree and the resulting tree root is the public key. Using one time signature (OTS) the signer has to ensure that its private key is never reused which requires maintaining a key state. The stateless signature scheme SPHINCS+ alleviates this issue but with the cost of much larger signature sizes. Bernstein and Lange [BL17] conclude that hashed-based signature schemes using Merke trees are well-understood, and reliable regarding their security properties.

Table 2.2 shows the parameter sets for the three post-quantum signature candidates. Compared to the classical elliptic curve signature scheme ECDSA it is evident that both the key sizes and the signature sizes are significantly larger, except for SPHINCS+. With SPHINCS+ the key sizes are rather small, but as it is a stateless hash-based signature scheme, the signature sizes are in the range of $17 - 49$ kB. Regarding the use in certificates and with the goal to choose a resource-conserving algorithm, a

Signature algorithm	Public key size (Bytes)	Private key size (Bytes)	Signature size (Bytes)	Security level
ECDSA $_{\text{nistp384}}$	48	48	96	–
Dilithium2	1312	2528	2420	2
Dilithium3	1952	4000	3293	3
Dilithium5	2592	4864	4595	5
Falcon-512	897	1281	690	1
Falcon-1024	1793	2305	1330	5
SPHINCS+$_{\text{-SHA256-128f-simple}}$	32	64	17088	1
SPHINCS+$_{\text{-SHA256-192f-simple}}$	48	96	35664	3
SPHINCS+$_{\text{-SHA256-256f-simple}}$	64	128	49856	5

Table 2.2.: Post-quantum signature algorithms and parameter sets

signature scheme with key and signature sizes as small as possible would be desired. Thus, Falcon would be the best choice out of the three schemes.

All of them are secure under the notion of EUF-CMA (Definition 2.2.3). Depending on the parameter set, different security levels (Table 2.1) can be achieved. The larger the key sizes are chosen, the greater computational resources are required for an attacker to break the scheme.

Although Falcon would be the more favorable scheme due to its smaller key and signature sizes, NIST recommends the primary use of Dilithium because of its performance and less complexity of implementation [CSR22]. Therefore, in this thesis, we select the Dilithium parameter set with the smallest key and signature sizes, Dilithium2, to evaluate the post-quantum impact on the certificate management protocol.

2.2.3. Related Work

Bindel et al. [Bin+17] and Kampanakis et al. [Kam+18] have studied the integration of post-quantum signatures in X.509 certificates. One of the first performance evaluations of post-quantum authentication in TLS 1.3 was performed by Sikeridis, Kampanakis, and Devetsikiotis [SKD20]. However, they solely focused on server authentication. Later, this performance study has been extended for mutual authentication integrating all NIST round three candidates in the wolfSSL library [Pau+22]. To the best of our knowledge no research has been conducted on the integration of PQC in certificate management protocols.

2.3. X.509 Public Key Infrastructure and Certificate Management

X.509 Public Key Infrastructure (PKI) [Int22] is a standard for managing digital certificates, which are used to establish trust and enable secure communication over the internet. A PKI involves a hierarchical system of trusted authorities that issue, manage, and revoke digital certificates, which contain public keys used for encryption and authentication.

In the following, the main components of a PKI and the processes of certificate life-cycle management are introduced. Afterwards, existing certificate management protocols are compared and related work regarding the use of certificate management in industrial network is reviewed.

2.3.1. PKI Components

The following terminology adopted from Brockhaus, Oheimb, and Fries [BOF23] and Adams et al. [Ada+05] will be used in this document to describe the entities involved in the certificate management.

Certificate Authority (CA) A Certificate Authority (CA) issues certificates.

Registration Authority (RA) A CA can delegate certificate management tasks including end entity authentication and authorization checks for incoming requests to a Registration Authority (RA), which is an optional PKI component. An RA can also convert between several certificate management protocols and other protocols that offer certificate management-related operations.

End Entity An end entity is typically a device or application that possesses a public-private key pair for which it requires a certificate.

PKI management operation A PKI management operation describes the entirety of all messages belonging to a single transaction between PKI entities.

PKI management entity All entities of the PKI which are not end entities are referred to as PKI management entities, e. g., the CA and the RA.

PKI entity A PKI entity is an end entity or a PKI management entity.

2.3.2. Certificate Life-cycle Management

Certificate management describes all processes a digital certificate undergoes over the span of its entire life time. Buchmann [Buc13, Chapter 7.1] differentiates between three phases: the *Certificate Generation Phase* when the certificate is initially issued, the *Certificate Validity Phase* during which it can be used for authentication, and finally the *Certificate Invalidity Phase*.

Certificates are issued with a limited validity period because the identity of the entity authenticated by CAs might change or the security of the public key may

be compromised either accidentally or even intentionally by an attacker. Therefore, certificate life cycle management is needed to maintain the necessary information and manage the transitions between the phases Atreya et al. [Atr+02, Chapter 3]. The four main tasks in this life cycle are initialization, certificate request, renewal and revocation.

Initialization During the initialization of an application or device information about the PKI such as the address of the CA including its certificate are provisioned. For the scope of the thesis, it is assumed that the provisioning of the PKI details is performed out of band.

Certificate Request Having information about the trusted CA, the application can now request a certificate. For this, it sends its identity and the public key which shall be authenticated by the CA and provides a *proof of possession* to prove that they own the private key corresponding to the public key. The CA validates the proof of possession and on success, it replies with the signed application certificate.

Certificate Renewal Upon expiration of its certificate an application can request a renewal. Depending on the security policy it one can continue to use the public key linked to the certificate or replace it with a new key. The renewal is simpler than a certificate request since using the signing key of the already authenticated certificate is a sufficient proof of identify towards the CA. Therefore, it is possible to automate the renewal process.

Certificate Revocation Sometimes it is necessary to actively revoke a certificate even before it expires. The most common reason for that is a change in the identity of its owner. If a CA revokes a certificate, it also has to propagate this information to all communication partners of the certificate owner. Typically, this is accomplished with a Certificate Revocation List (CRL) which is maintained by the CAs.

2.3.3. Comparing Existing Certificate Management Protocols

The PKI standardization for X.509 certificates is mainly driven by the Internet Engineering Task Force (IETF) and targets internet infrastructures: The CMP [Ada+05, RFC 4210] specifies all aspects of certificate management. In particular the communication protocol for certificate enrollment, renewal and revocation are described. The messages are self-contained and thus, can be transported via various means, typically over HTTP or emails. CMP is considered rather complex in its implementation and message syntax [Atr+02]. Therefore, Certificate Management over CMS (CMC) [SM08, RFC 5272] was introduced to support the simpler, widely used Cryptographic Message Syntax (CMS) [Hou09, RFC 5652] message format. Additionally, in CMC the communication overhead for a management operation is reduced to a single request and response.

Later, the protocols Simple Certificate Enrollment Protocol (SCEP) [Gut20, RFC 8894 - internet draft] and Enrollment over Secure Transport (EST) [PYH13, RFC 7030] have been published. They are both based on CMC and focus on methods for automated certificate enrollment on managed devices. The difference lies in whether a shared secret (SCEP) or TLS (EST) is used for authentication. Note that EST does not support querying for a certificate's revocation status. However, additional mechanisms can be used to check the validity of a certificate, such as CRLs and the Online Certificate Status Protocol (OCSP) [San+13, RFC 6960] can be used to check the validity of a certificate.

A relatively new protocol is Automatic Certificate Management Environment (ACME) [Bar+19, RFC 8555], known for it's use in the *Let's Encrypt* project, though its application is limited to the authentication of web servers. Compared to aforementioned protocols, ACME allows automatic verification of the ownership of an internet domain, and thus, enables simple automated certificate issuance without user interaction. Currently, an ACME based approach is proposed as an experimental draft to provision X.509 certificates on Internet of Things (IoT) devices leveraging a private CA instead of a public one [Swe22].

With the increasing demand of certificates especially for IoT devices, in 2020 the Limited Additional Mechanisms for PKIX and SMIME (LAMPS) working group of the IETF published a first draft a simple and automated certificate management protocol based on CMP, the Lightweight CMP Profile. Since then it has been revised several times, the version which is referred to in this thesis is version 21 [BOF23]. As stated before CMP itself is a complex framework for realizing certificate management in various ways, an instantiation of the protocol is referred to as profile. The Lightweight CMP Profile focuses especially on IoT and industrial uses cases by reducing the set of procedures to the most crucial operations and supporting only the mandatory options. For example, the necessary interaction for a certificate management operation can be reduced to a single message round-trip, similar to CMC.

Table 2.3 shows the mentioned certificate management protocols in direct comparison: Regarding the feature coverage, the CMP protocols are most suited to provide certificate management over the whole certificate lifecycle. Furthermore, the use of self-contained protection mechanisms allows flexibility towards the transport protocol and thus, the target network infrastructure. Additionally, it saves the overhead of establishing a TLS connection which again requires its own certificates compared to CMC and EST. Most importantly, due to its targeted profile for industrial networks, the Lightweight CMP Profile is further studied in this thesis regarding its extensibility for post-quantum certificates.

2.3.4. Post-quantum X.509 Certificates

To use post-quantum authentication, X.509 certificates need to provide post-quantum algorithm support. Sikeridis, Kampanakis, and Devetsikiotis [SKD20] analyzed in their work which fields of the certificate are affected as shown in Figure 2.2. The *subject* of the certificate is the identifier of the end entity that holds ownership over the public key provided in the *subject public key info* field. In this field, the post-quantum public

	CMP	Lw. CMP	CMC	SCEP	EST
Cert. request	✓	✓	✓	✓	✓
Cert. renewal	✓	✓	✓	✓	✓
Cert. revocation	✓	✓	–	–	–
Protection	Self-contained	Self-contained	TLS	Shared secret	TLS

Table 2.3.: Comparison of existing certificate management protocols

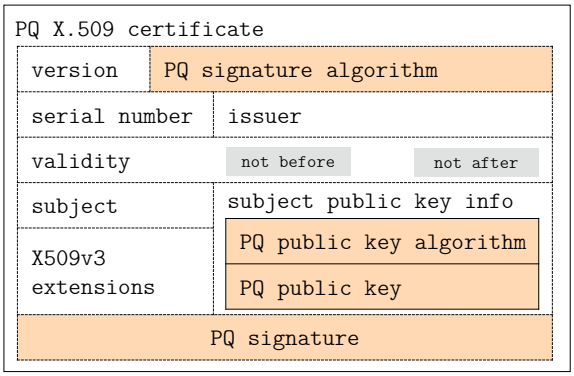


Figure 2.2.: Post-quantum (PQ) X.509 certificate

key and the identifier of its post-quantum signature algorithm are placed. The issuer then signs the certificate using the signature scheme indicated in the *PQ signature algorithm* field. The post-quantum signature is attached at the end of the certificate in the *signature* area. The size of the X.509 certificate and the related certificate chains will increase as a result of the post-quantum public key and post-quantum signature being added (Table 2.2).

2.3.5. Related Work

Mota [Mot16] and Ackermann [Ack19] compared CMP, CMC, SCEP and EST regarding their suitability for different network settings: Mota [Mot16] investigated the deployment of a PKI for IoT devices for the authentication of web services. He proposed the use of EST since it allows server-side private key generation for resource constrained devices. Ackermann [Ack19] worked on the integration of an automated certificate management system for OPC Unified Architecture (OPC UA), a cross-platform standard data exchange in industrial networks developed by the OPC Foundation.

In the last years several proprietary solutions emerged for cloud-based automated certificate life cycle management [Glo22; Key22b; MTG22]. Furthermore, in the research project *FieldPKI* a related subject is studied, namely the security life cycle management of heterogeneous industrial field bus environments [Off21].

2.4. Computer-Aided Security Protocol Verification

Security protocols are used to secure communication over insecure networks by utilizing cryptographic primitives. The design of security protocols, however, is particularly prone to errors. One of the most prominent examples in literature is the well-known Needham-Schroeder public-key protocol [NS78], in which 16 years after its publication Lowe [Low96] discovered a security error. Despite significant progress since then, many errors in current security protocols remain [Bla12]. Such errors on protocol level not only weaken the trust in security measures but also are not detectable by functional software testing because they appear only when a malicious adversary is present. In order to obtain actual assurances that a security protocol is correct the use of automatic verification tools can be helpful. Therefore, security protocol verification has been and continues to be a very active research area since the 1990s [Cre08; EMM09; Mei+13; Bla12; Bla16; KNT20].

Here, the foundations of security protocol verification are described and existing automatic tools are reviewed.

2.4.1. Security Models

A formal security analysis of a protocol comprises two parts, the definition of a security model and the proof of the desired security properties in this model. As described by Boyd, Mathuria, and Stebila [BMS20, Section 1.6] formal methods are divided in two different categories depending on the underlying model: the symbolic methods and the computational methods.

Computational Model Typically, the computational model is used to prove the security of cryptographic primitives by reducing the protocol to security properties of its underlying primitives all the way to mathematical complexity assumptions. For this, messages are represented as bit strings and primitives are functions operating on those bit strings. It is shown that if a security property of the protocol can be broken in this model with non-negligible probability, then the existence of an adversary breaking one of the protocol's security assumptions is implied.

Symbolic Model In the symbolic model, also referred to as the *Dolev-Yao model* [DY83] the attacker is modeled as a non-deterministic state machine and the cryptographic primitives are symbolic functions, i.e., it is assumed that the attacker cannot break their security properties (*perfect cryptography*). Having only a fixed set of actions at their disposal the general idea is to show that the attacker is not able to reach a bad state, representing a successful attack.

As the Dolev-Yao model is classically assumed for network security protocols [BMS20, Section 1.6], in this thesis the security verification is conducted in the symbolic model. Previous works have shown that many practical attacks can be described in the symbolic model and tool-based analysis of complex protocols, such as TLS 1.3, has been proven to be quite valuable [Cre+17; LZK20].

2.4.2. Authenticity as Security Property

Security protocols can have various security goals, which can be broadly categorized as trace properties and equivalence properties. The definitions and formalisms used in this thesis are adopted from a survey by Blanchet [Bla12] and the formalization of "authenticity" by Lowe [Low97].

Definition 2.4.1 (Trace properties)**.** *Trace properties are properties that are defined for each run of the protocol (execution trace). A protocol satistisfies a trace property in the symbolic model if it holds for all traces and in the computational model for all but a set of traces with a neglible probability. An example for a trace property is the fact that certain states are not reachable.*

Definition 2.4.2 (Equivalence or indistinguishability properties)**.** *Equivalence properties are processes which an adversary cannot distinguish from each other, e. g., the protocol under study and its specification. In the symbolic model this notion is referred to as* process equivalence*, whereas in the computational model it is called indistinguishability. Equivalences are useful in modeling subtle security properties and can be used for compositional proofs.*

Automating a proof of an equivalence property is more challenging than automating the proof of trace properties because they cannot be described within a single trace, they rather require relations between traces (or processes).

The security goal "authenticity", which is the focus of this work, can be expressed as trace property and thus, the proof for the chosen certificate management protocol is likely to be automated by a verification tool.

By the notion "authenticity" we mean, that if a participant A runs the protocol seemingly with a participant B, then B runs the procedure supposedly with A, and vice versa. In general, it is also required that A and B have the identical protocol parameter values.

In 1997, Lowe [Low97] introduced a hierarchy of authentication properties for the symbolic model which was later extended by Cremers [Cre08]. Lauser, Zelle, and Krauß [LZK20] informally summarized the different meanings of authentication:

Definition 2.4.3 (Authentication properties)**.** *Let there be a protocol that aims to authenticate a* responder *or* prover *B to an* initiator *or* verifier *A. Then the following security properties directly adopted from Lauser, Zelle, and Krauß [LZK20, Section 3.3.1] can be specified:*

Aliveness *Whenever A completes a run of the protocol, apparently with B, then B has previously been running the protocol.*

Weak agreement *Whenever A completes a run of the protocol, apparently with B, then B has previously run the protocol, apparently with A.*

Non-injective agreement *Whenever A completes a run of the protocol, apparently with B and some data values $\overrightarrow{v}$, then B has previously been running the protocol, apparently with A and $\overrightarrow{v}$.*

Injective agreement *In addition to the non-injective agreement, each run of A has to correspond to a unique run of B. This implies* replay protection.

Non-injective synchronization *This property extends the non-injective agreement by additionally requiring that the order of messages is preserved, i. e., they are received in the same order as sent.*

Injective synchronization *In addition to non-injective synchronization, it is required that each run of A corresponds to a unique run of B.*

Additionally, these properties can be extended with a notion of *recentness*, which can be accomplished, for example, by tying messages to a timestamp that is only valid for a specified interval. The appropriate properties desired to prove are use case specific and thus, need to be chosen based on the protocol under study.

Cremers and Mauw [CM12] provide formal definitions of the aforementioned properties, but for the understanding of this thesis the informal descriptions are sufficient.

In Chapter 4, we will revisit the definitions here for the security analysis of the certificate management protocol.

2.4.3. Comparing Existing Verification Tools in the Symbolic Model

Table 2.4 provides a comparison of existing tools for symbolic verification of security protocols which support the analysis of trace properties, in particular, authenticity. This list is not extensive, but rather an excerpt of surveys conducted by Barbosa et al. [Bar+21], Boyd, Mathuria, and Stebila [BMS20] and Kobeissi, Nicolas, and Tiwari [KNT20]. The comparison criteria are established by Barbosa et al. [Bar+21]:

Automated proof-finding (Auto) Can the tool automatically conduct the security analysis? Interactive tools require in-depth knowledge about the verification techniques and thus, make the usage of such tools difficult for beginners.

Unbounded number of session (Unbound) Can the tool analyze an unbounded number of protocol sessions? Bounded tools (○) have an explicitly limited number of sessions to be analyzed, and attacks beyond this cut-off are not considered. Unbounded tools (●) are able to prove the absence of attacks within the model, this comes at the cost of undecidability.

Trace properties (Trace) Can the tool verify trace properties? As determined in Section 2.4.2, authentication is described as trace property in the symbolic model. Therefore, the tool should support the verification of trace properties.

Equational theories (Eq-thy) What is the support for equational theories? Equations enable detecting a larger class of attacks, since they allow more detailed modeling of cryptographic constructions, e. g., Diffie-Hellman constructions.

Globaly mutable states (State) Does the tool support verification of protocols with global mutable state? Tools with mutable states of the protocol participants

enable reasoning support for analyzing complex, stateful attacks which is practical for many real-world protocols use shared databases or shared memory.

Link to implementation (Link) Can the tool extract/generate executable code from specifications in order to link symbolic security guarantees to implementations? This is not necessarily needed for the scope of this thesis, but might be of advantage in future work towards the security analysis of the protocol implementation.

In Table 2.4, a filled circle (●) means that the criteria is fulfilled to full extent and an empty circle (○) means that it is not fulfilled. A half filled circle (◐) indicates that the criteria is only fulfilled to a limited extent or under certain assumptions. For our purposes we desire a tool that is able to run an automated security analysis, supports unbounded executions, can verify trace properties, in particular authentication properties (Section 2.4.2) and has mutable states. Apart from these criteria, we desire the tool to be accessible to protocol designers without deep knowledge of cryptographic proofs and reasoning.

In direct comparison with the other tools, ProVerif [Bla16], Tamarin [Mei+13] and Verifpal [KNT20] qualify most under these criteria. In our selection process the automation of the tool has the highest priority, since we pursue the use of an easily understandable tool. Therefore, we see the need for interaction during the security analysis in Tamarin as a disadvantage, although its support for equational theories is more refined [KNT20]. Verifpal is a tool heavily inspired by ProVerif, but instead of using applied pi-calculus as modeling language [ABF17], it introduces its own more intuitive language. Different from other existing tools Verifpal focuses on the usability for engineers and practitioners. That is why it has primarily been chosen in this thesis to conduct the security verification. ProVerif is known to have a significant problem with analysis termination, whereas Verifpal makes compromises in analysis completeness to greatly increase the likelihood of termination. Another common problem among protocol verifiers in the symbolic model is that for complex protocols, the space of participants states and value combinations the verifier needs to assess may become too large to terminate the verification within a reasonable time. Verifpal mitigates this problem by using certain heuristic techniques and leveraging multi-threading [KNT20]. These circumstances need to be considered when discussing the soundness of the verification results in Chapter 4, as Kobeissi, Nicolas, and Tiwari [KNT20] state that formally, Verifpal is unable to guarantee "that it never misses an attack in any model that can be expressed within its language".

2.4.4. Cryptographic Protocol Analysis with Verifpal

To conduct a security analysis with Verifpal, the attacker, the protocol, and its participants as well as the desired security goals need to be described in the Verifpal modeling language. In this section, the relevant principles, and notations as specified by [KNT20] are introduced for later use in Chapter 4.

Figure 2.3 shows a high level overview of all Verifpal language components used in this work: First, it is specified whether the model will be examined under a *passive* or

Tool	Auto	Unbound	Trace	Eq-thy	State	Link
Desired properties for tool	●	●	●	●	●	○
Maude-NPA [EMM09]	○	●	●	●	○	○
ProVerif [Bla16]	●	●	●	◐	○	○
Scyther [Cre08]	●	●	●	○	○	○
Tamarin [Mei+13]	◐	●	●	●	●	○
VerifPal [KNT20]	●	●	●	◐	●	◐

Table 2.4.: List of tools for symbolic protocol analysis

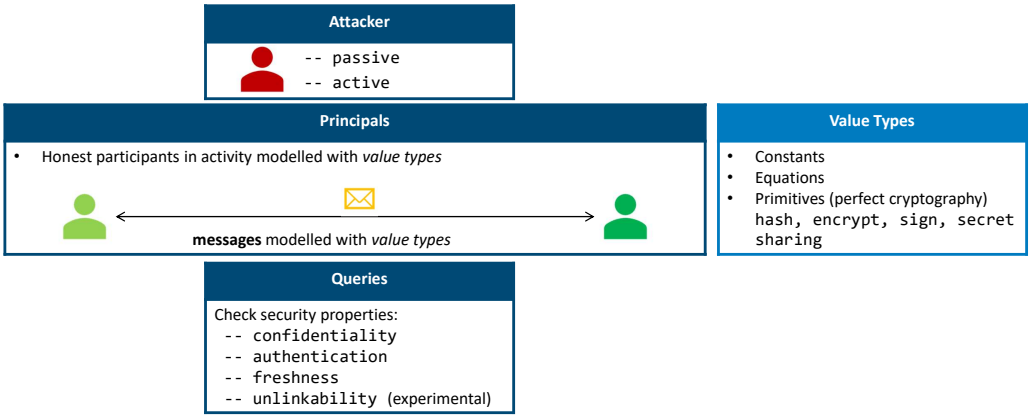


Figure 2.3.: Overview of Verifpal language components

active attacker. The attacker in Verifpal is understood as a Dolev-Yao attacker [DY83] (see Section 2.4.1). Informally, a passive attacker is only able to record transcripts of a protocol, i. e., in a real-world scenario they listen to all traffic that is transmitted over a network, and try to obtain as much information about the ongoing communication as possible, for example, to impersonate one of the communication partners. An active attacker is additionally able to modify an message as they cross the network.

Next, the honest parties participating in the protocol are defined, the *principals*. The internal behavior of the principals is described using a fixed set of *value types*. Such types include constants, equations and most importantly the cryptographic primitives which are assumed to be *perfect* as explained in Section 2.4.1. Afterwards, is it specified which messages are exchanged between the principals in order to perform the protocol activities. Finally, in *queries* it is expressed which security properties shall be analyzed by the tool. Verifpal offers four different types of queries: confidentiality, authentication, freshness, unlinkability; in this work we primarily focus on authentication and freshness.

In the following, a simple Verifpal model as shown in Figure 2.4 will be used to introduce the syntax in more detail. The principals Alice and Bob are communicating with each other. Alice first defines the constants for a secret key `a_sk`, and messages `x`, `y`. Constants in Verifpal are immutable, share a global namespace and can only be references in primitives. A constant declared with `knows` (`private` | `public`) will stay

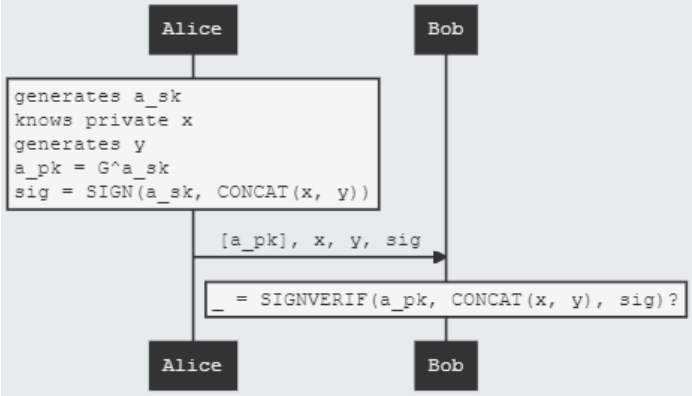


Figure 2.4.: A complete example Verifpal model of a simple protocol

the same across every execution of the protocol, whereas `generates` creates unique "fresh" values for every time the protocol is executed. So, while `a_sk` and `y` are fresh values, `x` stays fixed to one value. Next, she generates a public key `a_pk` associated with the secret key `a_sk`. The notation `G^a` resembles the Diffie-Hellman pubic key calculation, but semantically any type of public key generation is represented in this way. Afterwards, she signs the values `x`, `y` using the primitives `CONCAT` and `SIGN`. Verifpal supports set of core primitives such as `CONCAT` and cryptographic primitives for hashing, encryption, signatures and secret sharing. From the latter, we only require the signature primitives.

In the next step, Alice sends the public key `a_pk` and the message `x`, `y` to Bob. As `a_pk` is surrounded by brackets ([ ]), it becomes a *guarded* constant, meaning while an active attacker can read it, they cannot tamper with it.

Upon receiving Bob validates the message using `SIGNVERIF`. The question mark (?) behind `SIGNVERIF` makes the instantiation of the primitive *checked* That means that the protocol execution will be aborted at this point if the verification fails. We use _ to indicate that the return value of the verification, the message, will not be used afterwards.

In Table 2.5 the keywords and notations used in this thesis are summarized.

Notation	Description
`G^a : ga`	Calculates a public key `ga`
`knows [private | public] a`	Prior knowledge of a constant
`generates a`	Creates fresh value
`CONCAT(a, b ...) : c`	Concatenates between two to five values into one value
`SPLIT(CONCAT(a, b ...)) : a, b`	Splits a concatenation back to its component values
`SIGN(key, m) : sig`	Signature primitive with private key `key` and message `m`
`SIGNVERIF(G^k, msg, SIGN(k, m)) : m`	Verifies if signature can be authenticated
`[a]`	Guarded constant, active attacker cannot tamper with it
`SIGNVVERIF(...) ?`	Checked primitive
_	Used here to indicate that return value is ignored

Table 2.5.: Keywords and notations in Verifpal

3. Protocol Overview of Selected Lightweight CMP Profile

In Section 2.3.3, the Lightweight CMP Profile [BOF23] has been introduced as a tailored version of CMP to meet the needs of automated certificate management for IIoT. However, the protocol draft remains a framework which requires further profiling to address a specific use case or scenario. Therefore, we propose a variant of the protocol based on the request for comments (RFC) draft [BOF23] which aims for a lightweight implementation for end entities and takes the use of post-quantum certificates into account. The end entity is expected to have only limited resources, whereas it can be assumed that the PKI management entities have larger resources at their disposal. We focus on the three core certificate management operations as identified in Section 2.3.2: certificate request, renewal, and revocation.

This chapter is structured as follows: First, the requirements for industrial use cases are briefly described in Section 3.1. Then, the PKI architecture for this thesis is presented in Section 3.2 and the general aspects of CMP messages are elaborated in Section 3.3. Finally, the supported certificate management operations are explained in Section 3.4.

Here, only those details of the protocol specification are discussed, which are relevant for the remainder of the thesis. Further information, particularly the specific definitions of the message field types, can be found in the related standards for CMP [Ada+05; BOG22], Certificate Request Message Format (CRMF) [Sch05], and CMS [Hou09; Hou20]. The terminology used in this chapter corresponds to the definitions introduced in Section 2.3.1.

3.1. Requirements for Industrial and IoT Scenarios

In Section 1.1.1 and Section 2.1 the increasing need for security and availability in industrial networks has been elaborated. Especially, when relying on certificate based machine-to-machine communication (Section 2.1.2.2), it becomes crucial that the certificate management system is constantly available and cost-efficient. Therefore, high automation and reliability are required. Consequently, the best practice standard for industrial communication networks IEC 62443 requires in part IEC 62443-3-3 [Int13] for security level 2 and higher that “proper processes for issuance, management, verification, revocation, and audit for authorized devices, users, and processes involving identity and credential management ” [BOF23] are in place. According to International Electrotechnical Commission [Int09], security level 2 refers to systems which are

required to have protection measures against intentional misuse by simple means with few resources, general skills and low motivation.

The concept of security zones, introduced in Section 2.1.1, poses an additional challenge for certificate management systems. Typically, the PKI management entity is not deployed in the production zone, but rather located in a separate network either on-site or even off-site in a highly protected data center environment. Moreover, the production network is often heterogeneous in their use of network technologies. To cope with such network architectures, CMP uses authenticated self-contained messages which enable flexibility regarding the message transfer.

3.2. Revised Architecture

Different from the example architecture provided by Brockhaus, Oheimb, and Fries [BOF23] in our target PKI architecture an intermediate CA or `Sub_CA` directly communicates with the `End_Entity` to perform the certificate management operations, as shown in Figure 3.1. For this, the tasks of the RA as defined in Section 2.3.1 are taken over by the CA itself. The `Sub_CA` possesses its own identification certificate *ca_cert* and is authorized by a private `Root_CA` to issue certificates for IoT devices in the production network.

To enable secure automatic certificate enrollment, we assume that the IoT device (here: referred to as `End_Entity`) possesses a device certificate *dev_cert*, which has been issued by the `Root_CA`. It is assumed, that during the provisioning of the `End_Entity` in the production environment this certificate is installed on the device together with its trust chain. The *dev_cert* is intended to identify the `End_Entity` throughout its lifespan. As the certificate chain is typically needed to be transmitted during the CMP operations, and the post-quantum certificates are going to be considerably larger, a PKI architecture with a flat hierarchy is proposed here.

Here, the automated certificate management is realized for the `End_Entity` which uses the Lightweight CMP Profile to request application certificates from the `Sub_CA`. These application certificates are intended to be used for secure communication protocols as described in Section 2.1.2.2. Therefore, we limit the key usage of the issued certificates to signature certificates that facilitate the authentication of machine-to-machine communication. The communication between the `Root_CA` and the other PKI entities is out of scope.

In accordance with the pull model, all certificate management activities described in this chapter are initiated by the `End_Entity`. They generate the private-public key pair locally, create a CMP request message message protected with a digital signature and send it to the `Sub_CA`. Upon reception, the `Sub_CA` verifies and handles the request. This interaction is performed synchronously, i. e., a request is processed and responded to immediately after it has been received.

Alternative approaches with CMP support asynchronous communication as well as solutions for more centralized certificate management. In case of asynchronous communication, the `End_Entity` can poll for delayed messages ([BOF23, Section 5.1.7]). The centralization can be achieved through the CA centrally generating all

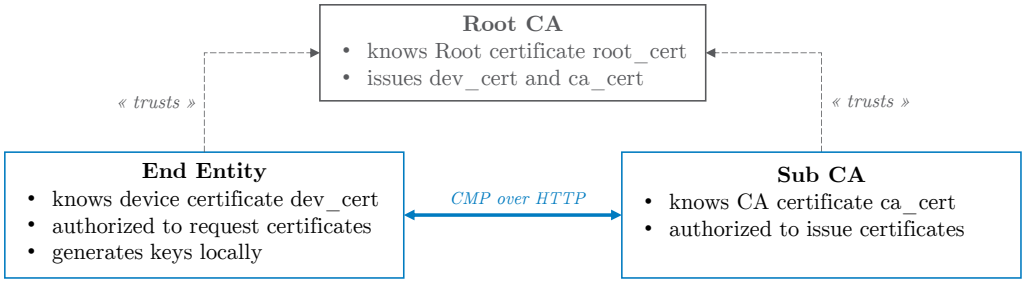


Figure 3.1.: Overview of proposed PKI architecture

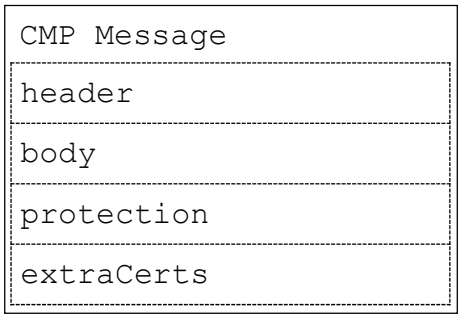


Figure 3.2.: CMP message structure

keys for the end entities and distributing them securely in a CMP message ([BOF23, Section 4.1.6]). Additionally, Brockhaus, Oheimb, and Fries [BOF23] state that the CMP messages can also be implemented in a way that the push model is supported. For this, the CA would proactively trigger the `End_Entity` to provide a CMP request making the `End_Entity` the receiver of the interaction rather than the initiator. As the certificate management in our scenario is used to provide certificates for applications on the control devices, the more lightweight solution is letting the `End_Entity` initiate the management operations – thus, implementing a push model. In the pull model, additional communication with the PKI management entity would be required to trigger the certificate issuance after the installation of an application.

Brockhaus, Oheimb, and Fries [BOF23] specify multiple methods how CMP messages can be exchanged: Either using HTTP, Constrained Application Protocol (CoAP) or piggybacking over any other reliable protocol, or transferring the message offline, e. g., via file transfer. The protocol is very flexible regarding the message transfer method because the protection of the CMP messages is self-contained. In our scenario, we assume message transfer over HTTP as in the industrial network infrastructure, it is already used for machine-to-machine communication (see: Section 2.1.2). So, under this condition, additional provisioning of a transport protocol can be saved with this approach.

pvno	sender	recipient
messageTime		protectionAlg
senderKID		transactionID
senderNonce		recipNonce
generalInfo:implicitConfirm		

Figure 3.3.: CMP message header structure

3.3. Generic Aspects of CMP Messages

All CMP messages follow the same structure: They comprise of a message header and body, a protection and an extraCerts field as depicted in Figure 3.2. The *header* contains meta information about the CMP protocol transaction. The *body* contains information necessary for the actual certificate management operations. The *protection* field is a signature-based message protection and *extraCerts* is a generic field to convey certificates.

In the following, all fields are described in more detail, except for the message body. The contents of the latter are specific to the PKI management operation and will be presented below.

3.3.1. CMP Message Header

Figure 3.3 shows the CMP message header structure.

The *pvno* field contains the protocol version number, which is CMP v2 for the management operations supported in this variant of the protocol.

The *sender* field contains the name of the message originator. The name must be identical to the subject of the certificate which is used to calculate the message protection, the CMP protection certificate. The *recipient* field should contain the name of the message recipient. Since our profile uses the pull model, the recipient of the first message of CMP management operation (or transaction) is a PKI management entity. Therefore, the *recipient* field should contain the subject distinguished name of the CA. In the following messages of the same transaction, it has value of the *sender* field of the previous message. In case of deviations from this specification, the field shall be handled gracefully, i. e., a different use does not automatically result in an error.

For the signature-based protection the *protectionAlg* field contains the algorithm identifier for the signature scheme and the *senderKID* field contains the subject key identifier of the CMP protection certificate used to calculate the message protection signature.

The *messageTime* is the time stamp of the time when the message was produced. Although it is an optional field it is typically used in implementations to verify the recentness of a message.

The *transactionID* is a 128 bit random value used as unique identifier of a transaction. It is newly generated for the first message of the transaction and will be kept the same for all following messages of the same transaction.

The *senderNonce* and *recipNonce* contain 128 cryptographically secure and fresh random bits. In the first message of a transaction the *recipNonce* must be absent; in all other messages, the *senderNonce* of the previously received message in this transaction becomes the value of the *recipNonce*.

The last field of the header is the *generalInfo*, which can contain different optional values. Amongst those is the *implicitConfirm* flag. In CMP, the actual management operation is communicated via a simple request-response message pair. But by default, the CA additionally requires confirmation that the end entity received the issued certificate and afterwards sends a confirmation message back to the end entity. Depending on the PKI policy and requirements for handling end entity certificates, it may be necessary for PKI management entities to learn whether the new certificate was accepted by the end entity. For our purposes, this information is not significant. Hence, this overhead of a further message round-trip is saved by setting the *implicitConfirm* flag when enrolling a new end entity and updating a valid certificate. In case of revocation requests the default message exchange already comprises of a singular request-response interaction. Therefore, here this flag must not be used.

3.3.2. CMP Message Protection

The CMP message protection serves for the message origin authentication and integrity protection for the CMP header and body. The *extraCerts* field is not covered by this protection. The default mechanism for the message protection is a digital signature. Alternatively, MAC-based protection can be used which requires shared secret information. A MAC is a piece of information that is calculated over the message using symmetric cryptography and a shared secret to ensure the message integrity. In particular, a MAC prevents an attacker from modifying a message or injecting a new message, without the receiver detecting that the original message was tampered with [KL15a, Section 4.2]. The recent internet draft by Brockhaus et al. [Bro+23] which is intended to obsolete RFC 4210 [Ada+05], proposes three different ways to establish the shared secret for the message protection: MAC with Pre-Shared Secret (MAC-PSS), MAC with Diffie-Hellman Key Agreement (MAC-DH) and MAC with Key Encapsulation Mechanism (MAC-KEM).

Especially, regarding the migration to post-quantum cryptography, KEMs are a viable alternative to signature-based message protection: With a KEM the symmetric key material for the MAC calculation is secured for transmission using asymmetric algorithms. Here, this option is not further investigated because it requires an additional round trip to complete a CMP management operation and thus, causes a higher communication overhead.

We use a signature-based message protection, where the entity protecting the message has to calculate the signature using the CMP protection key. As mentioned in Section 3.3.1, the *protectionAlg* field in the header indicates the used protection

CMP message type	Description
`ir`	Initial request message
`ip`	Response to `ir` message
`cr`	Certificate request message
`cp`	Response to `cr` message
`kur`	Key update request message (certificate renewal)
`kup`	Response to `kur` message
`rr`	Certificate revocation request message
`rp`	Response to `rr` message

Table 3.1.: Feature summary of selected Lightweight CMP Profile

algorithm, i. e., for signature-based protection it is required to contain an algorithm identifier for a signature.

The CMP protection certificate corresponding to the protection key together with its certificate chain is added to the *extraCerts* field of the CMP message so that the receiver is able to verify the message protection. In general, the *extraCerts* field is a generic field to convey any additional certificates which might be necessary to perform a PKI management operation. In the next section it will be further specified if other certificates other than the CMP protection certificate are contained in this field.

3.4. Supported PKI Management Operations

In this section, the procedures and message structures of the three main PKI management operations are specified. First, the transaction for enrolling an end entity to the PKI is described in Section 3.4.1. Afterwards, the certificate renewal operation follows in Section 3.4.2. Finally, in Section 3.4.3 the certificate revocation operation is explained. For the sake of readability, the `Sub_CA` will be referred to simply as CA in this section.

Table 3.1 provides an overview of the CMP message types of the operations supported by this profiled version of the protocol.

3.4.1. Enrolling an End Entity

For a certificate request the end entity generates a new public-private key pair which it intends to use for authentication purposes in some application. Then they can trigger the enrollment process to the PKI by sending a request to the CA.

In this request, the end entity has to authenticate themselves as well as provide proof that they possess the private key associated with the certificate template presented to the CA. As our solution architecture focuses on digital signature certificates (Section 3.2), we can use a certificate-based proof-of-possession (POPO) in the form of a self-signature. In Section 3.3.2, it has already been established that this specification uses signature-based message protection and hence, signature-based authentication.

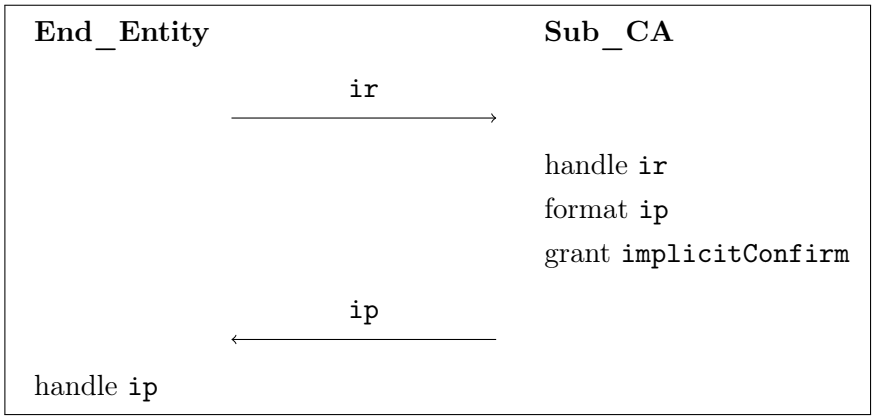


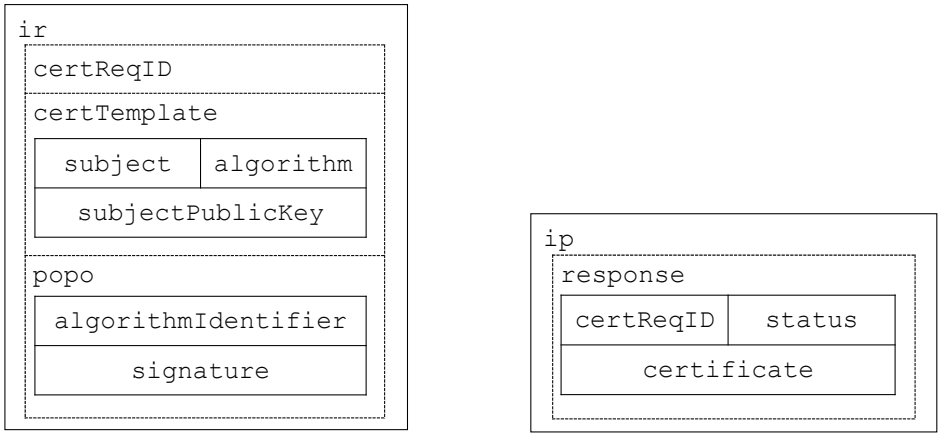
Figure 3.4.: Message flow for enrolling an end entity to a new PKI

However, there are different approaches in the way the end entity authenticates themselves to the PKI depending on whether it is their first request addressed to the CA or a subsequent one. Therefore, in the following the transaction for both scenarios are explained in more detail.

Enrolling an End Entity to a New PKI For the first transaction with the CA the end entity creates an initial request message *ir* as show in Figure 3.5a. The actual certificate request comprises of a *certReqId* field which must be 0 and a certificate template *certTemplate*. To create this template, the end entity generates a new public-private key pair and adds the public key in the *subjectPublicKey* field. The *algorithm* field contains the identifier of the signature scheme associated with the generated key pair. The *subject* field holds the name of the end entity, it will later become the subject of the issued certificate. As mentioned above, the end entity has to produce a proof for owning the public key in the certificate template. For this, they sign the template with the corresponding private key. Thus, the value in the *algorithmIdentifier* field of the *popo* container is identical to the *algorithm* field. The calculated signature is put in the *signature* field of the *popo*. With this the construction of the *ir* is completed.

Now the end entity needs to authenticate themselves towards the CA by using signature-based protection as described in Section 3.3.2. The CMP protection certificate used for the initial request is the device certificate which has been introduced in our architecture specification in Section 3.2. The private key associated with the device certificate is used to sign the CMP header and body containing the *ir*. The device certificate and its chain are placed in the *extraCerts* field. With this, the CMP message for the initial request is complete and sent to the CA.

Upon reception the CA verifies the authenticity of the end entity by validating the certificate and its chain received in the *extraCerts* field. Then they verify the message protection with this certificate. Now, the certificate request is processed and the CA issues a certificate for the provided public key and signs it with the private key of their CA certificate. In Section 3.3.1 is has been specified that the *implicitConfirm*



(a) initial request (b) initial response

Figure 3.5.: CMP message body structure for certificate requests

flag is set in the CMP header. Therefore, internally, the CA is configured to grant the $implicitConfirm$.

The response message *ip* (Figure 3.5b) contains the $certReqID$ value from the request message and a *status* value which indicates that the request has been accepted if positive, or rejected if negative. If accepted, the signed X.509 certificate is placed in the $certificate$ field. The CA calculates the message protection using the private key of their CA certificate. This certificate is then added to the $extraCerts$ container.

The end entity can validate the authenticity of the response message using the CA certificate, and then finally, retrieve the issued certificate from the message. The message flow for the initial certificate request is depicted in Figure 3.4.

Enrolling an End Entity to a Known PKI After the initial enrollment the end entity already possesses a certificate which has been issued by the CA. Therefore, they do not have to use their device certificate to authenticate themselves, but rather use the already issued certificate. For any additional certificates, e. g., for other applications, the end entity sends a certificate request to the CA and if granted, they will receive a new signed certificate as response as depicted in Figure 3.6. For this, the end entity generates a new public-private key pair for the $certTemplate$, calculates the *popo* with this new private key, and signs the CMP header and body with the private key of the already existing certificate.

Consequently, the message structures for requesting an additional certificate is identical to that shown in Figure 3.5 with one minor change: The body of the request must be of type *cr* and the response of type *cp*.

3.4.2. Updating a Valid Certificate

For a certificate renewal the end entity sends a key update request to the CA, and if granted, they receive a new signed certificate as response. It is required that the certificate to be updated cannot be expired yet or even be revoked at the time of the

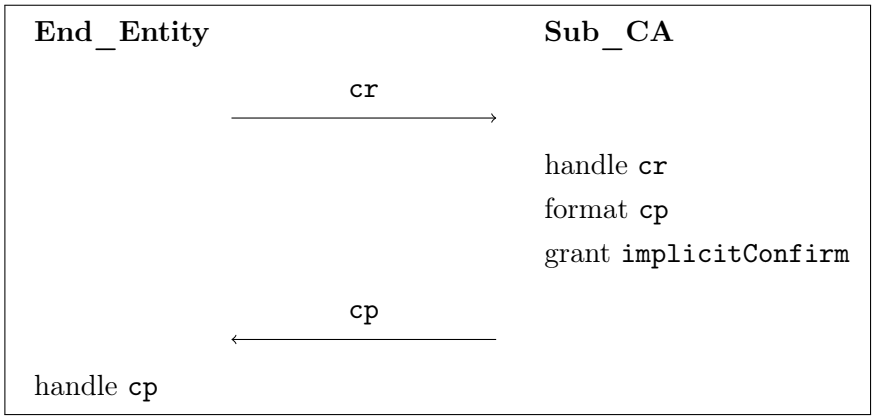


Figure 3.6.: Message flow for enrolling an end entity to a known PKI

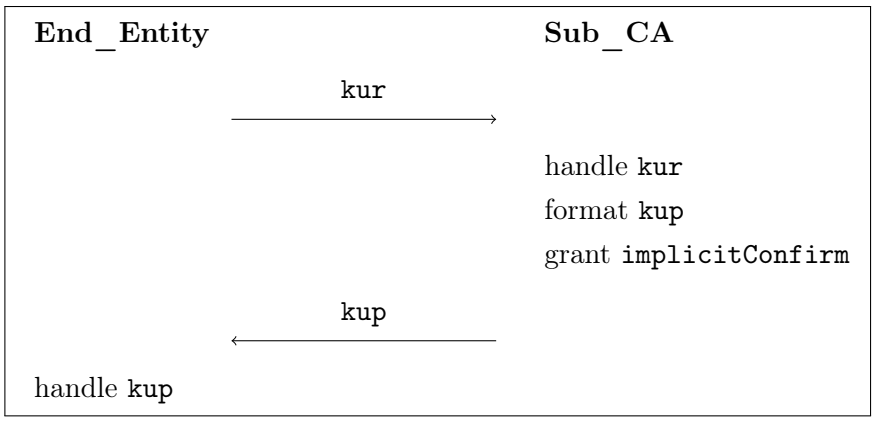


Figure 3.7.: Message flow for updating a valid certificate

request. Additionally, the new public-private key pair needs to be generated for the new certificate.

The message structures for updating a valid certificate is identical to that shown in Figure 3.5 with some minor changes: The body of the request must be of type *kur* and the response of type *kup*. The *subject* field must be identical to the end entity subject name used in the currently used certificate. The end entity uses the existing certificate they wish to update as CMP protection certificate. By signing the message with its corresponding private key, they both authenticate themselves and proof ownership of the certificate to be updated.

Additionally, after the *certTemplate* field optionally the *oldCertId* control, comprising of the *issuer* and *serialNumber* of the current certificate, can be added to clarify which certificate is to be updated. In our profile we do not use this field, since this information can be obtained from the CMP protection certificate which will be transmitted in the *extraCerts* field in any case.

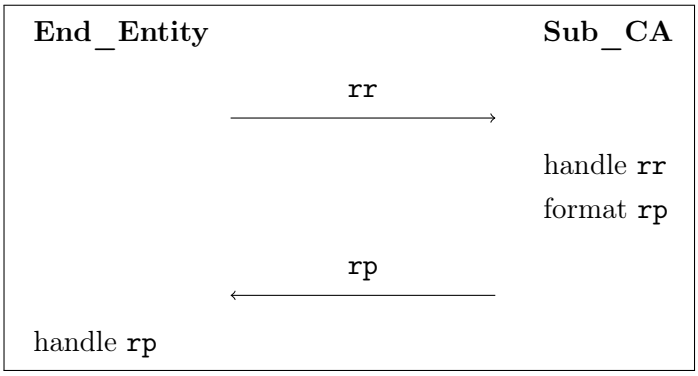
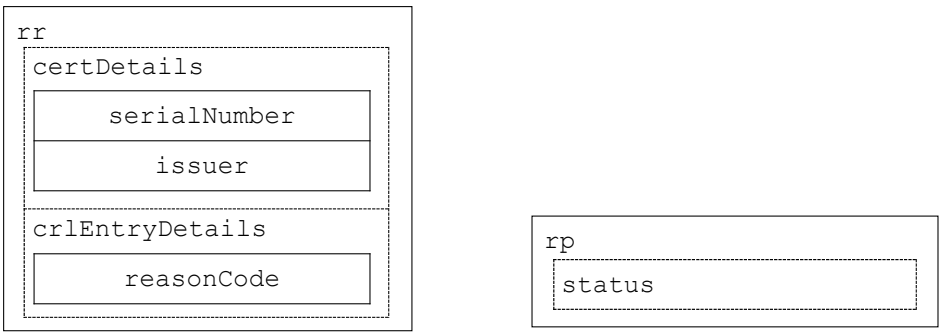


Figure 3.8.: Message flow for revoking a certificate



(a) revocation request (b) revocation response

Figure 3.9.: CMP message body structure for certificate revocation

3.4.3. Revoking a Certificate

To revoke a certificate the end entity issues a revocation request *rr* as shown in Figure 3.8 using the message structure illustrated in Figure 3.9. For this, the certificate which shall be revoked cannot be expired yet or already be revoked. The revocation request uses the CMP header specification as described in Section 3.3.1. The message body contains the details of the certificate to be revoked: its serial number and the name of the issuer. Additionally, in the *crlEntryDetails* field the reason for the revocation request is stated using a *reasonCode*. As message protection the CMP header and body must be signed with the certificate that is to be revoked to prove that the end entity is authorized to perform this action.

An end entity can only request a certificate revocation from the CA that issued the certificate. Therefore, upon reception of the *rr*, the CA validates whether they issued the certificate and if it is still valid at that time. After successful validation the CA sends a response message which basically contains one field which indicates the *status* of the revocation. A positive value means the revocation request has been accepted, a negative value, respectively, means rejection.

Protocol property	Options			
Certificate purpose	Digital signature			
Communication approach	☒	Pull model	☐	Push model
	☒	Synchronous	☐	Asynchronous
Key generation	☒	Local	☐	Central
Message transfer	☒	HTTP	☐	CoAP
	☐	Piggybacking	☐	Offline
Supported operations	☒	ir	☒	cr
	☒	kur	☒	rr
Proof-of-possession	Self-signature			
Message protection	☒	Signature	☐	MAC-PSS
	☐	MAC-DH	☐	MAC-KEM
Confirmation	☒	implicitConfirm	☐	certConf

Table 3.2.: Feature summary of the selected Lightweight CMP Profile

3.5. Profile Summary

To summarize the design choices made in our instantiation of the Lightweight CMP Profile, in Table 3.2 all significant features are listed and the selected options are marked. With this variant of the RFC draft we present a minimal certificate management solution which can be operated without additional user interaction. Therefore, it is well suited for its deployment in IIoT scenarios.

4. Security Analysis of Lightweight CMP Profile using Verifpal

In this thesis, the security of the Lightweight CMP Profile is analyzed in the symbolic model (Section 2.4.1) using the verification tool Verifpal [KNT20]. For this, we use the specification of the protocol as described in Chapter 3. The Verifpal syntax used in this chapter is specified in Table 2.5.

Figure 4.1 visualizes how the analysis is structured in Verifpal: The protocol itself is transferred into different Verifpal models for each certificate management operation. For each model, first, the initial state of the principals is specified and then, the protocol activities of the CMP operation are specified.

The `End_Entity` is an application that wants to use a certificate and therefore, needs to request those from a PKI. Here, a simple PKI is modeled by a hierarchy of two CAs, an intermediate CA called `Sub_CA` and the root CA `Root_CA`. This corresponds to the setup described in Section 3.2. The communication between the `Sub_CA` and the `Root_CA` is put in brackets as it is not part of CMP in our scenario. Nevertheless, it is necessary to include it in the model in order to establish a common trust anchor.

These preliminaries will be described in more detail in Section 4.1. Additionally, in that section the assumptions regarding the security of the cryptographic primitives and the CMP protocol details are specified. Based on these assumptions and the query capabilities of Verifpal, the security goals for the Lightweight CMP Profile are determined in Section 4.2. In Section 4.3, the behavior of the Dolev-Yao attacker model in Verifpal's analysis methodology is described.

Section 4.4 then provides the Verifpal models for the protocol. To model CMP three different models have been specified: enrolling an end entity to a new PKI, updating a valid certificate, and revoking a certificate. From a cryptographic point of view the protocol operations for enrolling an entity to a known PKI is identical to enrolling to a new PKI, just the semantics of the message fields differ and therefore, modeling this case is omitted.

Finally, we describe the security goals as Verifpal *queries* in and present the verification results for the protocol models in Section 4.5.

4.1. Assumptions and Preliminaries

For the security analysis we make certain security assumptions as described in the following. Additionally, preliminaries and adaptations of the Lightweight CMP Profile, which are necessary to define the Verifpal models, are outlined here.

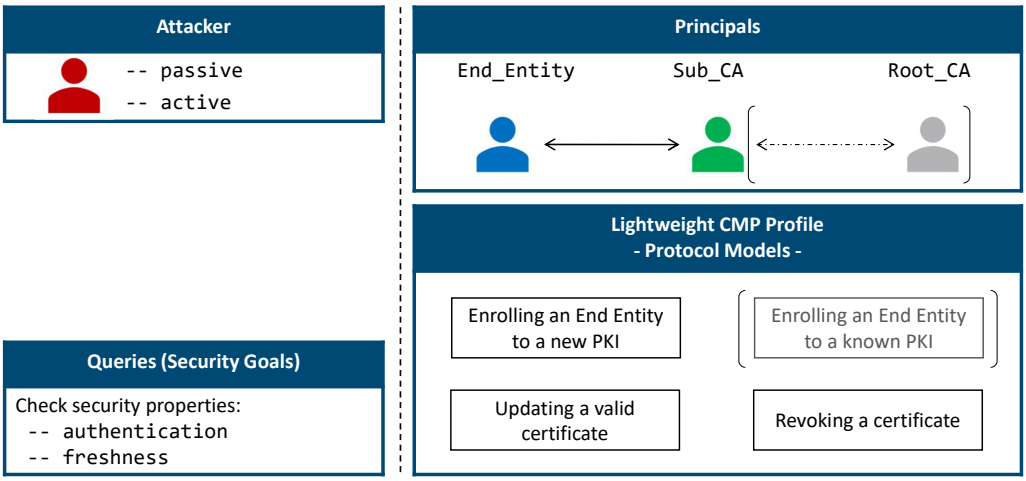


Figure 4.1.: Overview of Verifpal model for Lightweight CMP Profile

4.1.1. Security Assumptions

Verifpal conducts security proofs in the symbolic model and therefore assumes that all cryptographic primitives fulfill *perfect cryptography* (Section 2.4.1). Compared to this, the requirements we formulate for the certificate management protocol are weaker: The signature scheme used in the certificates and for authentication purposes shall be secure under the notion of EUF-CMA as defined in Section 2.2.1. Choosing Dilithium, the algorithm also has a security strength categorized as NIST security level 2, i. e., it is at least as hard to break as SHA-256 in terms of computational resources (Table 2.1).

Derived from the target architecture in Section 3.2, we assume that the `Root_CA` is already established as trust anchor in both the `End_Entity` and the `Sub_CA`. The *End_Entity* shall possess a *device certificate* signed by the `Root_CA` before the CMP protocol activities begin, and the `Sub_CA` shall possess a *CA certificate* signed by the `Root_CA`, respectively.

4.1.2. Certificate and CMP Message Representation

In Verifpal protocol messages can either be one simple value or a concatenation `CONCAT`$(a, b \dots)$ of up to five values. First attempts of defining a complete CMP protocol message by nesting multiple `CONCAT` statements have shown that it leads to state space explosion. This means that the space of value combinations that the verifier needs to assess becomes too large to terminate within a reasonable time. Therefore, only most crucial fields of the CMP message serving a cryptographic purpose are included in the models. These are in particular public keys, signatures and nonces. The same reasoning applies to the representation of a certificate: A signed certificate in Verifpal only comprises the public key and the signature itself. This simplification is a common approach among protocol verification in the symbolic model such as Cremers et al. [Cre+17].

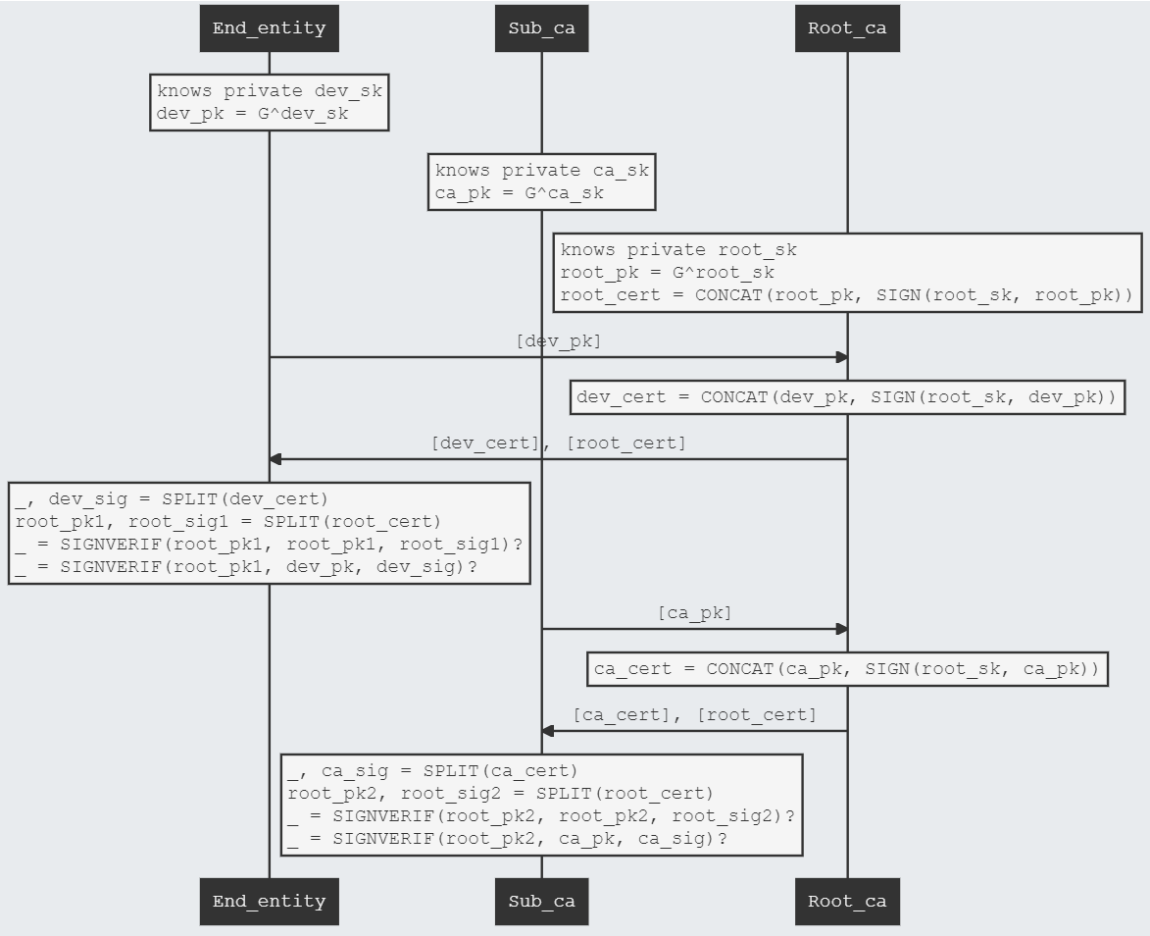


Figure 4.2.: Protocol diagram of trust anchor establishment model in Verifpal

4.1.3. Trust Anchor Establishment

The trust anchor is established in the real-world scenario with the out-of-band distribution of the root certificate, and the aforementioned device certificate and CA certificate. The correct way to translate this setup into the Verifpal model would require active interaction between the `End_Entity` / `Sub_CA` and the `Root_CA`. Figure 4.2 shows the protocol diagram of the Verifpal model in which the `Root_CA` transmits the signed device certificate (`dev_pk`, `dev_sig`) to the End Entity and the signed CA certificate (`ca_pk`, `ca_sig`) to the `Sub_CA`: All the transmitted values are modeled as *guarded constants* so that, while an active attacker can read it, they cannot tamper with it. This satisfies the desired properties towards the trust anchor establishment, since we are solely interested in attacks during the CMP protocol activity.

To reduce the complexity in the Verifpal models for the certificate management operations under investigation, this pre-protocol interaction is abstracted with the following assumption: The `End_Entity` and `Sub_CA` both own the public-private key pair of the `Root_CA` and create their own signed certificates locally, i. e., the CA certificate and the device certificate.

4.2. Security Goals

Our overall goal is to ensure the authenticity of the certificates which are issued through the certificate management protocol. In Section 2.4.2, more formal notions of authentication as a trace property have been introduced. Verifpal analyzes security properties by using a fixed set of queries, among those are *authentication queries* and *freshness queries* which are described as follows [KNT20]:

Authentication queries The purpose of an authentication query is to determine if Bob will depend on a particular value x in a crucial protocol activity (such as authenticated decryption or signature verification) if and only if he received that value from Alice. If x is successfully used by Bob for signature verification or a similar action without necessarily having been sent by Alice, the attacker has successfully impersonated Alice.

Freshness queries Freshness queries help identify replay attacks, in which an attacker modifies a message to appear valid in two different contexts. A freshness query in passive attacker mode will determine whether a message contains at least one generated, non-static value between sessions. In active attacker mode, it will determine whether a message may be successfully used between sessions and thus, rendered static.

The authentication query corresponds to the definition of *non-injective agreement* from Section 2.4.2. Together with the freshness query for detecting replay attacks, we get the notion of *injective agreement* from Section 2.4.2.

So, under the assumption in Section 4.1 , we aim to show that the certificate management operations of the Lightweight CMP Profile as specified in Chapter 3 fulfill the authentication property of *injective agreement*. In particular, the following properties will be proven by Verifpal:

SG 1 Injective agreement on a signed certificate for a newly enrolled end entity When enrolling an end entity to a new or known PKI, the `Sub_CA` can trust that the certificate template is freshly generated and sent by the `End_Entity`, and the `End_Entity` can trust that the corresponding signed certificate is sent by the `Sub_CA`.

SG 2 Injective agreement on a signed certificate after renewal When requesting a certificate renewal, the `Sub_CA` can trust that the certificate template is freshly generated and sent by the `End_Entity`, and the `End_Entity` can trust that the corresponding signed certificate is sent by the `Sub_CA`.

SG 3 Injective agreement on revoking a certificate When requesting a certificate revocation, the `Sub_CA` can trust that the certificate template containing a unique serial number is sent by the `End_Entity`, and the `End_Entity` can trust that the received revocation status information is sent by the `Sub_CA`.

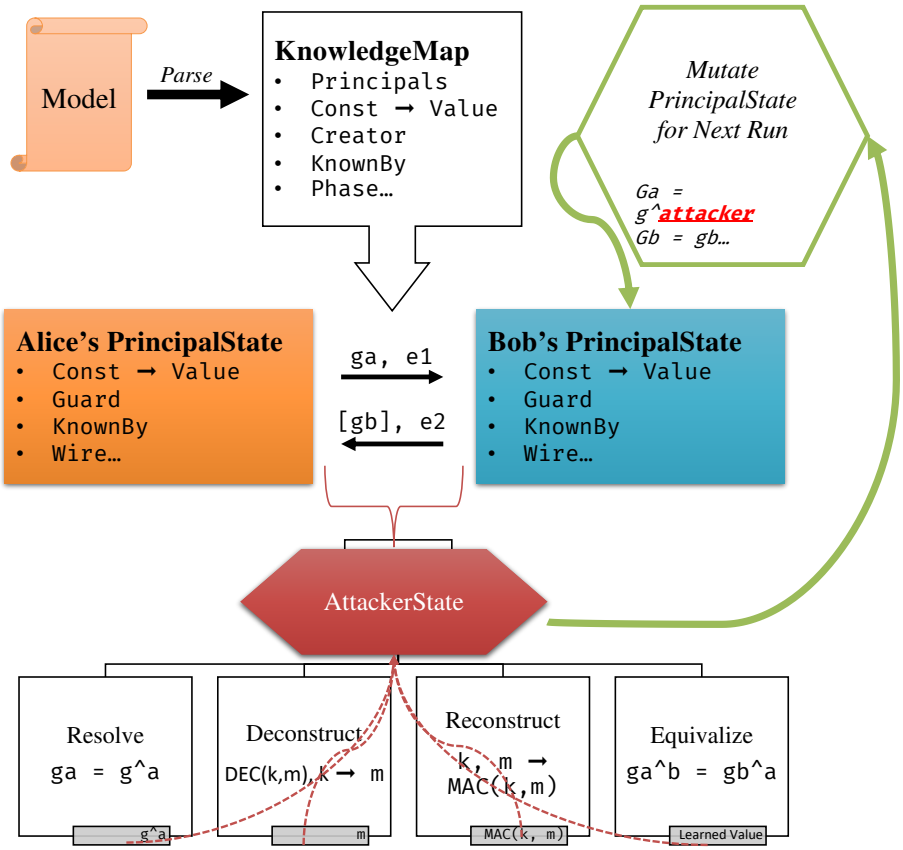


Figure 4.3.: Illustration of Verifpal analysis by Kobeissi, Nicolas, and Tiwari [KNT20]

4.3. Attacker Model

The Verifpal analysis is based on the Dolev-Yao attacker model as briefly introduced in Section 2.4.4. Based on the explanations by Kobeissi, Nicolas, and Tiwari [KNT20] regarding Verifpal's attacker analysis methodology, we derived an informal description of the attacker's procedures and capabilities.

Figure 4.3 shows the overall process of analyzing a Verifpal model. First, the model is translated into a global immutable *knowledge map* structure. From this the *prinicipal state* for each honest participant in the protocol is derived. As described in Section 2.4.3 these states are mutable. The attacker observes a protocol session and gathers all values sent between the principals. Then they apply four different types of transformations to the obtained values. Finally, they update the attacker's state with the newly learned values and start the next protocol execution with mutated principal states. After each step, the attacker verifies if they have found a contradiction to any query specified in the model.

As depicted in Figure 4.3 four main transformations are performed on the attacker's state, which contains the set of all values known by the attacker:

Resolve Resolving a certain constant to its assigned value.

Deconstruct Attempting to deconstruct a primitive or equation. For this, the attacker must obtain enough information from the protocol run to reserve-engineer the input parameters of a cryptographic operation. Then it can deconstruct the expression into a logical subset of its inputs.

Reconstruct Given that the attacker possesses all component values, attempting to reconstruct a primitive or equation.

Equivalize Determining whether the attacker can reconstruct or equivalize any values within a principal's state from the attacker's state. If successful, the equivalent values are added to the attacker's state.

The overall goal of the attacker is to obtain as many values as possible from their viewpoint on the network.

Passive attacker A passive attacker can only obtain values by deconstructing the values observed on the network during the protocol execution. They might potentially also be able to reconstruct these into different values.

Active attacker An active attacker can modify unguarded constants when they are sent over the network. Additionally, an active attacker can generate its own values such as a key pair which they have full control over. They can also create new values to substitute any unguarded constant exchanged between the principals. As each modification can result in learning new values, the attacker can make an unbounded number of modifications over an unbounded number of protocol executions.

4.4. Protocol Models

In the following it is described how the main certificate management operations of the Lightweight CMP Profile are translated into Verifpal models.

4.4.1. Enrolling End Entities

Figure 4.4 shows the protocol diagram for enrolling new end entities to the certificate authority by sending an *initial request* message which has been described in Section 3.4.1. For this, we declare two principals, the `Sub_CA` and `End_Entity`. As previously stated, both principals are assumed to know the private key of the `Root_CA`. Each principal then calculates the corresponding public key `G^root_sk`. Now the initial certificates are created: `Sub_CA` creates its own certificate by creating a public-private key pair and signing the public key using the `SIGN` primitive and the secret key of the root CA `root_sk`. So, the tuple (`ca_pk`, `ca_sig`) represents the `Sub_CA` certificate and (`dev_pk`, `dev_sig`) the device certificate of the `End_Entity`, respectively.

The actual CMP enrollment operation, as described in Section 3.4.1, begins at this point with the creation of the initial request message `ir` of the end entity: In the

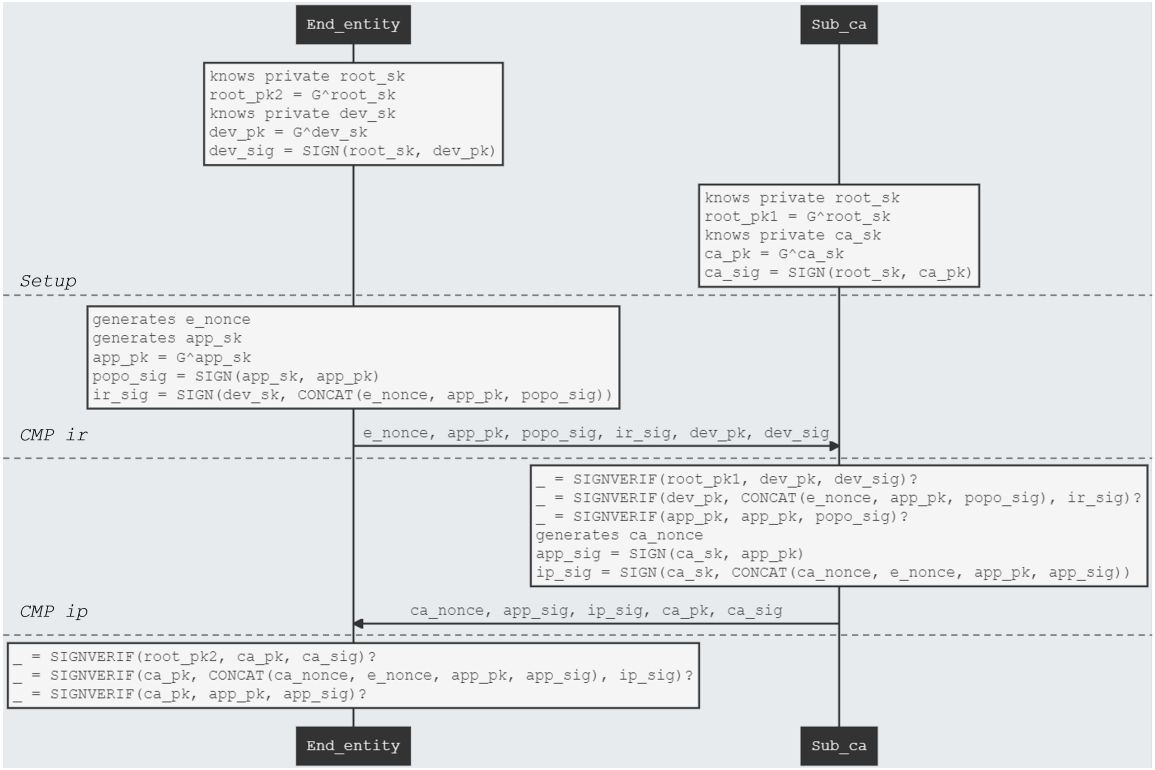


Figure 4.4.: Verifpal communication model of CMP initial request

CMP message structure	Verifpal message representation
header	`e_nonce`
body	`app_pk`, `popo_sig`
protection	`ir_sig`
extraCerts	`dev_pk`, `dev_sig`

Table 4.1.: Simplification of the CMP initial request message for Verifpal

CMP header most of the fields are static and used to identify the communication end points and message type. Since Verifpal does not support the equivalence test of non-cryptographic values, it is justifiable to neglect these fields in the model. The *sender nonce* field solely serves a cryptographic purpose enabling replay protection. For this, in each protocol run a new `e_nonce` is generated. Now, the CMP body shall contain the certificate template which the CA should sign. This again is modeled as simple public key `app_pk`. Additionally, the signature-based proof-of-possession `popo_sig` is calculated by signing the `app_pk` with its corresponding private key `app_sk`. Finally, the message protection signature is calculated over the sender nonce, the application's public key and the proof-of-possession value using the previously declared device secret key of the end entity `dev_sk`. Table 4.1 summarizes the Verifpal model for the CMP initial request message.

CMP message structure	Verifpal message representation
header	`ca_nonce`
body	`app_sig`
protection	`ip_sig`
extraCerts	`ca_pk`, `ca_sig`

Table 4.2.: Simplification of the CMP initial response message for Verifpal

The next phase to model is the corresponding response message `ip` created by the `Sub_CA`: For this, first the received *extraCerts*, the *protection* signature and the *proof-of-possession* are verified. Then, again from all CMP header fields we only declare a freshly generated *sender nonce*. The response message shall contain the signed application certificate created from the received certificate template. The signature is calculated using the `ca_sk` known to the `Sub_CA`. Afterward the message protection `ip_sig` is calculated over the sender nonce, receiver nonce and the signed application certificate.

Table 4.2 summarizes the response message fields as transmitted in the Verifpal model. Note that `e_nonce` and `app_pk` cannot be sent, since Verifpal does not allow receiving a constant despite already knowing it. However, from the attacker's point of view this does not limit the expressiveness of the model. Because upon receiving the `ip` message the end entity verifies the received CA certificate, the message protection and the validity of the application certificate signature. Therefore, the verification succeeds if and only if the same `e_nonce` and `app_pk` are used by both principals.

Since we assume that CMP is configured to use *implicitConfirm*, the enrollment of a new end entity is concluded at this point.

4.4.2. Updating a Valid Certificate

The Verifpal model visualized in Figure 4.5 addresses the key update operation of a valid certificate described in Section 3.4.2. After a successful initial enrollment of an end entity the trust anchor shifts from the `Root_CA` to the `Sub_CA`. Hence, the setup of the principals `Sub_CA` and `End_Entity` in Verifpal requires both to know the private key of the `Sub_CA`. This way, the `End_Entity` can create a valid signed application certificate (`app_pk`, `app_sig`) without any additional message transfer in this model. At this point this approach is justified, since it reflects the initial state of the key update operation in the real CMP protocol: The `Sub_CA` should possess its own signed CA certificate which has already been verified by the `End_Entity` during the initial enrollment. Therefore, as a simplification and the shifted trust anchor we can dispense with the modeling of the certificate chain up to the `Root_CA`. Moreover, it is assumed that during the initial enrollment the `End_Entity` received the CA public key and a signed application certificate (`app1_pk`, `app1_sig`). This prior knowledge is remodeled in the first part of the communication diagram in Figure 4.5 The actual key update request `kur` begins with the sender nonce `e_nonce` and a new public-private key pair for the application (`napp2_pk`, `app2_sk`). With the same reasoning of the

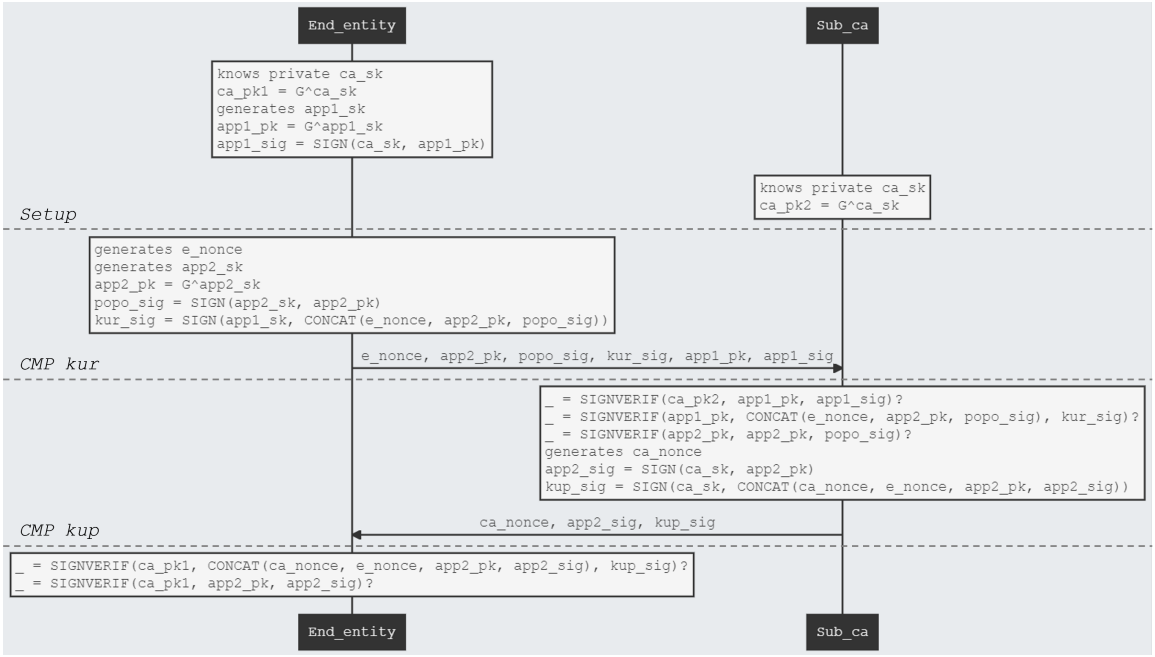


Figure 4.5.: Verifpal communication model of CMP key update

CMP message structure	Verifpal message representation
header	`e_nonce`
body	`app2_pk`, `popo_sig`
protection	`kur_sig`
extraCerts	`app1_pk`, `app1_sig`

Table 4.3.: Simplification of the CMP key update request message for Verifpal

cryptographic purpose as in Section 4.4.1 only the nonce field of the CMP header is represented in this model. Afterwards, the signature-based proof-of-possession `popo_sig` is calculated by signing the `app2_pk` with its corresponding private key `app2_sk`. Finally, the message protection is calculated over the sender nonce, the new public key and the proof-of-possession field using the private key `app1_sk` of the currently valid application certificate. Table 4.3 summarizes the fields of the key update request `kur`. Assuming that the CA does not store issued certificates, the application certificate (`app1_pk`, `app1_sig`) needs to be sent as *extraCerts*.

Upon receiving the request, the `Sub_CA` proceeds similar to the initial enrollment. First, the authenticity of the currently valid application certificate (`app1_pk`, `app1_sig` is verified, followed by the message protection and the proof-of-possession signatures. In case of a valid request, the response message `kup` is calculated. For this, a fresh sender nonce `ca_nonce` is generated and the the new public key for the application is signed with the private key of the `Sub_CA`. Finally, the newly signed application certificate (`app2_pk`, `app2_sig`) is protected by the `kup_sig` signature next to the `ca_nonce` and the `eenonce_kur` values. Here again, the `e_nonce` and `app2_pk`

CMP message structure	Verifpal message representation
header	`ca_nonce`
body	`app2_sig`
protection	`kup_sig`
extraCerts	–

Table 4.4.: Simplification of the CMP key update response message for Verifpal

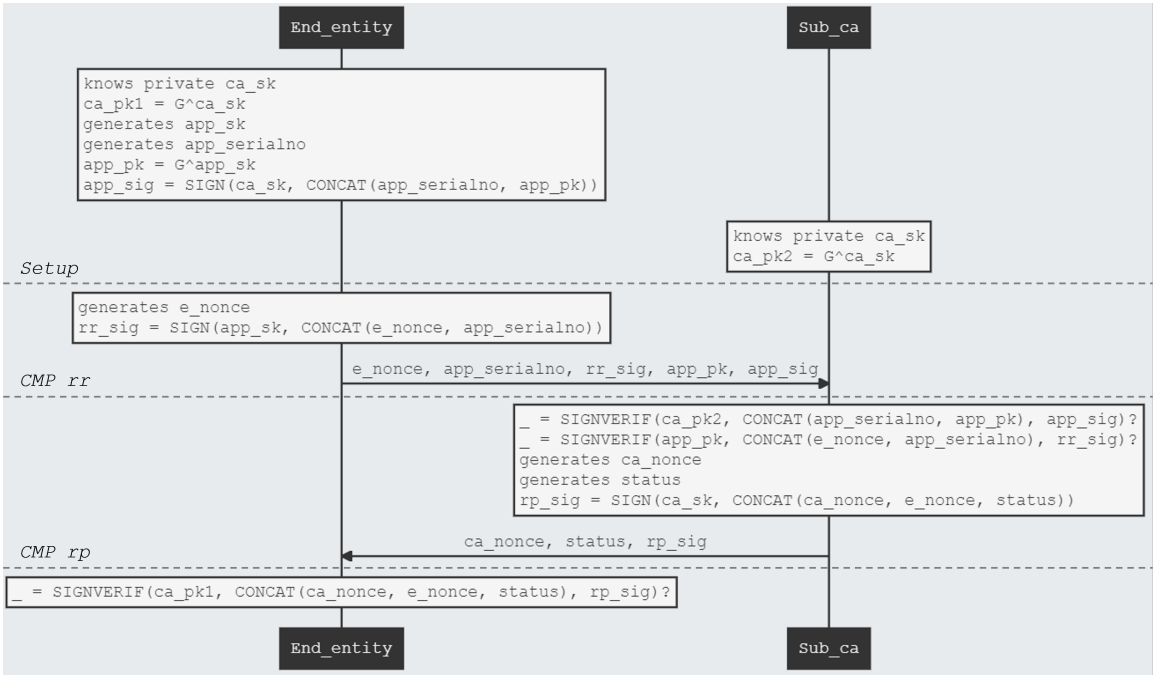


Figure 4.6.: Verifpal communication model of CMP revocation request

cannot be transmitted since both are already known to the `End_Entity`. Table 4.4 lists all the message fields of the `kup` which are sent in the Verifpal model. It shows that no additional *extraCerts* need to be attached to the message, because all necessary key material should already be known by the `End_Entity` at this point.

The final operations of the `End_Entity` are verifying the message protection `kup_sig` and the received signature for `app2_pk`. When successful, the old application certificate can be revoked.

4.4.3. Revoking a Certificate

The setup for the Verifpal model of the certificate revocation (Section 3.4.3) is almost identical to the key update described in Section 4.4.2. In the initial state, the principals `Sub_CA` and `End_Entity` know the key pair (`ca_sk`, `ca_pk`) reflecting the fact that the CA has already been established as trust anchor for the end entity. Additionally, the `End_Entity` again creates a signed application certificate but this time it is necessary to model the certificate with its serial number. In the case of a CMP

CMP message structure	Verifpal message representation
header	`e_nonce`
body	`app_serialno`
protection	`rr_sig`
extraCerts	`app_pk`, `app_sig`

Table 4.5.: Simplification of the CMP revocation request message for Verifpal

revocation request, instead of the signed certificate to be revoked itself, a certificate template containing serial number and issuer of the certificate is sent. Since the serial number is due to its uniqueness an additional ephemeral value, we include the serial number in the Verifpal certificate representation. For this, we generate a fresh constant `app_serialno` and calculate the signature over the serial number and the application's public key. This means, here a certificate is modeled slightly different than before as the tuple (`app_serialno`, `app_pk`, `app_sig`).

From the CMP header solely the `e_nonce` is defined as a *fresh* value. The revocation request `rr` in the CMP body represented with the `app_serialno` value. The message protection `rr_sig` is then calculated over the CMP header and body, i. e., sender nonce and the serial number. Table 4.5 shows the message fields of the `rr` message modeled in Verifpal and their correspondence to the CMP message structure. Note, that technically the serial number should be transmitted twice, once in the CMP body and once in the *extraCerts* field as part of the signed application certificate. However, it does not serve any cryptographic purpose sending a duplicate `app_serialno` value and Verifpal does not permit it regardless, it is justified to omit the value in the *extraCerts* field.

In the next step of the protocol, the `Sub_CA` verifies the received application certificate and the message protection value. Afterwards, through internal non-cryptographic processes it is determined whether the revocation request is granted. In any case a *PKIStatusInfo* object is created which contains the information if the request has been *accepted* or *rejected*. In the Verifpal model this can simply be abstracted to a freshly generated value `status`.

The response message `rp` additionally requires a new sender nonce `ca_nonce` for the header. The message protection `rp_sig` is then calculated over `ca_nonce`, the recipient nonce `e_nonce` and the revocation status `status`. As the CA certificate is also already known to the `End_Entity`, the transmission of *extraCerts* can be omitted, see Table 4.6. Like before the `e_nonce` will not be transmitted.

After receiving the response message the `End_Entity` verifies the message protection which concludes the protocol model of certificate revocation.

4.5. Analysis Results

As specified in Section 4.2, the security property of *injective agreement* shall be verified. It has been concluded, that using the `authentication` and `freshness` query, this can be

CMP message structure	Verifpal message representation
header	`ca_nonce`
body	`status`
protection	`rp_sig`
extraCerts	–

Table 4.6.: Simplification of the CMP revocation response message for Verifpal

		Passive attacker		Active attacker	
	Message	Auth.	Freshness	Auth.	Freshness
Certificate request	`app_pk`	✓	✓	✓	✓
	`app_sig`	✓	✓	✓	✓
Certificate renewal	`app2_pk`	✓	✓	✓	✓
	`app2_sig`	✓	✓	✓	✓
Certificate revocation	`app_serialNo`	✓	✓	✓	✓
	`status`	✓	✓	✓	✓

Table 4.7.: Verifpal analysis results

mapped to the Verifpal security goals. The `authentication` query takes a value from a specific message transfer as shown in Listing 4.1 and checks whether it is possible that an attacker can successfully forge this value. The `freshness` query additionally checks for a given value if an attacker can successfully reuse it in a new protocol execution. The queries for all Verifpal models look similar to the code shown in Listing 4.1: we verify the authentication and freshness of (`app_pk`, `app_sig`) (Section 4.4.1), (`app2_pk`, `app2_sig`) (Section 4.4.2) and (`app_serialNo`, `status`) (Section 4.4.3). As Table 4.7 shows all queries passed in the Verifpal analysis. Hence, the security properties specified in Section 4.2 are fulfilled under the security assumptions given in Section 4.1.

```
queries[
authentication? End_Entity -> Sub_CA: app_pk
authentication? Sub_CA -> End_Entity: app_sig
freshness? app_pk
freshness? app_sig
]
```

Listing 4.1: Queries in Verifpal model for certificate requests

Performance For the security analysis of the certificate management protocol, the tool Verifpal version 0.27.0 was used on a Ubuntu Server 20.04.1 SMP virtual machine, deployed with a 32-core Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz and 64 GB of memory.

In the case of a passive attacker, all models were analyzed with all virtual machine resources at their disposal. When analyzing the active attacker, it very quickly became

apparent that the execution time increases significantly with the complexity of the model. In addition, it was observed that when multiple processor cores are used, the utilization of the individual cores drops to 5 - 10 % from a certain degree of parallelization. To achieve the highest possible utilization of a core, and to keep the parallelization overhead within limits, it has been experimentally shown that the use of 10 cores for the analysis leads to an 80 - 90 % utilization, and thus, an efficient use of resources is achieved. Therefore, all three protocol models, instantiated with an active attacker, have been analyzed in parallel by assigning each analysis task to 10 dedicated CPU cores.

Table 4.8 shows the execution time of each analysis. Since the passive attacker can only obtain information from transmitted messages without applying any transformations (), the verification was completed within milliseconds. The analysis of the initial request with an active attacker required more than 12 h, a result of the nested signatures in the protocol model which allow a vast set of mutations for the attacker. Compared to that, the key update has been modeled very lean shifting the trust anchor from the Root_CA to the Sub_CA and therefore, reducing the number of variables and cryptographic operations. The certificate revocation analysis could also be performed within a reasonable amount of time, since the model is very simple and does not contain any new certificates.

	Passive Attacker	Active Attacker
Initial request	0.018 s	744 min 32.252 s
Key update	0.016 s	42 min 33.091 s
Revocation request	0.026 s	25 min 35.200 s

Table 4.8.: Execution time of Verifpal `verify` operation

5. Proof of Concept: Automated Post-Quantum Certificate Management

In this chapter, a proof-of-concept implementation for automated post-quantum certificate management in IIoT networks is presented. Based on the findings of the previous chapters, we specify the requirements of this concept in Section 5.1. Then, a target architecture for embedding a PKI in an industrial network is described in Section 5.2. Afterwards, the procedures for automating the certificate management operations are introduced in Section 5.3. Section 5.4 and Section 5.5 document the integration of post-quantum algorithms in the existing Lightweight CMP Profile implementations. In Section 5.6, the impact of PQC support on the end entity is assessed. Finally, in Section 5.7 the results are discussed w. r. t. their fulfillment of the previously defined requirements.

5.1. Requirements for Certificate Management in IIoT

In Section 2.3.3, the Lightweight CMP Profile has been identified as a suitable candidate since it can be implemented such that it fulfills the requirements for industrial scenarios (Section 3.1). Furthermore, we compared the NIST standardization candidates for post-quantum signature schemes in Section 2.2.2 and concluded, that the integration of Dilithium into the certificate management protocol will be studied in this thesis to provide proof-of-concept for post-quantum certificate management. Finally, in Section 1.1.1 the need for an automated approach has been motivated.

Next, a prototype shall be developed that fulfills the following requirements (REQ 1-5) for an automated post-quantum certificate management system for IIoT infrastructures:

REQ 1 Security zones The certificate management system shall comply with the security zones concept presented in Section 2.1.1. For this, the architecture specified in Section 3.2 shall be embedded in a prototypical IIoT network.

REQ 2 Lightweight end entity implementation The implementation of the end entity shall be as lightweight as possible, because it is expected to have limited resources (Section 2.1). Therefore, the certificate management system shall only support the most crucial certificate life-cycle management operations to enable machine-to-machine communication, i. e., use cases production execution and data streaming (Section 2.1.2.1): the prototype shall be able to handle certificate requests, renewals and revocations.

REQ 3 Self-contained authenticated messages The certificate management system shall perform the management operations from REQ 2 in an authenticated manner using security mechanisms which are independent from the chosen message transfer mechanism. For this, it shall provide an approach to protect the messages in a self-contained way.

REQ 4 Automation The certificate management system shall be designed in a way that it requires only a minimal amount of human interaction. The certificate renewal shall be fully automated, certificate requests and revocations shall only require minimal user interaction.

REQ 5 PQC support The certificate management system shall be able to support the issuance, renewal and revocation of post-quantum certificates. In particular, the Dilithium signature scheme with the parameter set of Dilithium2 has been chosen to be integrated in the prototype as explained in Section 2.2.2. The impact of the use of Dilithium certificates on the certificate management shall be evaluated regarding the memory consumption and execution time of the operation compared to "classical" certificates, here exemplarily represented by ECDSA certificates.

Requirements REQ 2 and REQ 3 are conceptually already fulfilled by the protocol design of the chosen certificate management protocol as described in Chapter 3 together with the profiling in Section 3.5. The latter reduces the feature richness and complexity of CMP to the most crucial options needed to support the desired management operations and thus, aiming for a lightweight end entity (REQ 2). Moreover, the security of the Lightweight CMP Profile against a Dolev-Yao attacker (Section 4.3) has been verified in this work (Section 4.5).

In the following, it is described how the requirements, especially REQ 1, REQ 4 and REQ 5 are met by the prototype developed in this thesis.

5.2. Composition of the Certificate Management System

The prototype is developed based on open-source projects which are implementing the Lightweight CMP Profile. In the following, we introduce the existing prototypes which are used to implement the automated certificate management system. Furthermore, we describe the security zone concept for our solution in order to enable a deployment in an industrial network.

5.2.1. Existing CMP Implementations

Since the standardization of Lightweight CMP Profile has only started in 2021, not many open-source implementations are available. As all the projects that have been used for our prototype are still in active development, Table 5.1 provides information about the used software versions and references to their project repositories.

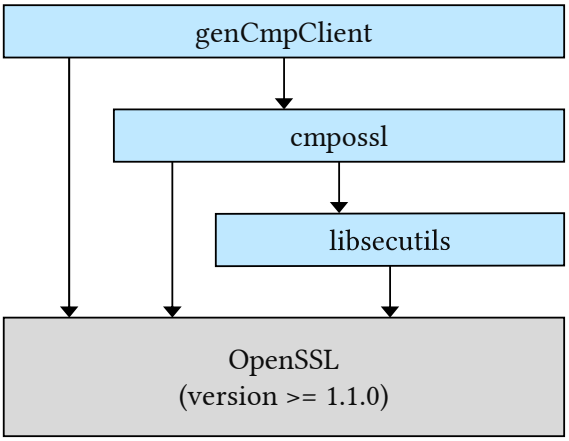


Figure 5.1.: Dependency structure of `genCmpClient`

To the best of our knowledge, the only available end entity implementation of this new profile is the C library `cmpossl` developed by Peylo and Oheimb [PO22]. `cmpossl` is a standalone CMP and HTTP client extension to OpenSSL, for versions 1.1.0 and higher [Ope22], i. e., the library implements the CMP profile using HTTP as message transfer mechanism. Additional to this library, a prototypical high-level application programming interface (API) and a CLI are provided by Kretschmer and Oheimb [KO22b], the `genCmpClient`. This software makes the detailed CMP API of `cmpossl` more practical to use for application programmers, focusing on the support of the main certificate management operations needed in industrial use cases. Therefore, with the `genCmpClient` we are able to implement the functionality of the Lightweight CMP Profile as specified in Chapter 3. To simplify the access to some of the functions of the OpenSSL library, `genCmpClient` uses the library `libsecutils` by Oheimb and Schilling [OS22]. Figure 5.1 visualizes the dependencies between the `genCmpClient` application and the used libraries.

On the side of the PKI management entity, we know of two different open source projects written in Java which support the Lightweight CMP Profile: The free software CA package formerly known as Enterprise JavaBeans Certificate Authority, now simply `EJBCA` [Key22a], and the `lightweightCmpRa` [KO22c], a CLI-based RA application for demonstration and test purposes. In the pursuit of using a software that can also be deployed in a real-world environment with little effort, we first explored the suitability of `Enterprise JavaBeans Certificate Authority (EJBCA)` for our prototype. However, working with `EJBCA` has shown that building the software from its source code requires unreasonably high effort due to its software architecture. Moreover, in the process it became apparent that the integration of PQC in such a complex software environment would affect several other modules of `EJBCA` additional to the CMP module. Consequently, it would be necessary to familiarize oneself with and adjust components which are not of interest in this thesis, but nevertheless required to ensure the correct functioning of the `EJBCA` CA server. Therefore, this approach has been discarded, and instead, we are now presenting a certificate management

application using the `lightweightCmpRa` which satisfies our goal to develop a proof of concept.

The core CMP functions of `lightweightCmpRa` are encapsulated in a separate software, the `cmpRaComponent` [KO22a]. This software supports all main features of CMP message processing including their generation, protection and validation. But it is not a CA and therefore, it cannot issue certificates. For this, the software intends that the requests of the end entity are forwarded to an external CA which has to be provided by the application developer. However, the `lightweightCmpRa` comes with a mock CA implementation for its test classes, referred to as `mockCa`, in the following. We will utilize this `mockCA` to realize the functionality of the `Sub_CA` as specified in Chapter 3. For the purpose of our prototype, we summarize the `lightweightCmpRa`, the `cmpRaComponent` and the `mockCa` under the term `Sub_CA`.

Software	Lang.	Project Repository	Version
`libsecutils`	C	[OS22]	commit 3199ea8
`cmpossl`	C	[PO22]	commit fa1a4ee
`genCmpClient`	C	[KO22b]	commit a791495
`cmpRaComponent`	Java	[KO22a]	version 2.2.0, commit 038ad5e
`lightweightCmpRa`	Java	[KO22c]	commit c8eb6ca

Table 5.1.: Overview of the software used in the prototype

5.2.2. PKI Architecture in IIoT Networks

In Section 3.2 we introduced our PKI architecture envisioned for use in industrial networks. However, to be embedded into an actual IIoT infrastructure, requirement `REQ 1` demands the certificate management system to comply with the IEC 62443 security zones concept described in Section 2.1.1. Figure 5.2 shows the prototypical demonstration environment which has been developed for our implementation:

The certificate management shall facilitate certificate-based communication between control applications in the production zone and applications in the on-premise zone. Physically, our demonstration setup consists of two devices: A Raspberry Pi 4 representing a constrained control device in the production zone, and an Ubuntu virtual machine in the on-premise zone.

The *production zone (red zone)* represents a network-separated zone for the operation of information technology (IT) systems that are required for the manufacturing process. Communication into or out of the red zone shall be prohibited by default; communication connections that are required for operation shall be configured in the firewall. Semantically, a red zone can be envisioned as one station on a plant floor and interconnections between red zones are not allowed. A manufacturing execution system which supervises, manages, controls and monitors the whole production process is intended to be deployed in the *on-premise zone (yellow zone)*. To access machines in the production zone from the *company zone (green zone)* conduits need

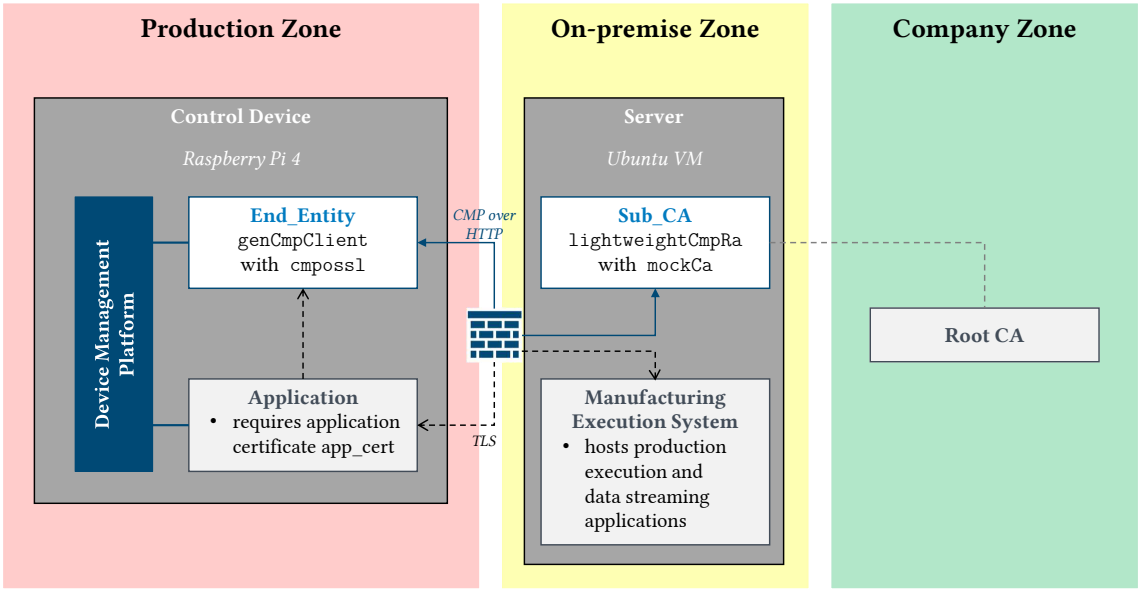


Figure 5.2.: Architecture of the IIoT production environment demonstrator.

to be established at the zone borders, i. e., firewall configurations as described in Section 2.1.1.

The certificate management tool shall enable two different use cases for IT in production: production execution and data streaming facilitating both client-server communication and the publish-subscribe pattern. As described in Section 2.1.2, the certificates used for both communication patterns are similar in case TLS and MQTT are used. In this work, we focus on the management of TLS certificates. An application on the control device, which needs to communicate with the manufacturing execution system in the yellow zone, shall be able to automatically renew an application certificate.

For this, the PKI architecture from Section 3.2 shall be embedded in the three zones: The `End_Entity` is envisioned to be an application on the control device in the red zone that can be controlled by a device management platform, which centrally administers all applications and services on the control device. The `Sub_CA` is located in the yellow zone. For the sake of completeness, we also show the `Root_CA` operated in the company network, but it is not part of our experimental setup (see Section 3.2 for details).

In the scope of this thesis, we implement the functionality of the `End_Entity` on the Raspberry Pi and the `Sub_CA` on the Ubuntu VM, respectively. Our primary goal is to provide a proof of concept for the automated post-quantum certificate management. Therefore, we focus on the implementation of the Lightweight CMP Profile and the integration of post-quantum cryptography in the existing prototypes introduced in Section 5.2.1.

5.3. Automation of the Certificate Management Operations

As the protocol follows the pull model (Section 3.2), all certificate management operations are initialized by the `End_Entity`. Therefore, to meet `REQ 4` the interaction with the end entity application `genCmpClient` needs to be automated. For this, we assume that the certificate management application is connected to a device management platform. A human user interacts with this platform during the provisioning process, and whenever they need to install a new application on the device or make changes to the existing ones.

In the following sections, the proposed interaction between the user, the platform, the `End_Entity` and the `Sub_CA` for each certificate management operation are explained.

5.3.1. Certificate Request

The certificate request is split in two phases triggering the initial request `ir` during the device provisioning and the actual application certificate request `cr` during operation when the application is installed.

Figure A.1 visualizes the activities for enrolling the `End_Entity` to the `Sub_CA`. When the user deploys a new control device in the production zone, they need to perform a several tasks. As assumed in Section 3.2 and in Section 5.2.2, they install the device certificate with its corresponding public-private key as well as the root certificate as trust anchor. Additionally, the network information about the `Sub_CA` is provided to the `End_Entity`, i. e., the network address and port of the `Sub_CA`. Afterwards, a background process is scheduled to trigger the initial request: For this, the `End_Entity` generates a new key pair and sends the `ir` message to the `Sub_CA`. The information about the certificate subject and key usage for the certificate template are provided by the platform. We use `ir` to get an operational certificate which shall be used to authenticate further transactions with the `Sub_CA`. In real-world environments it is not usual, that the PKI management entity and the end entity already share the same trust anchor. Instead, the device certificate is rather issued by the device manufacturer whose CA needs to be trusted by the `Sub_CA`. In our system, we simplified this fact as the device certificate provisioning is out of scope. So, upon receiving the operational certificate in the *ip* message by the `Sub_CA`, the `End_Entity` stores the certificate and notifies the user via the platform..

In the next phase, shown in Figure A.2 we assume that the user configures and installs a new application on the device using the device management platform. If the configuration requires certificates, the platform schedules a job during the installation which triggers the `End_Entity` to send a certificate request to the `Sub_CA` using the operational certificate for authentication. Again, the `End_Entity` generates a new key pair when triggered and sends `cr` to the `Sub_CA`. After receiving the new application certificate, the `End_Entity` stores it on the device and makes it available to the application. The user is notified after the installation of the application about the completed certificate request.

5.3.2. Automated Certificate Update

Each certificate comes with an expiration date as shown in Figure 2.2. To renew the certificate in due time before the currently valid application certificate expires, a job is scheduled directly after the installation of the application certificate which triggers the update at a pre-configured time t_{update}. This time is a specific point in time which is before the expiration time t_{not_after} specified by a buffer period t_{buffer}. So, that is $t_{update} = t_{not\_after} - t_{buffer}$. The parameter t_{buffer} is determined by the PKI operator and security policies. This activity can already be seen in Figure A.1 and Figure A.2. Figure A.3 is the corresponding sequence diagram visualizing the activities for automatically updating a valid certificate on a control device before it expires.

When the update job is triggered at time t_{update}, the `End_Entity` generates a new key pair and sends the `kur` message to the `Sub_CA` authenticating themselves using the current application certificate. After the certificate has been updated, the user is notified via the device management platform. Again, for the new certificate a new job is scheduled which shall update this certificate in the future. This can easily be realized with the Linux shell command `at` given the calculated time t_{update} [IT18].

5.3.3. Certificate Revocation

When the user changes a policy or any authorization settings for the application, the device management platform checks if the validity of the used application certificate is affected. If, for example the certificate shall no longer be used, the platform triggers the `End_Entity` to revoke the certificate. For this, the `End_Entity` creates a `rr` message with a revocation reason code given by the platform and sends the message to the `Sub_CA`. The latter handles the request and revokes the certificate. The result of the certificate revocation is then reported to the user via the platform. Figure Figure A.4 visualizes the interaction for a certificate revocation as described here.

5.4. Lightweight CMP End Entity

The functionality of the `End_Entity` is realized using the `genCmpClient` introduced in Section 5.2.1. In the following, the functionality and configuration of the software is described. Subsequently, the integration of the Dilithium algorithm for post-quantum certificate mangement is described. The `End_Entity` functionality is implemented on a Raspberry Pi 4 Model B Rev 1 running Ubuntu 20.04 LTS. The Pi 4 is equipped with a ARM Cortex-A72 (ARM64 architecture) and 8 GB RAM.

5.4.1. Functionality

The CLI of `genCmpClient` comes with four pre-defined *use cases* which correspond to the CMP operations in `REQ 2` as shown in Table 5.2: `imprint`, `bootstrap`, `update` and `revoke`. Each command calls a high-level API function in `genCmpClient` which encapsulates the low-level API calls to the core CMP in the `cmpossl` library. In Figure 5.3 the function calls to the `cmpossl` library triggered by the high-level API

`genCmpClient` command	Certificate management operation
`imprint`	enroll end entity to new PKI (`ir`/`ip`, Section 3.4.1)
`bootstrap`	request a certificate (`cr`/`cp`, Section 3.4.1)
`update`	update a valid certificate (`kur`/`kup`, Section 3.4.2)
`revoke`	revoke a certificate (`rr`/`rp`, Section 3.4.3)

Table 5.2.: `genCmpClient` CLI commands

calls for certificate requests and renewals in the `genCmpClient` are visualized. For example, the `CMPclient_imprint()` is used in the diagram to initiate the enrollment of a new end entity; the function calls of the `bootstrap` and the `update` are similar. Additionally, in Figure 5.4 the function calls of the `revoke` command are shown.

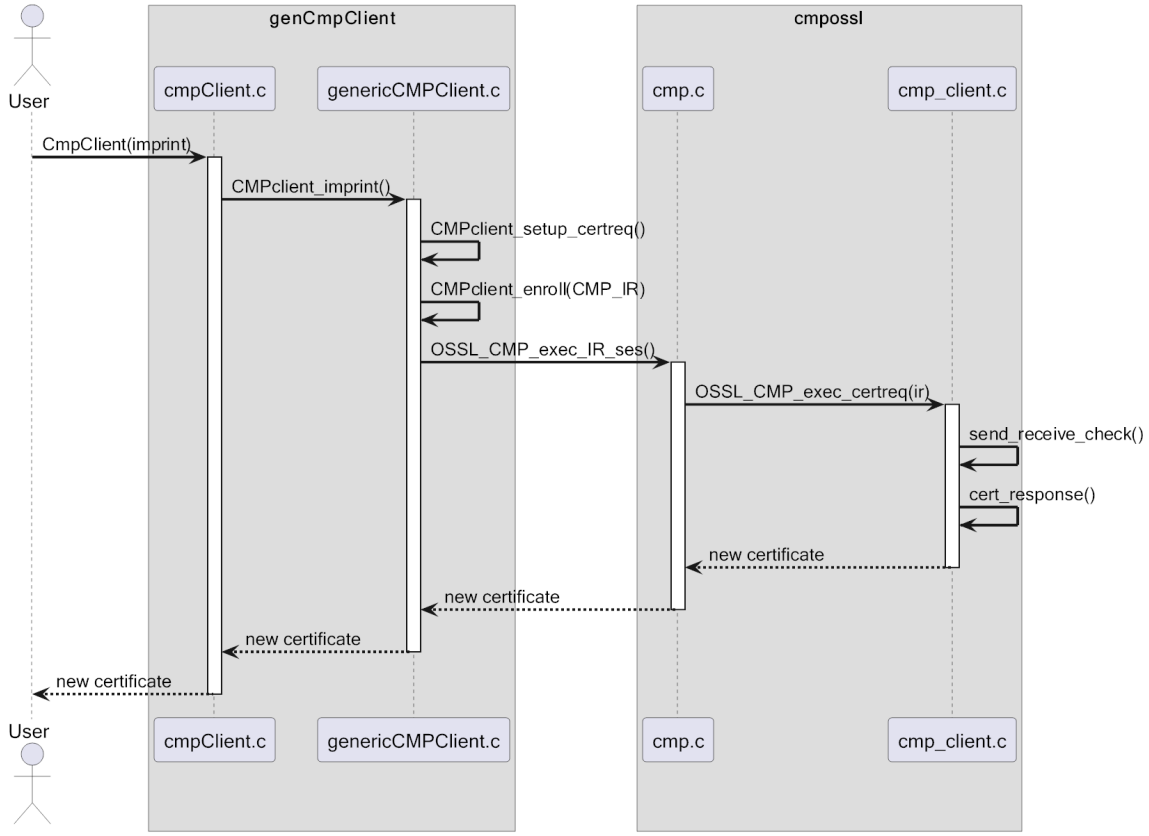


Figure 5.3.: Function call sequence for CMP initial request operation

The diagram in Figure 5.5 shows the processing of a certificate management operation when the request is executed via the CLI tool. First, the `genCmpClient` instantiates a `CmpClient` in `CMPclient_init()` where the OpenSSL library is initialized. Afterwards, in `CMPclient_prepare()` the internal CMP context data structure is allocated and the CMP parameters which are common to all use cases are set-up. This parameter set contains the network address of the HTTP server, the client and server authentication certificates and the enrollment options. A complete list of all supported parameters is

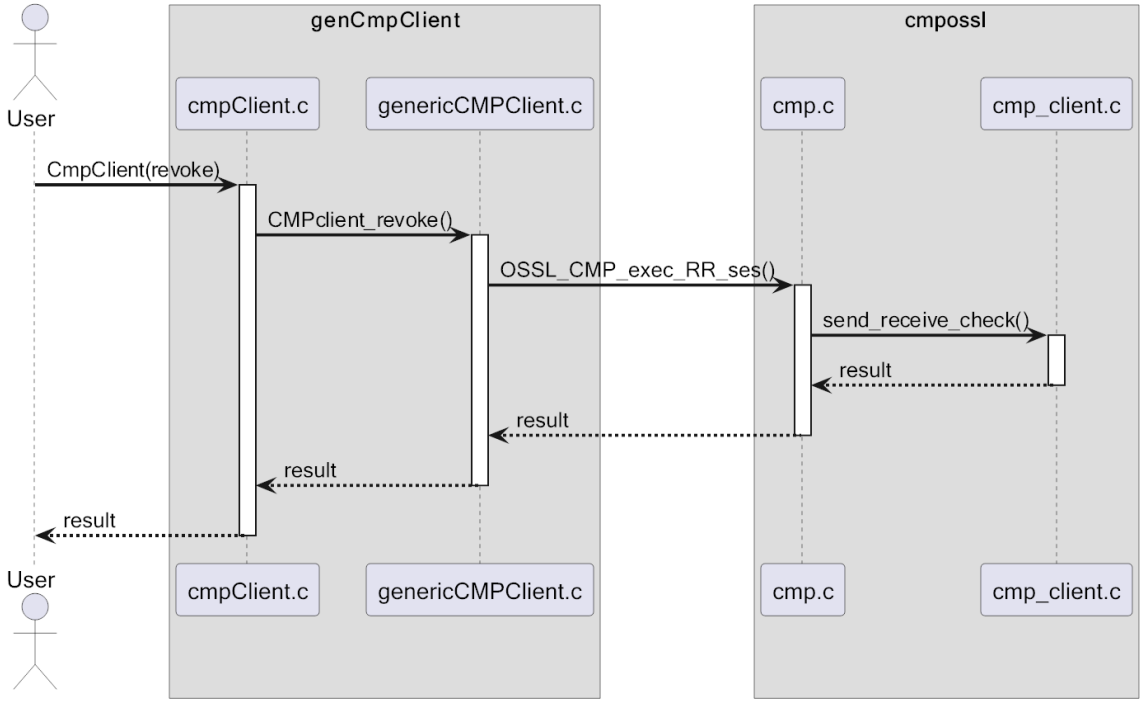


Figure 5.4.: Function call sequence for CMP revocation request operation

provided in the OpenSSL 3.0 manual [Aut22]. After this step, the use case specific functions are called which in return execute the actual CMP operation. In case of `imprint`, `bootstrap` and `update` a new certificate is returned by the function which is then saved by the application before termination. Otherwise the command is directly completed by deallocating the CMP context and all internal data structures in the `CMPclient_finish()` function.

5.4.2. Configuration

Instead of passing the parameters of a command through the CLI to the application, it is an OpenSSL best practice to specify them in a configuration file. Listing 5.1 shows the parameter set for initializing the `genCmpClient`. It contains the address of the HTTP server on which the PKI management entity (the `Sub_CA` implementation) is deployed. As described in Section 3.2, the initial certificates have been specified with the certificate of `Root_CA` as trust anchor and the device certificate as authentication method. The options `implicitConfirm` and `popo` have been activated in order to comply with the profile features specified in Section 3.5. Now, for the certificate management operations in the configuration file additional sections are defined where information such as the subject and the path to the new public key and its key type, or in case of revocation the reason code, are provided.

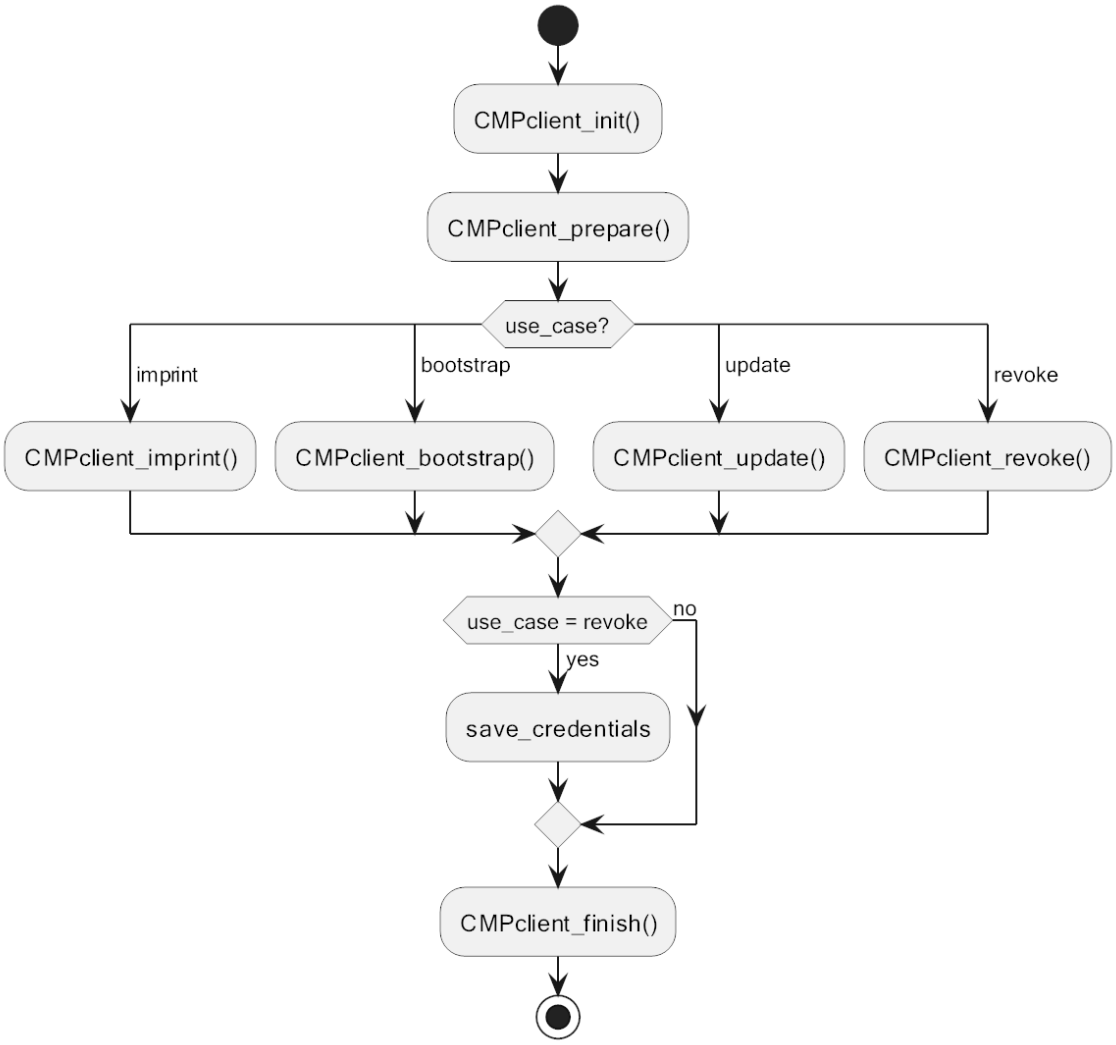


Figure 5.5.: Procedure for certificate management operation when executing a `genCmpClient` command

5.4.3. Integration of Dilithium2 Signature Scheme

By default, the `genCmpClient` uses any OpenSSL version higher than version 1.1.0 which is assumed to be already installed on the system. To support post-quantum certificates, we migrated the project to use the `libOQS` crypto library [SM21]. In particular, we use OQS-OpenSSL_1_1_1 [SM22] – a fork of OpenSSL 1.1.1s that adds quantum-safe key exchange and authentication algorithms using `liboqs` for prototyping and evaluation purposes. Replacing the standard OpenSSL installation with OQS-OpenSSL requires installing it manually: Listing 5.2 shows how the library was configured to be used as a shared library with debug flags. It is necessary to set the `RPATH` variable for the OpenSSL library path when enabling the `shared` option to ensure OpenSSL uses the correct `libssl` and `libcrypto` libraries after installation. The `genCmpClient` was then rebuild with a custom `OPENSSL_DIR` pointing to the OQS-

```
[CmpRa] # LightweightCmpRa
# message transfer
server = http://10.163.28.209:6000/onlinelra
# client authentication
cert = pqc-credentials/cmp-creds/trusted/Device_Chain.crt.pem
key = pqc-credentials/cmp-creds/Device.key.pem
#server authentication
trusted = pqc-credentials/cmp-creds/trusted/Root_CA.crt.pem
# certificate enrollment
subject = "/CN=Client Test Application"
out_trusted = pqc-credentials/cmp-creds/trusted/Root_CA.crt.pem
popo = 1
implicit_confirm = 1
```

Listing 5.1: `genCmpClient` configuration

OpenSSL installation directory. Additionally, it turned out that the `libsecutils` library was using a custom definition of `bool` for logging purposes which conflicted with the OQS-OpenSSL installation. Therefore, the macro was renamed in order to fix this issue. Another problem was caused by the use of the memory error detector `AddressSanitizer` [Ser+12] in `libsecutils`, as it is not supported on ARM64 architectures. So, deactivating this feature resolved previous runtime errors. Since the cryptographic operations are all encapsulated in the OpenSSL library, no changes in the implementation of `genCmpClient` and `cmpossl` were required.

We used the OQS-OpenSSL to generate both the ECDSA and the Dilithium certificates for the initialization of the `End_Entity` and the `Sub_CA`. For this, first a root certificate, i. e., self-signed certificate, was created. Then a certificate signing request was created for the device certificate using the OpenSSL `req` command. Afterwards, the device certificate was signed with the root certificate using the OpenSSL `ca` command. In a similar fashion, the certificate for the `Sub_CA` has been created.

To run the `genCmpClient` with Dilithium certificates we adapted the configuration described in the previous section and replaced the ECDSA certificates with their Dilithium certificate equivalent.

```
./config -Wl, -rpath=OPENSSL_LIB_PATH -Wl,--enable-new-dtags shared --prefix=
    OPENSSL_DIR --openssldir=OPENSSL_DIR no-asm -g3 -O0 -fno-omit-frame-pointer -fno
    -inline-functions
```

Listing 5.2: Build config command for OQS-OpenSSL_1_1_1

5.5. PKI Management Entity

The functionality of the `Sub_CA` is realized with the `lightweightCmpRa` application [KO22c] which uses the `cmpRaComponent`. Figure 5.6 shows the components of the `cmpRaComponent` which main task is the processing of CMP messages from the `End_Entity`. The `RA Upstream` block refers to an externally provided CA which issues and revokes certificates. In our prototype this functionality is covered by a `mockCA` which performs these tasks in a rudimentary way. The `RA Downstream` block is the interface at which the CMP requests are received. The necessary cryptographic functions are encapsulated in the component `cryptoservices` which is used for message generation, protection, and validation. The `Sub_CA` functionality is implemented on a Ubuntu Server 20.04.1 SMP virtual machine, deployed with a 32-core Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz and 64 GB of memory.

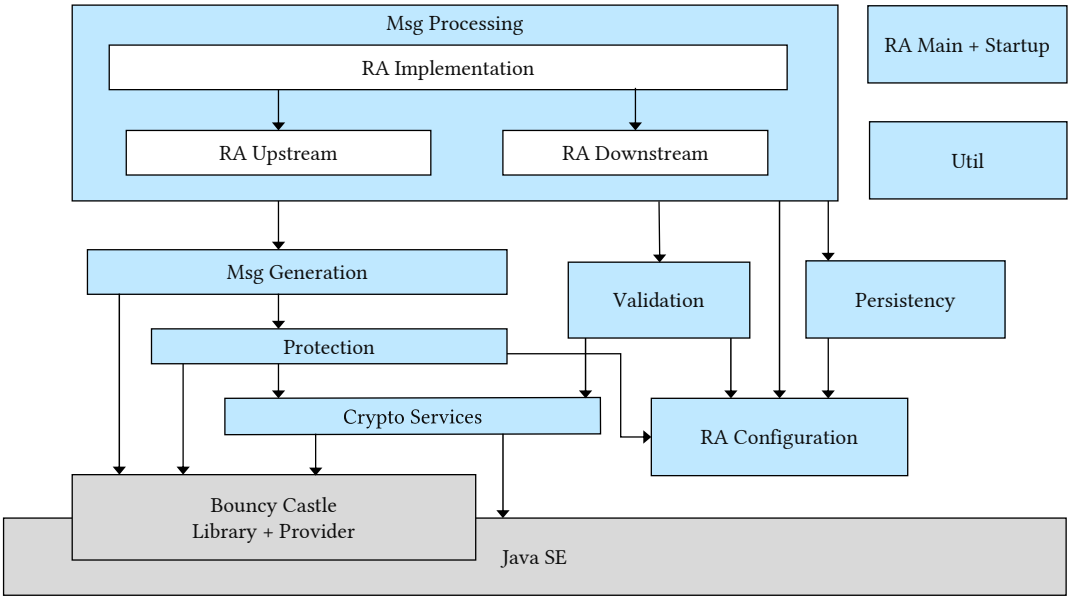


Figure 5.6.: Component design of `CmpRaComponent` [KO22a]

Figure 5.7 shows the interaction of the `cmpRaComponent` with its downstream and upstream interfaces. The configuration interface accessed by the `lightweightCmpRa` is used to instantiate the `cmpRaComponent`. At the downstream interface, i. e., HTTP server on the `Sub_CA`, the messages of the `End_Entity` are received and forwarded to the RA as a request for processing. After validating the message, the upstream interface, i. e., the `mockCA` is contacted to either get a new certificate issued or a certificate revoked. When this upstream exchange is completed, the response message

is generated and protected by the `cmpRaComponent` and sent to `End_Entity` via the downstream interface.

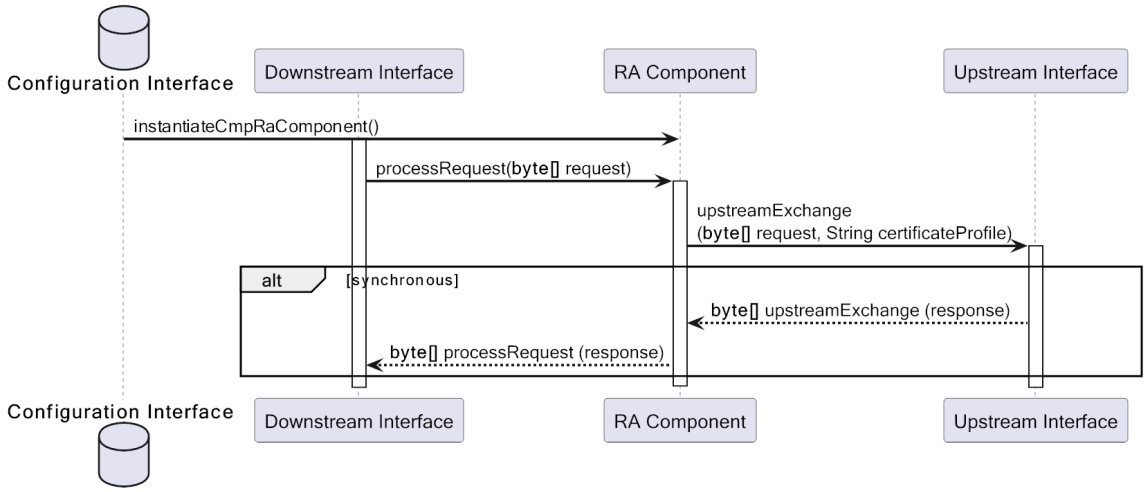


Figure 5.7.: Sequence diagram for instantiating the `CmpRaComponent` [KO22a]

The configuration of the authentication options and the trusted certificates is provided by YAML files which are used during the initialization of the HTTP server.

5.5.1. Integration of Dilithium2 Signature Scheme

The cryptographic primitives of the `lightweightCmpRa` are provided by the Bouncy Castle library, version 1.72 [Leg22]. This software version already supports post-quantum algorithms, in particular Dilithium2. However, in order to use the PQC functions it is necessary to instantiate a new Java Security Provider, the `BouncyCastle Post-Quantum Security Provider (BCPQC)` in the `cryptoservices` component. The X509 certificate handling functionality is assigned to the default provider whilst only PQC primitives are in `BCPQC`. Therefore, the `cryptoservices` as well as the message generation, protection and validation classes have been extended with a case distinction for ECDSA and Dilithium. Doing so, it is required to switch the security provider in the specific functions executing the cryptographic primitives sign and verify depending on the currently active algorithm.

Implementation Obstacles Due to the distribution of the functionality in the different components, the integration of Dilithium2 proved to be more complex than initially expected. Especially, the fact that the Bouncy Castle library encapsulates the cryptographic primitives in two different security providers made the integration more difficult. In the end, the integration could not be successfully completed. The latest status is, that during the message validation, the `cmpRaComponent` calls the standard Bouncy Castle security provider instead of `BCPQC`, although the latter is initialized in the function context. We verified with the Java debugger that the messages sent by the `End_Entity` are received correctly and well-formatted. However,

the message processing aborted with an error before the message generation step could be executed.

5.6. Impact of Post-quantum Cryptography on End Entity

In this work, one main requirement for the certificate management system is to pursue a lightweight approach for the end entity as it is run on a constrained device (REQ 2). Therefore, we assess the memory consumption and the execution time of our prototype as it has been described in the previous sections. The goal is to assess the impact of the integration of post-quantum signature schemes and certificates in the certificate management system. For this, we use the ECDSA algorithm with curve P-384 as baseline for our comparison. The evaluation is performed on a Raspberry Pi 4 Model B Rev 1 running with Ubuntu 20.04 LTS. The Pi 4 is equipped with an ARM Cortex-A72 (ARM64 architecture) and 8 GB RAM.

5.6.1. Certificate and Message Sizes

The first aspect of this assessment is the size of certificates and CMP messages. The certificates used in the prototype are generated with OQS-OpenSSL; Table 5.3 list the sizes of the certificates. For operational and application certificates, we can only provide an estimate based on the other certificate sizes as they could not be obtained via the CMP implementation. Nevertheless, it is evident that the Dilithium certificates are approximately 6 times larger than the ECDSA certificates.

Certificate type	ECDSA size (Bytes)	Dilithium2 size (Bytes)
Root	671	4259
Device	781	4369
`Sub_CA`	659	4247
Operational	$\sim$ 710	$\sim$ 4300
Application	$\sim$ 710	$\sim$ 4300

Table 5.3.: Sizes of DER-formatted certificates

This directly influences the size of the CMP messages transmitted to and received from the `Sub_CA`. Reviewing the message fields as they are specified in Section 3.3 and Section 3.4, the biggest contributing factor to the message size are the public keys, signatures and certificates conveyed in them. Table 5.4 provides the number of each depending on the message type. Knowing the sizes for these parameters from Table 2.2 and Table 5.3, the following estimates can be made: Based on these numbers, it can be assumed that the size of the request messages for new certificates and updating certificates (`ir`/`cr`/`kur`) increases by factor 10 when changing from ECDSA to Dilithium. The response messages `ip`/`cp`/`kup` are estimated to be 7.5 times larger with Dilithium than with ECDSA. And finally, the message sizes for certificate revocation `rr`/`rp` approximately increase by factor 8. Since the other fields

in the certificates and the messages stay the same in terms of their size, the estimates derived here are assumed realistic. In the worst case, up to 11 kByte of data would be transmitted over the network in a single CMP message when using Dilithium2 certificates.

Message type	# public keys	# signatures	# certificates
`ir/cr/kur`	1	2	1
`ip/cp/kup`	–	1	2
`rr/rp`	–	1	1

Table 5.4.: Number of public keys, signatures and DER-formatted certificates in CMP messages

5.6.2. Execution Time

Algorithm	Sign	Verify	Sign/s	Verify/s
384 bits ecdsa (nistp384)	0.0362s	0.0241s	276	414
dilithium2	0.0005s	0.0002s	2091.0	6629.7

Table 5.5.: OpenSSL performance benchmark of signature algorithms on Raspberry Pi 4

To assess the execution time of a CMP transaction, we performed a benchmark of the sign and verify functions of the ECDSA-p384 and Dilithium2 algorithms as these cryptographic operations take up the main part of the run time. For the benchmark, the OQS-OpenSSL performance test application `speed` has been used to measure the execution time of the cryptographic primitives provided by the OQS-OpenSSL library. The benchmark results on the Raspberry Pi 4 are provided in Table 5.5. They are calculated by running each function for 10 s and then determining the average time for a single execution and the average number of completed executions per second. In direct comparison, the values show that a Dilithium2 signing operation is indeed 7.24 times faster than the ECDSA signing operation. For the signature verification, the difference is even larger: The average time to verify a Dilithium2 signature is 120.5 faster than a ECDSA signature.

Table 5.6 summarizes the number of sign and verify operations in `genCmpClient` for each message type. Together with the benchmark, the expected execution time for processing a CMP request or response can be extrapolated. In case of the request messages `ir/cr/kur/rr` the execution time is expected to be 7.24 times faster for Dilithium2 compared to ECDSA, and for the response messages `ip/cp/kup/rp` by a factor of 120.5, respectively. So, in terms of execution time the certificate management system benefits from the migration to Dilithium certificates, improving the necessary processing time significantly.

Message type	# Sign	# Verify
`ir/cr/kur`	2	-
`ip/cp/kup`	-	3
`rr`	1	-
`rp`	-	2

Table 5.6.: Number of sign and verify operations in `genCmpClient` for each message type

Requirement	Fulfillment status	
`REQ 1` Security zones	(✓)	(Section 5.2.2)
`REQ 2` Lightweight end entity	✓	(Sections 3.5 and 5.4)
`REQ 3` Self-contained messages	✓	(Sections 3.3.2, 4.5 and 5.4)
`REQ 4` Automation	(✓)	(Section 5.3)
`REQ 5` PQC support	(✓)	(Sections 5.4.3 and 5.5.1)

Table 5.7.: Overview of requirements and their fulfillment status

5.7. Discussion

Table 5.7 summarizes the fulfillment status of the requirements specified in Section 5.1. Some of the requirements could only be fulfilled in the design concepts developed in this thesis, but not fully implemented in the prototype. Therefore, their fulfillment is considered only partially complete and marked with (✓). With the use of the Lightweight CMP Profile requirements `REQ 2` and `REQ 3` are met. The latter has been formally verified with computer-aided verification in Chapter 4. The automation concept in Section 5.3 enables a fully automated certificate update, certificate requests require minimal user interaction, but not directly with the certificate management system. Due to the incomplete implementation of the PKI management entity functionality described in Section 5.5, the impact of PQC could not be evaluated in detail. However, the assessment in Section 5.6 already indicates that the use of Dilithium2 would require less execution time than ECDSA with the cost of up to 10 times larger message sizes.

6. Conclusion

With the increasing interconnectedness of devices in modern IIoT networks the demand for authenticated communication has risen. The current best practice for this is the use of digital certificates and PKIs. As industrial networks grow, it can become difficult for network administrators and device operators to manually manage the number of devices and certificates. Automated certificate management can scale to handle large networks, ensuring that all devices have up-to-date and valid certificates. Moreover, with the long lifetime of IIoT devices it is desired to use future-proof cryptographic algorithms, which protect against threats such as attacks aided by quantum computers.

Therefore, in this thesis an approach for automated post-quantum certificate management for IIoT infrastructures has been developed. Based on the current Internet-Draft for Lightweight CMP Profile [BOF23] we specified a reduced feature set that supports the most crucial certificate management operations for IIoT devices: requesting new certificates, renewing and revoking them. Moreover, we formally analyzed the security properties of the protocol using the verification tool Verifpal. As a result, we showed that the protocol messages as they have been modeled in Section 4.4 are unforgeable for a Dolev-Yao attacker, which formally corresponds to the authentication notion of injective agreement. Additionally, we developed a PKI architecture embedded in the IEC62443 security zones introduced in Section 2.1.1 and provided a concept to reduce human interaction with the proposed certificate management system. As a result, the certificate update is fully automated except for the notification of the user. A proof of concept has been implemented using an existing CMP application and extended with Dilithium2 algorithm support. In a first assessment, it has been estimated that the execution time of the certificate management operations can be reduced with the use of Dilithium2 instead of ECDSA signature scheme approximately by factor 7 for request messages, and factor 120 for response messages. However, this comes at the cost of significantly increased certificate and message sizes due to the large public key and signature sizes of the Dilithium algorithm.

Future Work We propose to extend our approach with a MAC-KEM as message protection mechanism. In the NIST standardization process the lattice-based KEM Kyber has been chosen to be standardized [CSR22]. As MACs are in general much smaller than digital signatures, this approach can reduce the message sizes with the cost of additional communication to establish a shared secret between the communication partners using Kyber [Bro+23]. However, the use of symmetric cryptography might enhance the execution time when implemented in hardware.

Another research direction could address the issue of initial trust establishment. So far, in our work, we made the assumption that both the PKI management entity and

the end entity possess initial certificates. Especially on the side of the control devices, a concept for secure certificate provisioning should be evaluated.

This thesis focused on a solution for post-quantum certificate management. An open question here is, whether migrating from classical to post-quantum certificates affects the security of the certificate management protocol. To the best of our knowledge, currently there are no restrictions in the Lightweight CMP Profile [BOF23] on using a less secure certificate to request a certificate with stronger security properties. So, it should be investigated how the protocol can be utilized in the migration process and if its security properties hold in this use case.

Bibliography

[ABF17] Martin Abadi et al. "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication". In: *J. ACM* 65.1 (Oct. 2017). ISSN: 0004-5411. DOI: 10.1145/3127586. URL: https://doi.org/10.1145/3127586.

[Ack19] Tim Ackermann. "PKI-basierte Zertifikatsverwaltung für OPC UA". Not published. Bachelor's thesis. Karlsruhe: Karlsruhe Institute of Technology, 2019.

[Ada+05] C. Adams et al. *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. RFC 4210 (Proposed Standard). RFC. Updated by RFC 6712. Fremont, CA, USA: RFC Editor, Sept. 2005. DOI: 10.17487/RFC4210. URL: https://www.rfc-editor.org/rfc/rfc4210.txt.

[AJS20] Monjur Ahmed et al. "Security in Decentralised Computing, IoT and Industrial IoT". In: *Industrial IoT : Challenges, Design Principles, Applications, and Security*. Ed. by Ismail Butun. Cham: Springer International Publishing, 2020, pp. 191–211. ISBN: 978-3-030-42500-5. DOI: 10.1007/978-3-030-42500-5_5.

[Atr+02] Mohan Atreya et al. *Digital Signatures*. New York: McGraw-Hill/Osborne, 2002. ISBN: 0072194820. DOI: 282653. URL: http://www.loc.gov/catdir/description/mh024/2002282653.html.

[Aum+22] Jean-Philippe Aumasson et al. *SPHINCS+ – Submission to the 3rd round of the NIST post-quantum project*. Specification. 2022. URL: https://sphincs.org/data/sphincs+-r3.1-specification.pdf (visited on 07/30/2022).

[Aut22] The OpenSSL Project Authors. *openssl-cmp*. 2022. URL: https://www.openssl.org/docs/manmaster/man1/openssl-cmp.html (visited on 04/29/2023).

[Ban+19] Andrew Banks et al., eds. *MQTT Version 5.0*. OASIS Standard. Organization for the Advancement of Structured Information Standards, 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (visited on 07/26/2022).

[Bar+19] R. Barnes et al. *Automatic Certificate Management Environment (ACME)*. RFC 8555 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Mar. 2019. DOI: 10.17487/RFC8555. URL: https://www.rfc-editor.org/rfc/rfc8555.txt.

[Bar+21] Manuel Barbosa et al. "SoK: Computer-Aided Cryptography". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 777–795. DOI: 10.1109/SP40001.2021.00008.

[Bin+17] Nina Bindel et al. "Transitioning to a Quantum-Resistant Public Key Infrastructure". In: *Post-Quantum Cryptography*. Ed. by Tanja Lange and Tsuyoshi Takagi. Vol. 10346. Springer eBook Collection Computer Science. Cham: Springer, 2017, pp. 384–405. ISBN: 978-3-319-59878-9. DOI: 10.1007/978-3-319-59879-6_22.

[BL17] Daniel J. Bernstein and Tanja Lange. "Post-quantum cryptography". In: *Nature* 549.7671 (Sept. 2017), pp. 188–194. ISSN: 1476-4687. DOI: 10.1038/nature23461.

[Bla12] Bruno Blanchet. "Security Protocol Verification: Symbolic and Computational Models". In: *Principles of Security and Trust*. Ed. by Pierpaolo Degano and Joshua D. Guttman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3–29. ISBN: 978-3-642-28641-4. DOI: 10.1007/978-3-642-28641-4_2.

[Bla16] Bruno Blanchet. "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif". In: *Foundations and Trends® in Privacy and Security* 1.1-2 (2016), pp. 1–135. ISSN: 2474-1558. DOI: 10.1561/3300000004.

[BMS20] Colin Boyd et al. *Protocols for Authentication and Key Establishment*. 2nd ed. 2020. Springer eBooks Computer Science. Berlin, Heidelberg: Springer, 2020. ISBN: 978-3-662-58145-2. DOI: 10.1007/978-3-662-58146-9.

[BOF23] Hendrik Brockhaus et al. *Lightweight Certificate Management Protocol (CMP) Profile*. Internet-Draft. Work in Progress. Feb. 2023. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-lightweight-cmp-profile/21/.

[BOG22] Hendrik Brockhaus et al. *Certificate Management Protocol (CMP) Updates*. Internet-Draft. Work in Progress. June 2022. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-cmp-updates/23/.

[Bro+23] Hendrik Brockhaus et al. *Internet X.509 Public Key Infrastructure – Certificate Management Protocol (CMP)*. Internet-Draft. Work in Progress. Mar. 2023. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-rfc4210bis/06/.

[Buc13] Johannes A. Buchmann. *Introduction to Public Key Infrastructures*. Springer eBook Collection Computer Science. Berlin, Heidelberg: Springer, 2013. ISBN: 978-3-642-40656-0. DOI: 10.1007/978-3-642-40657-7.

[CM12] Cas Cremers and Sjouke Mauw. "Security Properties". In: *Operational Semantics and Verification of Security Protocols*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 37–65. ISBN: 978-3-540-78636-8. DOI: 10.1007/978-3-540-78636-8_4. URL: https://doi.org/10.1007/978-3-540-78636-8_4.

[Cre+17] Cas Cremers et al. “A Comprehensive Symbolic Analysis of TLS 1.3”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Ed. by Bhavani Thuraisingham. ACM Digital Library. New York, NY: ACM, 2017, pp. 1773–1788. ISBN: 9781450349468. DOI: 10.1145/3133956.3134063.

[Cre08] Cas J. F. Cremers. “The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols”. In: *Computer Aided Verification*. Ed. by Aarti Gupta et al. Vol. 5123. Springer eBook Collection Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 414–418. ISBN: 978-3-540-70543-7. DOI: 10.1007/978-3-540-70545-1_38.

[CSR22] CSRC | NIST. *Announcing PQC Candidates to be Standardized, Plus Fourth Round Candidates | CSRC*. 2022. URL: https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4 (visited on 07/25/2022).

[Dah+22] Markus Dahlmanns et al. “Missed Opportunities”. In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. Ed. by Yuji Suga. ACM Digital Library. New York,NY,United States: Association for Computing Machinery, 2022, pp. 252–266. ISBN: 9781450391405. DOI: 10.1145/3488932.3497762.

[DR08] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). RFC. Obsoleted by RFC 8446, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919, 8447, 9155. Fremont, CA, USA: RFC Editor, Aug. 2008. DOI: 10.17487/RFC5246. URL: https://www.rfc-editor.org/rfc/rfc5246.txt.

[DY83] D. Dolev and A. Yao. “On the security of public key protocols”. In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–208. ISSN: 0018-9448. DOI: 10.1109/TIT.1983.1056650.

[EMM09] Santiago Escobar et al. “Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties”. In: *Foundations of Security Analysis and Design V*. Ed. by Alessandro Aldini et al. Vol. 5705. SpringerLink Bücher. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–50. ISBN: 978-3-642-03828-0. DOI: 10.1007/978-3-642-03829-7_1.

[Gam22] Jay Gambetta. “Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing”. In: *IBM* (2022). URL: https://research.ibm.com/blog/ibm-quantum-roadmap-2025 (visited on 07/30/2022).

[GKS19] O. Garcia-Morchon et al. *Internet of Things (IoT) Security: State of the Art and Challenges*. RFC 8576 (Informational). RFC. Fremont, CA, USA: RFC Editor, Apr. 2019. DOI: 10.17487/RFC8576. URL: https://www.rfc-editor.org/rfc/rfc8576.txt.

[Glo22] GlobalSign, Inc. *GlobalSign IoT Identity Platform: PKI-based cloud IoT Identity Platform*. 2022. URL: https://www.globalsign.com/en/internet-of-things/iot-identity-platform (visited on 07/28/2022).

[GT11] Wolfgang Granzer and Albert Treytl. "Security in Industrial Communication Systems". In: *Industrial Communication Systems*. Ed. by Bogdan M. Wilamowski and J. David Irwin. 2nd ed. The Industrial Electronics Handbook. CRC Press, 2011. Chap. 22. ISBN: 9781439802816. DOI: 10.1201/b10603-24. URL: https://www.routledgehandbooks.com/doi/10.1201/b10603-24.

[Gut20] P. Gutmann. *Simple Certificate Enrolment Protocol*. RFC 8894 (Informational). RFC. Fremont, CA, USA: RFC Editor, Sept. 2020. DOI: 10.17487/RFC8894. URL: https://www.rfc-editor.org/rfc/rfc8894.txt.

[Hou09] R. Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652 (Internet Standard). RFC. Updated by RFC 8933. Fremont, CA, USA: RFC Editor, Sept. 2009. DOI: 10.17487/RFC5652. URL: https://www.rfc-editor.org/rfc/rfc5652.txt.

[Hou20] R. Housley. *Update to the Cryptographic Message Syntax (CMS) for Algorithm Identifier Protection*. RFC 8933 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Oct. 2020. DOI: 10.17487/RFC8933. URL: https://www.rfc-editor.org/rfc/rfc8933.txt.

[HPS98] Jeffrey Hoffstein et al. "NTRU: A ring-based public key cryptosystem". In: *Algorithmic Number Theory*. Ed. by Joe P. Buhler. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 267–288. ISBN: 978-3-540-69113-6.

[Int09] International Electrotechnical Commission. *Industrial communication networks - Network and system security: Part 1-1: Terminology, concepts and models*. Geneva, 2009.

[Int13] International Electrotechnical Commission. *Industrial communication networks - Network and system security: Part 3-3: System security requirements and security levels*. Geneva, 2013.

[Int22] International Telecommunication Union. *X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks*. 2022. URL: https://www.itu.int/rec/T-REC-X.509/en (visited on 05/01/2023).

[IT18] IEEE and The Open Group. *at – The Open Group Base Specifications Issue 7, 2018 edition. IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*. 2018. URL: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/at.html (visited on 04/29/2023).

[Jos+22] David Joseph et al. "Transitioning organizations to post-quantum cryptography". In: *Nature* 605.7909 (2022), pp. 237–243. DOI: 10.1038/s41586-022-04623-2.

[Kam+18] Panos Kampanakis et al. *The Viability of Post-quantum X.509 Certificates*. Cryptology ePrint Archive, Paper 2018/063. 2018. URL: https://eprint.iacr.org/2018/063.

[Key22a] Keyfactor. *EJBCA PKI*. Nov. 2022. URL: https://github.com/Keyfactor/ejbca-ce.

[Key22b] Keyfactor, Inc. *Keyfactor Command: Certificate lifecycle automation*. 2022. URL: https://www.keyfactor.com/wp-content/uploads/CLA-Datasheet-Keyfactor.pdf (visited on 07/28/2022).

[KL15a] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Second edition. Chapman & Hall/CRC cryptography and network security. Boca Raton, London, and New York: CRC Press, 2015. ISBN: 9781466570269.

[KL15b] Eric D. Knapp and Joel Thomas Langill. "Establishing Zones and Conduits". In: *Industrial network security*. Ed. by Eric D. Knapp and Joel Thomas Langill. Amsterdam and Heidelberg: Elsevier / Syngress, 2015, pp. 261–281. ISBN: 9780124201149. DOI: 10.1016/B978-0-12-420114-9.00009-5.

[KNT20] Nadim Kobeissi et al. "Verifpal: Cryptographic Protocol Analysis for the Real World". In: *Progress in Cryptology – INDOCRYPT 2020*. Ed. by Karthikeyan Bhargavan et al. Vol. 12578. Springer eBook Collection. Cham: Springer International Publishing and Imprint: Springer, 2020, pp. 151–202. ISBN: 978-3-030-65276-0. DOI: 10.1007/978-3-030-65277-7_8.

[KO22a] Andreas Kretschmer and David von Oheimb. *CMP RA component*. Version 2.2.0, commit 038ad5e. Dec. 2022. URL: https://github.com/siemens/cmp-ra-component.

[KO22b] Andreas Kretschmer and David von Oheimb. *generic CMP client*. Version commit a791495. Nov. 2022. URL: https://github.com/siemens/gencmpclient.

[KO22c] Andreas Kretschmer and David von Oheimb. *Lightweight CMP RA*. Version commit c8eb6ca. Dec. 2022. URL: https://github.com/siemens/LightweightCmpRa.

[Leg22] Legion of the Bouncy Castle Inc. *The Bouncy Castle Crypto Package For Java*. Version 1.72. 2022. URL: https://github.com/bcgit/bc-java/tree/1.72.

[Léo+21] Léo Ducas et al. *CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation (Version 3.1)*. Specification. 2021. URL: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf (visited on 07/30/2022).

[Low96] Gavin Lowe. "Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Tiziana Margaria-Steffen and Bernhard Steffen. Vol. 1055. SpringerLink Bücher. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 147–166. ISBN: 978-3-540-61042-7. DOI: 10.1007/3-540-61042-1_43.

[Low97] Gavin Lowe. "A Hierarchy of Authentication Specifications". In: *Proceedings 10th Computer Security Foundations Workshop*. 1997, pp. 31–43. DOI: 10.1109/CSFW.1997.596782.

[LZK20] Timm Lauser et al. "Security Analysis of Automotive Protocols". In: *Computer Science in Cars Symposium*. Ed. by Björn Brücher. ACM Digital Library. New York,NY,United States: Association for Computing Machinery, 2020, pp. 1–12. ISBN: 9781450376211. DOI: 10.1145/3385958.3430482.

[Mar11] Thilo Sauter Martin Wollschlaeger. "Industrial Internet". In: *Industrial Communication Systems*. Ed. by Bogdan M. Wilamowski and J. David Irwin. 2nd ed. The Industrial Electronics Handbook. CRC Press, 2011. Chap. 56. DOI: 10.1201/b10603-60. URL: https://www.routledgehandbooks.com/doi/10.1201/b10603-60%200.

[Mei+13] Simon Meier et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *Computer Aided Verification*. Ed. by Helmut Veith and Natasha Sharygina. Vol. 8044. SpringerLink Bücher. Berlin, Heidelberg: Springer, 2013, pp. 696–701. ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_48.

[Mos18] Michele Mosca. "Cybersecurity in an Era with Quantum Computers: Will We Be Ready?" In: *IEEE Security & Privacy* 16.5 (2018), pp. 38–41. ISSN: 1540-7993. DOI: 10.1109/MSP.2018.3761723.

[Mot16] Shailesh Mota. "Secure Certificate Management and Device Enrollment at IoT Scale". Master's thesis. Espoo: Aalto University. School of Science, 2016. URL: http://urn.fi/URN:NBN:fi:aalto-201611025260.

[MTG22] MTG AG. *MTG Certificate Lifecycle Manager*. 2022. URL: https://www.mtg.de/export/sites/default/.galleries/documents/de/pdf_online/Flyer/Flyer_MTG_CLM_042022_en.pdf (visited on 07/28/2022).

[NIS22] NIST. *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*. Oct. 2022. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf (visited on 04/21/2023).

[NS78] Roger M. Needham and Michael D. Schroeder. "Using Encryption for Authentication in Large Networks of Computers". In: *Commun. ACM* 21.12 (Dec. 1978), pp. 993–999. ISSN: 0001-0782. DOI: 10.1145/359657.359659. URL: https://doi.org/10.1145/359657.359659.

[NW17] Ruben Niederhagen and Michael Waidner. *Practical Post-Quantum Cryptography: White Paper*. Ed. by Michael Waidner. 2017. URL: https://www.sit.fraunhofer.de/fileadmin/dokumente/studien_und_technical_reports/Practical.PostQuantum.Cryptography_WP_FraunhoferSIT.pdf?_=1503992279 (visited on 05/24/2022).

[Off21] Offenburg University of Applied Sciences. *Project Register Field PKI*. 2021. URL: https://www.hs-offenburg.de/forschung/vorhabenregister/crt-pub/project/show/576 (visited on 07/27/2022).

[Ope22] OpenSSL Software Foundation, Inc. *OpenSSL Project*. 2022. URL: https://github.com/openssl/openssl.

[OS22] David von Oheimb and Benjamin Schilling. *libsecutils*. Version commit 3199ea8. Nov. 2022. URL: https://github.com/siemens/libsecutils.

[Pau+22] Sebastian Paul et al. "Mixed Certificate Chains for the Transition to Post-Quantum Authentication in TLS 1.3". In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '22. Nagasaki, Japan: Association for Computing Machinery, 2022, pp. 727–740. ISBN: 9781450391405. DOI: 10.1145/3488932.3497755. URL: https://doi.org/10.1145/3488932.3497755.

[Pie+20] Pierre-Alain Fouque et al. *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU*. Specification. 2020. URL: https://falcon-sign.info/falcon.pdf (visited on 07/30/2022).

[PO22] Martin Peylo and David von Oheimb. *CMPforOpenSSL (cmpossl)*. Version commit fa1a4ee. Nov. 2022. URL: https://github.com/mpeylo/cmpossl.

[Pro+19] Stefan Profanter et al. "OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols". In: *2019 IEEE International Conference on Industrial Technology (ICIT)*. 2019, pp. 955–962. DOI: 10.1109/ICIT.2019.8755050.

[PYH13] M. Pritikin (Ed.) et al. *Enrollment over Secure Transport*. RFC 7030 (Proposed Standard). RFC. Updated by RFCs 8951, 8996. Fremont, CA, USA: RFC Editor, Oct. 2013. DOI: 10.17487/RFC7030. URL: https://www.rfc-editor.org/rfc/rfc7030.txt.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/rfc/rfc8446.txt.

[San+13] S. Santesson et al. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. RFC 6960 (Proposed Standard). RFC. Updated by RFC 8954. Fremont, CA, USA: RFC Editor, June 2013. DOI: 10.17487/RFC6960. URL: https://www.rfc-editor.org/rfc/rfc6960.txt.

[Sch05] J. Schaad. *Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)*. RFC 4211 (Proposed Standard). RFC. Updated by RFC 9045. Fremont, CA, USA: RFC Editor, Sept. 2005. DOI: 10.17487/RFC4211. URL: https://www.rfc-editor.org/rfc/rfc4211.txt.

[Ser+12] Konstantin Serebryany et al. "AddressSanitizer: A Fast Address Sanity Checker". In: *USENIX ATC 2012*. 2012. URL: https://www.usenix.org/conference/usenixfederatedconferencesweek/addresssanitizer-fast-address-sanity-checker.

[Sho94] P. W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. Ed. by Shafi Goldwasser. Los Alamitos, Calif.: IEEE Computer Soc. Pr, 1994, pp. 124–134. ISBN: 0-8186-6580-7. DOI: 10.1109/SFCS.1994.365700.

[SKD20] Dimitrios Sikeridis et al. "Post-Quantum Authentication in TLS 1.3: A Performance Study". In: *Proceedings 2020 Network and Distributed System Security Symposium*. Ed. by Dongyan Xu and Ahmad-Reza Sadeghi. Reston, VA: Internet Society, 2020. ISBN: 1-891562-61-4. DOI: 10.14722/ndss.2020.24203.

[SLB20] Alparslan Sari et al. "Industrial Networks and IIoT: Now and Future Trends". In: *Industrial IoT : Challenges, Design Principles, Applications, and Security*. Ed. by Ismail Butun. Cham: Springer International Publishing, 2020, pp. 3–55. ISBN: 978-3-030-42500-5. DOI: 10.1007/978-3-030-42500-5_1.

[SM08] J. Schaad and M. Myers. *Certificate Management over CMS (CMC)*. RFC 5272 (Proposed Standard). RFC. Updated by RFC 6402. Fremont, CA, USA: RFC Editor, June 2008. DOI: 10.17487/RFC5272. URL: https://www.rfc-editor.org/rfc/rfc5272.txt.

[SM21] Douglas Stebila and Michele Mosca. *libOQS: C library for prototyping and experimenting with quantum-resistant cryptography*. Open Quantum Safe (OQS) project. Dec. 2021. URL: https://github.com/open-quantum-safe/liboqsl.

[SM22] Douglas Stebila and Michele Mosca. *openSSL: Fork of OpenSSL that includes prototype quantum-resistant algorithms and ciphersuites based on liboqs*. Open Quantum Safe (OQS) project. Jan. 2022. URL: https://github.com/open-quantum-safe/openssl.

[Swe22] Michael Sweet. *ACME-Based Provisioning of IoT Devices*. Internet-Draft draft-sweet-iot-acme-01. Work in Progress. Internet Engineering Task Force, Apr. 2022. 10 pp. URL: https://datatracker.ietf.org/doc/draft-sweet-iot-acme/01/.

[SWW15] Ahmad-Reza Sadeghi et al. “Security and Privacy Challenges in Industrial Internet of Things”. In: *Proceedings of the 52nd Annual Design Automation Conference*. DAC ’15. San Francisco, California: Association for Computing Machinery, 2015. ISBN: 9781450335201. DOI: 10.1145/2744769.2747942. URL: https://doi.org/10.1145/2744769.2747942.

[W3C] W3C. *Web Services Glossary: Web service*. URL: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice (visited on 04/26/2023).

A. Appendix

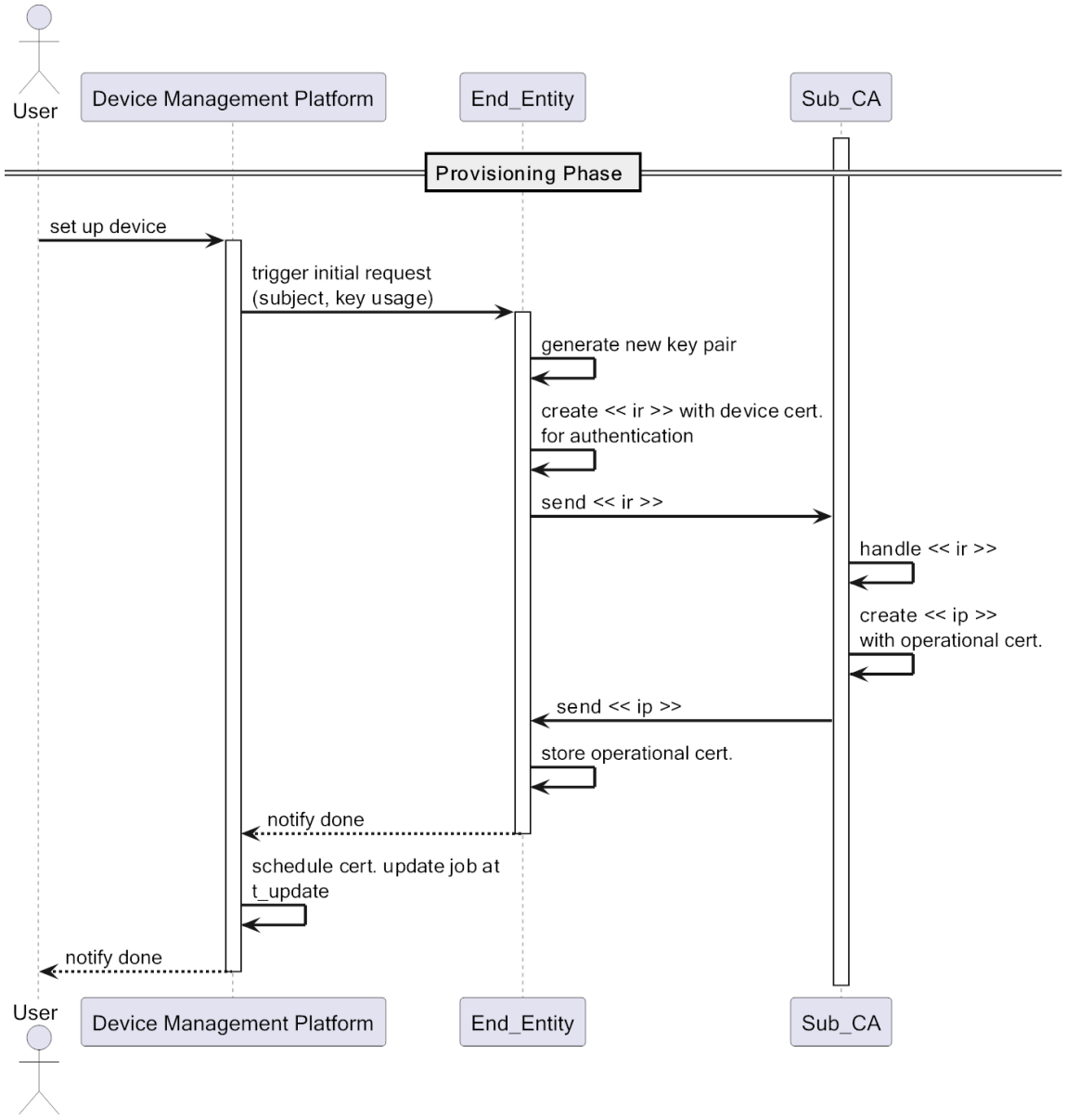


Figure A.1.: Sequence diagram for enrollment of end entity

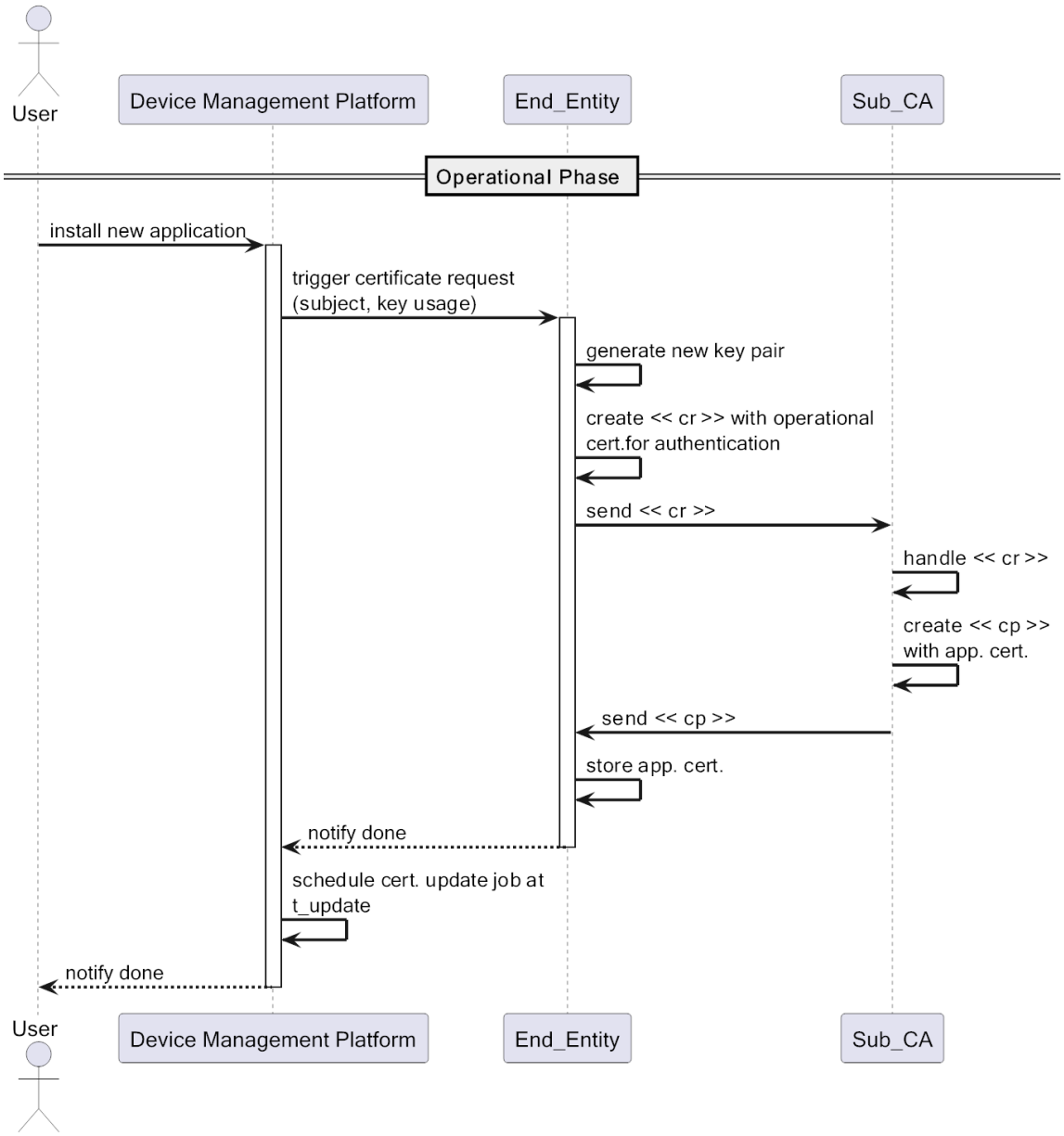


Figure A.2.: Sequence diagram for certificate requests

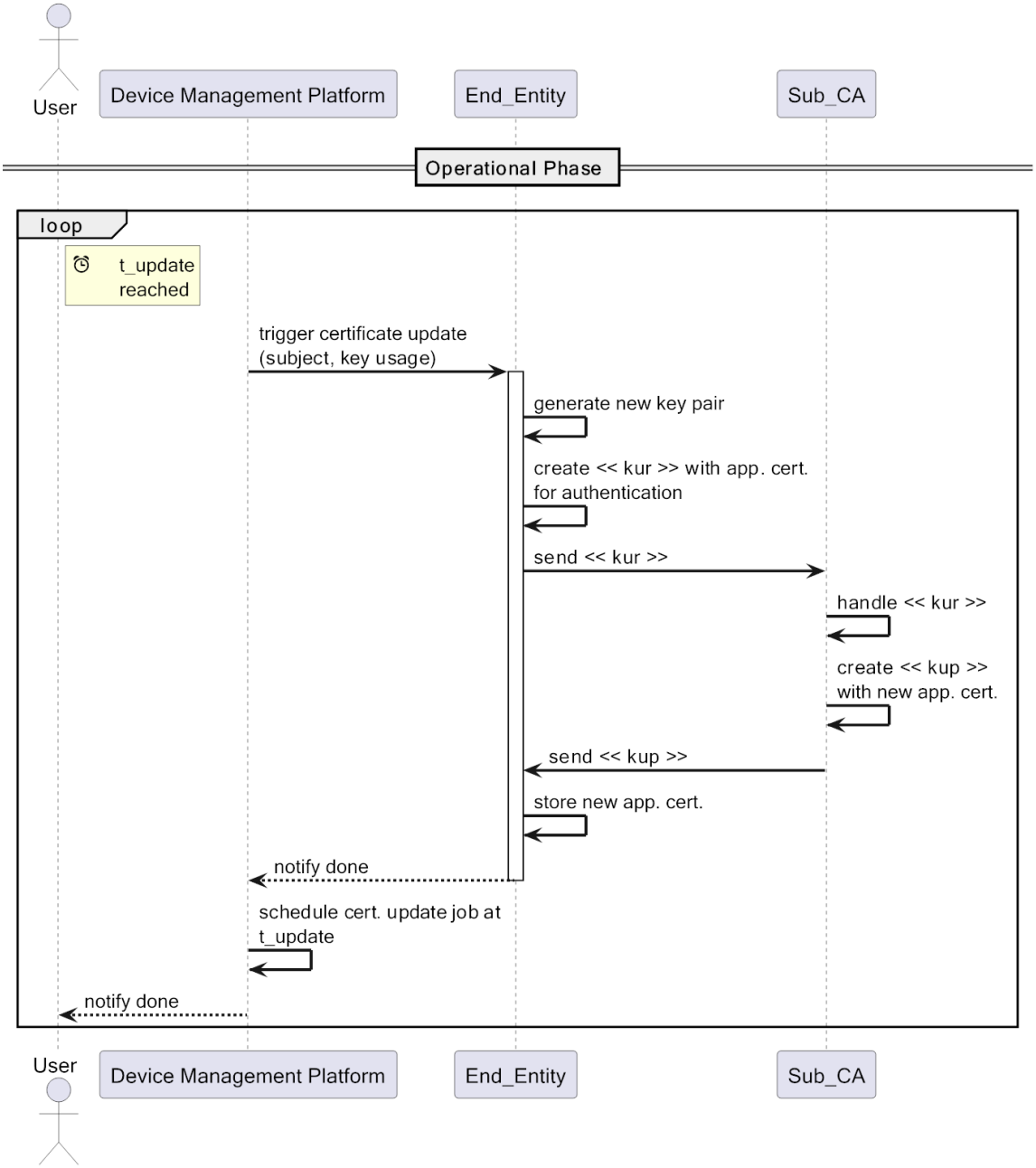


Figure A.3.: Sequence diagram for automated certificate renewal

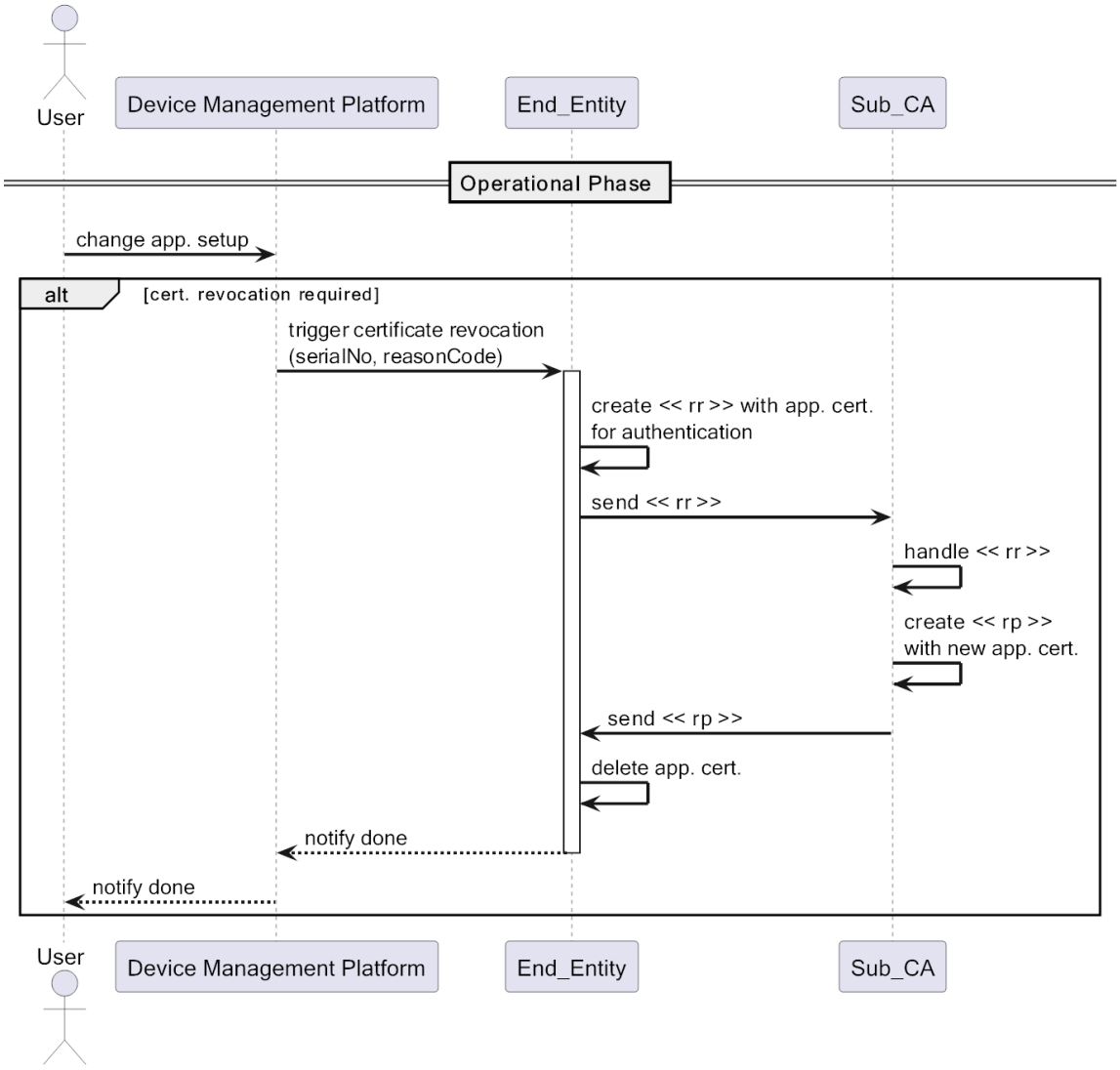


Figure A.4.: Sequence diagram for certificate revocation