PhD-FSTM-2026-061
Faculty of Science, Technology and Medicine

# DISSERTATION

Defence held on 26 June 2026 in Esch-sur-Alzette

to obtain the degree of

# DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

# EN INFORMATIQUE

by

Jan Oupický

# On Migration to Quantum-Safe Cryptography

## Dissertation defence committee

Dr. Peter Y. A. Ryan, Dissertation Supervisor
*Professor, Université du Luxembourg*

Dr. Jean-Sébastien Coron, Chair
*Professor, Université du Luxembourg*

Dr. Ludovic Perret
*Professor, EPITA*

Dr. Peter Browne Rønne
*Research Scientist, Université du Luxembourg*

Dr. Alexander Wiesmaier
*Professor, Darmstadt University of Applied Sciences*

# Affidavit / Statement of originality

I declare that this thesis:

- is the result of my own work. Any contribution from any other party, and any use of generative artificial intelligence technologies have been duly cited and acknowledged;
- is not substantially the same as any other that I have submitted, and;
- is not being concurrently submitted for a degree, diploma or other qualification at the University of Luxembourg or any other University or similar institution except as specified in the text.

With my approval I furthermore confirm the following:

- I have adhered to the rules set out in the University of Luxembourg's Code of Conduct and the Doctoral Education Agreement (DEA), in particular with regard to Research Integrity.
- I have documented all methods, data, and processes truthfully and fully.
- I have mentioned all the significant contributors to the work.
- I am aware that the work may be screened electronically for originality.

I acknowledge that if any issues are raised regarding good research practices based on the review of the thesis, the examination may be postponed pending the outcome of any investigation of such issues. If a degree was conferred, any such subsequently discovered issues may result in the cancellation of the degree.

# Abstract

In 1994, Peter Shor developed a quantum algorithm that can efficiently solve the factorization and discrete logarithm problems. These problems are the foundation of modern cryptographic algorithms and the existence of an efficient algorithm for solving them threatens their security. Although large-scale quantum computers capable of breaking current cryptographic algorithms are not yet available, advances in quantum technology and the risk of "store now, decrypt later" attacks make it essential to prepare for migration to quantum-safe cryptography today.

For the past decade, the cryptographic community has been actively developing algorithms that are not exploitable by Shor's algorithm; such algorithms are said to be "quantum-safe" or "post-quantum." However, cryptographic algorithms do not exist in isolation—they are components of larger cryptographic systems, such as protocols or other cryptographic primitives. Migrating these systems to use quantum-safe algorithms is therefore a complex task. This thesis studies the migration of several widely used cryptographic systems to post-quantum cryptography.

In particular, we consider the migration of the public-key infrastructure (PKI), the TLS protocol, the XML Encryption and XML Signature standards, the SAML security framework, the Advanced Electronic Signature (AdES) standard, and the password-authenticated public-key encryption (PAPKE) primitive. For each topic, we analyze how the presence of quantum adversaries affects the security of existing constructions and identify components that rely on cryptographic assumptions vulnerable to quantum attacks. Based on this analysis, we propose approaches for adapting these systems to use post-quantum cryptographic algorithms.

The proposed solutions are considered in two main forms: purely post-quantum and hybrid post-quantum. Purely post-quantum approaches rely exclusively on post-quantum algorithms, while hybrid post-quantum approaches combine classical and post-quantum mechanisms in order to provide security as long as at least one of the underlying assumptions remains secure. Hybrid post-quantum approaches are primarily motivated by uncertainty regarding the security of post-quantum algorithms. For each proposal, we discuss the advantages and disadvantages of the design, including deployment complexity, compatibility with existing systems, and performance considerations. In several cases, we also provide proof-of-concept implementations and evaluate their performance through benchmarks.

We study post-quantum public-key infrastructures, which form a fundamental component of many other cryptographic systems. We show that constructing a purely post-quantum PKI does not fundamentally differ from constructing a classical PKI. For hybrid deployments, we propose two approaches—the composite hybrid and the parallel hybrid—which differ primarily in their implementation complexity. We also survey and analyze existing proposals for post-quantum TLS. In contrast, systems such as XML encryption and XML signatures, the SAML security framework, and the eIDAS-related AdES standard have received little attention in the

context of post-quantum migration. For these systems, we outline possible migration approaches and provide proof-of-concept implementations.

In addition, we present an example migration plan for a trust service provider that relies on all of the discussed cryptographic systems, together with the methodology used to develop it. Finally, we consider the migration of the cryptographic primitive PAPKE. We propose a novel construction that replaces an ideal cipher with a half-ideal cipher, resulting in more efficient quantum-safe constructions and motivating the study of a new public-key encryption property called decryption robustness.

# Acknowledgments

I would like to begin by thanking my supervisor, Prof. Dr. Peter Y. A. Ryan, for providing me with the opportunity to conduct this research under his supervision, for his guidance throughout my doctoral studies, and for establishing the APSIA group and fostering an inspiring and friendly environment, especially on Fridays.

I would like to give special thanks to my daily supervisors Dr. Johannes Müller and Dr. Peter Browne Rønne. From the beginning of my studies, Johannes' mentorship has helped me become the researcher I am today, and Peter's guidance has been equally invaluable. I could not have asked for more encouraging, helpful, and kind daily supervisors.

Furthermore, I am deeply grateful to Mr. Thomas Kopp, Chief Scientist at LuxTrust, for sharing his expertise and providing essential feedback as part of our industrial partnership during my PhD. His industry experience was invaluable, and I am honored by his participation on my defense jury.

Moreover, I would like to thank my co-authors on the publications included in this thesis. Thank you for insightful discussions, shared efforts, and the privilege of working alongside you. I owe a special thanks to Dr. Marjan Škrobot and Dr. Afonso Arriaga, with whom I spent endless hours going over our proofs, debugging our work, and enjoying conversations beyond our research.

I am also deeply grateful to Dr. Yan Kim, not only for sharing his experiences of going through this journey just before me, but also for being the best roommate I could have asked for.

Moreover, I would like to thank the rest of the APSIA group members and alumni: Dr. Rafieh Mosaheb, Dr. Ivo Maffei, Dr. Dimiter Ostrev, Dr. Jordan Naccache, Dr. David Mestel, Dr. Marius Lombard-Platet, and Dr. Aditya Damodaran. Thank you for the numerous insightful discussions over lunch and coffee, and for providing such a welcoming and supportive environment.

Last but not least, I would like to express my deepest gratitude to my family and friends for their unwavering support and for keeping me sane throughout this journey.

---

I acknowledge the use of large language models, which were used exclusively for grammar checking and stylistic refinement during the writing process.

“*dbbe5c094e87048d6c46c4df2e9813bfb2b359c4857369cea4d22740235d12d826ceb64e*”

# Contents


**1 Introduction** **1**
- 1.1 A Brief History of Cryptography . . . 2
- 1.2 Cryptography Around Us . . . 3
- 1.3 Hard Problems and Cryptography . . . 4
- 1.4 Quantum Computers and Cryptography . . . 7
- 1.5 Post-Quantum Cryptography . . . 13
- 1.6 Post-Quantum Cryptography vs. Quantum Cryptography . . . 14
- 1.7 Migration to Post-Quantum Cryptography . . . 15
- 1.8 Structure . . . 17
- 1.9 Contributions . . . 19

**2 Preliminaries** **21**
- 2.1 Notation . . . 21
- 2.2 Symmetric Cryptography . . . 23
- 2.3 Asymmetric Cryptography . . . 27
- 2.4 Post-Quantum Cryptography in Detail . . . 32
- 2.5 Threat Model . . . 48

**3 Post-Quantum Public-Key Infrastructure** **49**
- 3.1 Public-Key Infrastructure Components . . . 50
- 3.2 Purely Post-Quantum Public-Key Infrastructure . . . 76
- 3.3 Hybrid Post-Quantum Public-Key Infrastructure . . . 86
- 3.4 Conclusion . . . 95

**4 Post-Quantum TLS** **97**
- 4.1 Overview of TLS . . . 98
- 4.2 Post-Quantum Authentication in TLS . . . 102
- 4.3 Post-Quantum Key Exchange in TLS . . . 107
- 4.4 Post-Quantum Authenticated Key Exchange in TLS . . . 113
- 4.5 Symmetric Primitives in Post-Quantum TLS . . . 117
- 4.6 Post-Quantum TLS Performance Experiment . . . 118
- 4.7 Conclusion . . . 126

**5 Post-Quantum XML and SAML** **127**
5.1 XML Security . . . 128
5.2 Post-Quantum XML Security . . . 135
5.3 SAML Framework . . . 143
5.4 Post-Quantum SAML SSO . . . 147
5.5 Conclusion . . . 153

**6 Post-Quantum Advanced Electronic Signatures** **155**
6.1 Advanced Electronic Signatures . . . 156
6.2 Purely Post-Quantum Advanced Electronic Signatures . . . 182
6.3 Hybrid Post-Quantum Advanced Electronic Signatures . . . 185
6.4 Performance of Post-Quantum AdES . . . 191
6.5 Conclusion . . . 196

**7 Post-Quantum Migration Planning** **199**
7.1 Existing Migration Guidance . . . 200
7.2 Component Evaluation . . . 201
7.3 Migration Roadmap . . . 206
7.4 Compliance Aspects . . . 210
7.5 Eventual Migration to Purely Post-Quantum Cryptography . . . 210
7.6 Conclusion . . . 211

**8 Post-Quantum Password-Authenticated Public-Key Encryption** **213**
8.1 Decryption Robustness . . . 215
8.2 Password-Authenticated Public-Key Encryption . . . 229
8.3 PAPKE-HIC: Post-Quantum PAPKE . . . 244
8.4 PAPKE-HIC Performance . . . 261
8.5 Conclusion . . . 267

**9 Conclusion and Outlook** **271**
9.1 Conclusion . . . 271
9.2 Outlook . . . 272

*Bibliography* 277

# Chapter 1

# Introduction


1.1 A Brief History of Cryptography . . . 2
1.2 Cryptography Around Us . . . 3
1.3 Hard Problems and Cryptography . . . 4
1.3.1 The Integer Factorization Problem . . . 5
1.3.2 The Discrete Logarithm Problem . . . 6
1.3.3 Generalization: The Hidden Subgroup Problem . . . 6
1.4 Quantum Computers and Cryptography . . . 7
1.4.1 Impact of Shor's Algorithm . . . 8
1.4.2 Impact of Grover's Algorithm . . . 10
1.4.3 Store Now, Decrypt Later . . . 12
1.5 Post-Quantum Cryptography . . . 13
1.6 Post-Quantum Cryptography vs. Quantum Cryptography . . . 14
1.7 Migration to Post-Quantum Cryptography . . . 15
1.8 Structure . . . 17
1.9 Contributions . . . 19
1.9.1 LuxTrust Project . . . 19
1.9.2 Publications . . . 19


Whenever I talk to my friends and family about my work, I am reminded that most people outside my hyper-specific field are unaware that cryptography is omnipresent in our lives. This illustrates that cryptographers are not doing a good enough job of advertising their research contributions to society. With that in mind, this introduction strives to be accessible to readers with a minimal background in cryptography.

In this chapter, our goal is to outline the motivation behind the core topic of this thesis—the migration to quantum-safe cryptography. We provide a high-level overview of cryptography as a whole and discuss the impact of quantum computers on our current (classical) cryptographic

systems. We then define what quantum-safe (or post-quantum) cryptography means and subsequently explain what the migration process actually entails. We conclude by presenting the overall structure of this thesis and summarizing its main contributions.

## 1.1 A Brief History of Cryptography

The typical cryptography textbook or course begins with several examples of how cryptography was used historically. The most prominent example—Scytale—comes from ancient Greece. Scytale is a wooden stick together with a narrow paper strip. The paper strip is wrapped around the stick and then the sender writes a message along the stick. Once unwrapped, the paper strip contains letters, but they are seemingly scattered randomly across the tape. To recover the original message, the recipient must wrap the paper strip around a wooden stick of the same dimensions in the same manner.

Scytale is an example of a *cipher*—a method for hiding a message, so that nobody except the intended recipient can recover the message. The process of hiding the message is called *encryption* and the process of recovering the message is called *decryption*. The stick is the tool that allows the recipient to decrypt the message; we call such a tool the *decryption key*. It is "hard" for anyone without the key to recover the message. The tool used by the sender is called the *encryption key*. Observe that the encryption key is the same as the decryption key. This makes Scytale a *symmetric cipher*. If the keys differed, then it would be called an *asymmetric cipher*. Cryptography also has a name for the message itself: it is called the *plaintext*, and the scrambled message is called the *ciphertext*.

Scytale is a nice example because the encryption and decryption processes are visual and tangible. In that sense, it is unfortunate that more advanced ciphers lose this property as they become more complex. Another notable cipher is the Enigma machine used by Nazi Germany during World War II. The Enigma is an electromechanical device rather than a stick (as expected after millennia of progress) and its inner workings are already too complex to be explained in a few sentences. The complexity of the Enigma is illustrated by the amount of effort it took the Allies to crack it.[1] By "cracking the Enigma," we mean that the ciphertext can be decrypted without knowing the decryption key, which in this case was not a stick, but one of the $\approx 2^{67}$ possible internal configurations of the Enigma [Ral24].

The one-time pad is maybe the most timeless example of a symmetric cipher because of its *perfect* security properties, on which we elaborate further in Section 1.3. To encrypt a message consisting of $n$ bits, a one-time pad requires a key of the same length. To encrypt the first message bit, the sender "XORs it"[2] with the first key bit. Other message bits are encrypted analogously bit by bit. The decryption procedure is the same except the recipient XORs the ciphertext with the key. The obvious drawbacks of the one-time pad are that the key must be the same size as the message and that the key cannot be reused without compromising security.

Scytale, the Enigma, and the one-time pad are all examples of symmetric ciphers, i.e., the same key is used for encryption and decryption. Asymmetric cryptography was invented in

[1] The story is fairly accurately told in the movie "The Imitation Game." However, the movie has been criticized for insufficiently portraying the essential role played by Polish cryptographers in breaking the Enigma.

[2] "To XOR" refers to applying the *eXclusive OR* bit operation. The result is 1 if exactly one of the bits is 1, and 0 otherwise.

the 1970s by Diffie and Hellman [DH76].[3] They proposed a method, called Diffie-Hellman key exchange, by which two people, whom we will call Alice and Bob, can agree on a secret key while communicating over an *insecure channel*. Agreeing on a secret key is straightforward if we assume that nobody can hear Alice or Bob talking—one of them chooses the secret key and tells it to the other. The breakthrough of the Diffie-Hellman key exchange is that Alice and Bob can agree on the secret key even when we consider that someone, whom we will call Eve, eavesdrops on their conversation.

The Diffie-Hellman key exchange is fairly simple if we abstract away the mathematical details. Alice and Bob both run what is called a *key generation algorithm*. The key generation algorithm outputs two values: a secret key $sk$ and a public key $pk$. Alice and Bob exchange their *public* keys ($pk_A$ and $pk_B$) over the insecure channel. Finally, they run a *key derivation algorithm*, denoted as `Derive`, which takes as inputs a secret key $sk$ and a public key $pk$ and outputs a shared key. The essential property of `Derive` is that it outputs the same shared key for two pairs of inputs $(sk_A, pk_B)$ (Alice's calculation) *and* $(sk_B, pk_A)$ (Bob's calculation) all while Eve has no way of deriving the same shared key from the public values $pk_A$ and $pk_B$, which she has intercepted.

This is the basis of asymmetric cryptography—the public keys are not secret; they can be made *public*. This is why asymmetric cryptography is also called *public-key cryptography*. The term *secret-key cryptography* is sometimes used to refer to symmetric cryptography. Another landmark cryptographic algorithm is the RSA algorithm, invented a few years after the Diffie-Hellman key exchange. RSA is named after its inventors Rivest, Shamir, and Adleman [RSA78]. Notably, RSA can be used to encrypt messages as well as to *digitally sign* messages. Conceptually, digital signatures[4] work in the same way as wet (ink) signatures: a *signer* applies their signature on a message, and this signature serves as proof that the message was approved by the signer, which can later be verified by a *verifier*. However, the guarantees that the signature is not forged differ. Wet signatures rely on the hardness of physically reproducing someone's handwriting, whereas digital signatures rely on the hardness of a mathematical problem; see Section 1.3 for an overview of such problems.

If we ignore the recent adoption of *post-quantum* algorithms (the topic of this thesis), all the *public-key* cryptography used today is some variant of Diffie-Hellman or RSA.

## 1.2 Cryptography Around Us

We dare to say that an overwhelming majority of people use cryptography daily, and they do not even know about it, which is a good thing! People interact with cryptography as soon as they wake up—by checking their phone [Del25]. The simple act of checking the news, sending a message to your family and friends via WhatsApp [Wha24] or iMessage [Ste24], or reading your emails involves cryptography.[5]

Let us elaborate on where cryptography comes into play in the act of "checking the news."

[3] It was revealed later that a similar algorithm was invented in secret a few years prior by the British intelligence agency GCHQ [Ell70].

[4] Not to be confused with general *electronic signatures*, which often simply refer to digitized images of wet signatures. The security of such electronic signatures does not rely on hard mathematical problems, and they are arguably easier to forge than wet signatures.

[5] Even opening an app on the phone involves cryptography since the app's files are encrypted.

For example, consider that the news website in question is The New York Times. The connection between https://www.nytimes.com/ and your phone uses the TLS protocol [Res18], which we discuss in detail in Chapter 4. The purpose of TLS is to establish a secure channel between a client (your phone) and a server (https://www.nytimes.com/). A "secure channel" means a communication channel where nobody except the two participants can read, send or modify transferred messages. In this context, it provides the following assurances:

- Authenticity and Integrity: You can be certain that what you are reading is really The New York Times and not some malicious website impersonating https://www.nytimes.com/.
- Confidentiality: Nobody, except the server, knows what news articles you read.[6]

Under the hood, this is achieved using cryptography as TLS uses Diffie-Hellman (and potentially RSA) together with constructions from symmetric cryptography.

A similar story unfolds when you message someone via WhatsApp, except it utilizes a different protocol to achieve similar goals. Messaging apps such as WhatsApp, iMessage, or Signal [MP16], are *end-to-end encrypted*. End-to-end encryption refers to the security property that even though messages are relayed by some intermediate servers (in the case of WhatsApp the servers are owned by Meta), they cannot be modified or read by anyone except the sender and the receiver. Note that this sounds conceptually similar to what TLS provides for the client and the server. However, the TLS protocol is not suited for client-to-client messaging as it was not designed for asynchronous communication between two clients. Additionally, protocols used in messaging apps offer further features, such as group messaging.

Apart from the internet-focused examples above, cryptography is also used in credit cards, passports, access cards/key fobs you use to unlock the doors at home or at work. Cryptography plays a role in securing your computer or phone from viruses—operating systems may require all software to be digitally signed by the creator to prevent execution of unauthorized software. Of course, this is not an exhaustive list, but it hopefully illustrates that cryptography is truly omnipresent in our lives.

## 1.3 Hard Problems and Cryptography

It may come as a surprise that the majority of cryptographic constructions used in practice have no "guarantee of security" in the sense that they are considered secure only because nobody has yet shown how to break them.

However, there is a category of cryptographic algorithms which are so-called *information-theoretically secure*, such as the above-mentioned one-time pad. If a scheme is information-theoretically secure, it means that no matter how many computational resources you use, you cannot break its security. The one-time pad cannot be broken because if we are given a ciphertext, then for any possible message there exists a key that decrypts to it, i.e., we cannot deduce any meaningful information about the encrypted message from the ciphertext.[7] Of course, if we know,

[6] The website domain is not protected; it is either leaked by DNS or SNI in TLS 1.3 [Res18]. Eve can still learn that you read The New York Times, but she cannot know which articles you read.

[7] The hexadecimal quotation found in the front matter can decrypt to anything. For example, try the key `929e326c27f36ce81e66afb141ef33d1ddc179b0ed1a07a584a64f21577d5bf84da0d939` and the key `88d1316c6ef776e81823aaab47f766cc92c22cabf1121da7cbbc096e0d733cf608e09860`.

e.g., that the underlying message is in English and has 10 words, we can discard some keys that decrypt to nonsense; however, we will still be left with keys decrypting to any possible English message of 10 words. This is different from, e.g., the Enigma, where the majority of keys can be discarded because they will decrypt to nonsense.

Unfortunately, information-theoretic security comes with the cost of being impractical. One-time pad keys grow linearly with the size of the encrypted message, whereas, e.g., the Enigma's key is of constant size. In addition, the one-time pad key cannot be reused because XORing two ciphertexts encrypted with the same key reveals the XORed plaintext messages.

This is why the majority of cryptographic constructions are only *computationally secure*, which means that they can be technically broken, but only by using an extensive amount of computational resources. Computational security relies on certain *hardness assumptions*, which are hypotheses that some problems are hard to solve. By "hard to solve," we mean that the amount of computation required to solve them increases exponentially with the size of the input.

Let us first present the two most important hard problems in (public-key) cryptography (Sections 1.3.1 and 1.3.2), and then, in Section 1.4, we discuss how they are not actually hard in the world of quantum computers.

### 1.3.1 The Integer Factorization Problem

The *integer factorization problem* is stated as follows: Let $p$ and $q$ be prime numbers, given only their product $n = p \cdot q$, recover $p$ and $q$. Anyone can quickly solve the problem when $n = 143$ (the solution is $p = 11, q = 13$ since $143 = 11 \cdot 13$), but solving the problem when $n$ is a 4000-bit number is thought to be impossible in our lifetimes using classical computers.

That is because the fastest known *classical* algorithm for solving the integer factorization problem is the *General Number Field Sieve* (GNFS) [LL93], which is presumed to have *near*-exponential complexity.[8] It means that it would take millennia to factor a 4000-bit number even on the most powerful computers today. More importantly, exponential complexity allows us to increase parameters slightly while gaining exponentially more security, e.g., factoring a 5000-bit number is $\approx 2^{17}\times$ more difficult than factoring a 4000-bit one—increasing the input size only 1.25× makes the problem $2^{17}\times$ harder.

In certain special cases, however, the integer factorization problem is easy even for large $n$. For example, if $p$ and $q$ are close to each other, $n$ can be factored efficiently[9] using Fermat's factorization method [Böc23]. However, on average (for random $p$ and $q$) the problem is hard.

Notably, the security of RSA relies on the hardness of the integer factorization problem. Specifically, it has been proven that by breaking RSA, one can solve a related "RSA problem," which is also assumed to be hard. It can be easily shown that if one can factor integers, then one can break RSA. However, the other implication—if one can break RSA, then one can factor integers—is an open problem. Ideally, we would want the other implication to hold to establish the security of RSA, because then one could say "RSA is safe because factoring integers is assumed to be hard." There is such a cryptosystem invented by Rabin [Rab79], but it is not used much in practice; this is often attributed in part to the fact that RSA was invented first.[10]

[8] Technically, it is *presumed* to have *sub-exponential* complexity, but for illustrative purposes we omit this distinction. Note that there is also no proof for the complexity.

[9] The complexity is roughly polynomial in $|p - q|$.

[10] Katz and Lindell [KL20] call it a "historical accident."

### 1.3.2 The Discrete Logarithm Problem

The other important problem in public-key cryptography is the *Discrete Logarithm Problem* (DLP). The problem is stated as follows: Let $G$ be a cyclic group with a generator $g \in G = \langle g \rangle$, given $y \in G$, find $x$ s.t. $y = g^x \in G$.

In some groups, such as $G = (\mathbb{Z}, +)$, the discrete logarithm is easy since for $g = 1$ and any $y = g^x = x \cdot g = x \cdot 1 = x$, i.e., the discrete logarithm w.r.t. $g = 1$ of $y$ is $y$ in $G = (\mathbb{Z}, +)$. Groups in which the discrete logarithm is hard are prime-order subgroups of $\mathbb{F}_p^*$ or elliptic curves over $\mathbb{F}_p$, where $\mathbb{F}_p$ denotes a finite field of prime order $p$.

There is a slight difference in the difficulty of DLP in $\mathbb{F}_p^*$ (referred to, a bit imprecisely, as DLP over finite fields), and DLP in elliptic curves over $\mathbb{F}_p$ (referred to as DLP over elliptic curves). The fastest known algorithms for solving DLP over *finite fields*, such as the index calculus [Adl79] and the function field sieve [AH99], have sub-exponential complexity, which is the same complexity as for factoring (Section 1.3.1). However, the fastest algorithms for solving DLP over *elliptic curves*, such as Pollard's rho algorithm [Pol75], have exponential complexity. This is why elliptic curve-based algorithms can work in smaller fields, which brings efficiency, compared to finite field-based algorithms. To achieve the same level of security (128 bits), algorithms based on DLP over elliptic curves work with finite fields of size 256 bits whereas algorithms based on DLP over finite fields work with finite fields of size 3200 bits [BSI26].

The following section (Section 1.3.3) is slightly more technical, since it formulates the previous problems in a more general setting.

### 1.3.3 Generalization: The Hidden Subgroup Problem

The Integer Factorization Problem and the Discrete Logarithm Problem are actually instances of a more general problem called the *Hidden Subgroup Problem* (HSP), which is defined as follows: Let $G$ be a group and $H \leq G$ its subgroup, $X$ be a set and let there be a function $f : G \to X$ s.t. $\forall g_1, g_2 \in G : f(g_1) = f(g_2) \iff g_1 H = g_2 H$. Given $G, X$ and *oracle access* to $f$, find the generating set of $H$.

**Remark**

In cryptography, having "oracle access" to some function refers to the ability to interact with a "black box" that computes such a function, i.e., the problem solver, typically referred to as *the adversary*, can submit inputs to the black box and receive the corresponding outputs.

The reduction of DLP to HSP is as follows. We are given a group $G'$ of size $p$, its generator $g \in G'$ and $h \in G'$ and our goal is to find $x \in \mathbb{Z}_p : g^x = h \in G'$. We solve the HSP where $G = \mathbb{Z}_p^2, X = G', f : \mathbb{Z}_p^2 \to G', (a, b) \mapsto g^a \cdot h^b = g^{a+b \cdot x}$. Let $H = \{(c, d) \in \mathbb{Z}_p^2 : c + d \cdot x = 0\}$. We need to verify that $f$ and $H$ satisfy $\forall g_1, g_2 \in G : f(g_1) = f(g_2) \iff g_1 H = g_2 H$.

$g_1 H = g_2 H \implies f(g_1) = f(g_2)$: Denote $g_1 = (c_1, d_1), g_2 = (c_2, d_2)$, we need to verify that $g^{c_1 + d_1 \cdot x} = g^{c_2 + d_2 \cdot x}$ given that $(c_1, d_1) + H = (c_2, d_2) + H$. The coset equality means that for any $(a_1, b_1) \in H$ there exists $(a_2, b_2) \in H$ s.t. $c_1 + a_1 = c_2 + a_2$ and $d_1 + b_1 = d_2 + b_2$. We use these equalities to substitute for $c_1$ and $d_1$:

$$g^{c_1 + d_1 \cdot x} = g^{(c_2 + a_2 - a_1) + (d_2 + b_2 - b_1) \cdot x} = g^{(c_2 + d_2 \cdot x) - (a_1 + b_1 \cdot x) + (a_2 + b_2 \cdot x)} = g^{c_2 + d_2 \cdot x}.$$

In the last equality we used the definition of $H$.

$f(g_1) = f(g_2) \implies g_1H = g_2H$: Denote $g_1 = (c_1, d_1), g_2 = (c_2, d_2)$, we know that $g^{c_1+d_1 \cdot x} = g^{c_2+d_2 \cdot x}$ which implies that $c_1 + d_1 \cdot x = c_2 + d_2 \cdot x \in \mathbb{Z}_p$. We must show that any element from $(c_1, d_1) + H$ can be expressed as an element of $(c_2, d_2) + H$ and vice versa. Let $(c_1 + a_1, d_1 + b_1) \in (c_1, d_1) + H$, then $a_1 + b_1 \cdot x = 0$. Let $a_2 = a_1 + c_1 - c_2, b_2 = b_1 + d_1 - d_2$, then it can be easily shown that $(a_2 + c_2, b_2 + d_2) \in (c_2, d_2) + H$ using the assumption. Proving the other direction is analogous.

We have shown that this is a valid instance of the hidden subgroup problem, which we assume we can solve. The HSP solution is the generating set of $H$, which in this case is also a one-dimensional vector space over $\mathbb{Z}_p$, i.e., we get $(s, t) \in \mathbb{Z}_p^2$ as the basis of $H$. Since $(x, -1) \in H$, we recover $x = \frac{s}{-t}$.

The reduction of the integer factorization to HSP is Shor's algorithm (Section 1.4.1).

## 1.4 Quantum Computers and Cryptography

This section introduces the motivation behind the contents of this thesis. The motivation is *quantum computers*, or rather, their impact on cryptography. Simply put, a quantum computer is a more powerful variant of our contemporary *classical* computers. Describing how classical computers work, let alone the quantum ones, is beyond the scope of this thesis.

However, we do not have to understand why and how they work to assess their impact. One fundamental difference is that classical computers work with *bits* and quantum computers work with *qubits*. A bit can only be in two states: 0, or 1. A qubit can be in a so-called *superposition* of 0 and 1, and there are infinitely many possible superpositions. Classical computers take bits as input, apply some *classical algorithm* to those bits, and output the result. Quantum computers work analogously except they work with qubits and *quantum algorithms*. Classical algorithms fundamentally consist of *classical* logic gates (or operations) and quantum algorithms consist of *quantum* logic gates. Quantum logic gates are a superset of classical logic gates, which implies that quantum computers can do everything that classical computers can, and more.

The number of qubits that a quantum computer can compute with is the primary metric used to evaluate how powerful the quantum computer is. One must differentiate between two types of qubits: logical qubits and physical qubits. A *logical qubit* is an idealized mathematical model of a qubit used in theory, e.g., during the design of quantum algorithms. A *physical qubit* is a real physical object, which is used to construct the logical qubit. It is estimated that in practice 10–1000 [Gid+23; Web+26; Cai+26] physical qubits are needed to construct one logical qubit. This is because physical qubits are very unstable. Importantly, logical qubits are constructed using quantum error-correction codes, and the specific number needed also depends on the "quality" of the logical qubit, which is determined by properties such as its error rate (the likelihood of errors occurring during operations).

In late 2025, the state-of-the-art quantum computers have at most hundreds of *physical* qubits corresponding to a varying number of *logical* qubits. Quantinuum's quantum computer has 32 physical qubits corresponding to 4 logical qubits [Qua24], Google's Willow chip has 105 physical qubits corresponding to 1 logical qubit [Goo24] and IBM's latest quantum chips Nighthawk [IBM25b] and Heron R2 [IBM24] have 120 and 156 physical qubits, respectively. IBM has also published a roadmap that outlines the future development of their quantum computers [IBM25c]. They predict that by 2029 they will possess a quantum computer with 200 *logical*

qubits and 100 million gates [IBM25a]. *Gate count* is another metric that measures the complexity of the quantum algorithms the computer is able to run. However, it is difficult to directly compare these metrics since there are several approaches to building quantum computers with various tradeoffs [BSI25]. We are currently in an era analogous to early computing in the 1940s–1950s, when several approaches to building classical computers were explored. It remains an open question whether a single architecture will ultimately prevail.

We have introduced quantum computers and mentioned that they are "more powerful" than classical computers. Now we present the evidence for this claim, which are two quantum algorithms named after their inventors: Shor's algorithm (Section 1.4.1) and Grover's algorithm (Section 1.4.2).

### 1.4.1 Impact of Shor's Algorithm

Recall from Section 1.3 that two of the most important problems in (public-key) cryptography are the *integer factorization problem* (Section 1.3.1) and the *discrete logarithm problem* (Section 1.3.2). Both of these problems are instances of the *hidden subgroup problem* (Section 1.3.3).

Shor's algorithm [Sho94] was invented in 1994 by Peter Shor. It is a quantum algorithm for the hidden subgroup problem. Note that "Shor's algorithm" typically refers to the integer factorization variant of the algorithm; however, we use it as an umbrella term for both: the integer factorization algorithm and the discrete logarithm algorithm.

The breakthrough is that this algorithm runs in *polynomial time*, i.e., it is exponentially faster[11] than the best known classical algorithms. It being a polynomial time algorithm means that increasing parameters, such as choosing larger groups or primes, does not help—see [Ber+17] for an exploration of "Shor-resistant" parameters for RSA.

The main "downside" of Shor's algorithm is that it is a *quantum* algorithm, i.e., it cannot be run on classical computers. Even though there exist quantum computers, these computers are not powerful enough to be able to run Shor's algorithm on any relevant inputs, such as numbers with thousands of bits.

The term *Cryptographically Relevant Quantum Computer* (CRQC) [NSA24] is sometimes used for quantum computers that will be able to run Shor's algorithm on cryptographically relevant inputs. Since contemporary quantum computers cannot do much in terms of cryptanalysis, by "quantum computer" we mean a CRQC unless stated otherwise. There is no exact definition of a CRQC since the number of required resources to run Shor's algorithm on cryptographically relevant inputs is constantly changing; see Section 1.4.1.1 for details.

Although nobody can predict the future, the current consensus seems to be that CRQCs will materialize sometime in the 2030s or the 2040s. In 2025, BSI stated [BSI25]: "Our conservative estimate is that cryptographically relevant quantum computers are likely to be available within 15 years." Extrapolating from NSA's migration timeline [NSA22], the NSA expects a similar date. Every year, the Global Risk Institute surveys experts in the quantum computing field on their predictions on when CRQCs will materialize. In the latest report from early 2026, almost 70% of respondents think there is at least a 50% chance that CRQCs will be developed in the next 15 years [MP26].

[11] Strictly speaking, this is a simplification because, as discussed in Section 1.3, the best known classical algorithms for integer factorization and finite-field DLP are sub-exponential.

#### 1.4.1.1 Quantum Resource Estimates

To be able to assess what constitutes a cryptographically relevant quantum computer (CRQC), we must know the resources required for running Shor's algorithm on cryptographically relevant inputs. In practice, these inputs are 2048-bit RSA public keys regarding factorization and 256-bit elliptic curve public keys regarding discrete logarithm computation. The required resources can be expressed in terms of memory and time.

By memory, we mean the number of logical qubits and, consequently, the number of physical qubits required to represent these logical qubits. Since logical qubits are constructed from physical qubits using quantum error-correction codes, the encoding efficiency depends on the type of code used. For example, *surface codes* require more physical qubits per logical qubit than *qLDPC codes*. On the other hand, surface codes have been tested experimentally whereas qLDPC codes have not, to the best of our knowledge. Therefore, one must be cautious when comparing results from different publications; for example, 1 000 000 physical qubits on one architecture may be easier to achieve than 10 000 on a different architecture.

By time, we mean the runtime required to compute the solution. The runtime is related to the number of quantum gates required to implement the algorithm since it can be assumed that each gate processing takes a certain amount of time. However, the translation from gate count to runtime is not straightforward and may vary between different quantum computer architectures or gate types. For example, the authors of [Bab+26] distinguish between "fast-clock" CRQCs based on superconducting qubits and "slow-clock" CRQCs based on neutral atoms. Consequently, it is difficult to compare results that consider different quantum computer architectures.

Furthermore, based on the results from [Bab+26; Gid25; Web+26; Cai+26], we observe that there is a general space-time tradeoff between the number of qubits and the runtime—the faster the runtime, the more qubits are needed. In the following discussion, we present the most extreme results: the ones with the lowest memory requirements or the ones with the fastest runtime.

**Quantum Resource Estimates for Factoring.** In 2025, it was estimated that around 900 000 physical qubits corresponding to around 1 600 logical qubits are needed to factor a 2048-bit number [Gid25].

In early 2026, these estimates were "improved": the authors of [Web+26] estimate that around 100 000 physical qubits are needed while the authors of [Cai+26] estimate only 10 000 physical qubits. The number of required logical qubits is less than 1 500—the exact number depends on various tradeoffs. Notably, these latest results ([Web+26; Cai+26]) consider different assumptions than [Gid25]: the authors of [Gid25] use surface codes to realize logical qubits, whereas the authors of [Web+26; Cai+26] use more efficient qLDPC codes.

The authors of [Gid25] estimate that 1–10 billion gates are required to factor a 2048-bit number, which they equate to a week of runtime. The authors of [Web+26] do not provide an explicit gate count, but they estimate that factoring a 2048-bit number using 100 000 physical qubits will take a month. The authors of [Cai+26] report that their space-efficient construction (the one that requires only 10 000 physical qubits) has a runtime in the order of 10 000 *days*, i.e., *decades*.

**Quantum Resource Estimates for Computing Discrete Logarithms.** Another important topic concerns the resource estimates for computing discrete logarithms on 256-bit elliptic curves. Notably, based on current estimates, breaking elliptic curve cryptography requires fewer resources than breaking RSA.

In February 2026, the authors of [CFS26] estimated that computing a discrete logarithm on a 256-bit elliptic curve requires 1 192 logical qubits and hundreds of billions of gates.

In April 2026, the authors of [Bab+26] estimated that computing a discrete logarithm on a 256-bit elliptic curve requires 1 200–1 450 logical qubits, which corresponds to fewer than half a million physical qubits. Although the number of required qubits is not an improvement compared to [CFS26], the number of gates was drastically reduced to less than one hundred million. The authors claim that this translates to a runtime of less than *half an hour*.

Notably, the authors of [Bab+26] have not published the details of their results, but merely a zero-knowledge proof[12] that they have the improved algorithm, citing responsible disclosure: "However, the escalating risk that detailed cryptanalytic blueprints could be weaponized by adversarial actors necessitates a shift in disclosure practices. Accordingly, we believe it is now a matter of public responsibility to share refined resource estimates while withholding the precise mechanics of the underlying attacks." Although this is an innovative way to "present" a result, it may set an unfortunate precedent for not publishing novel results in this research area.

**Conclusion**

Every cryptographic algorithm whose security relies on the hardness of the discrete logarithm problem or the integer factorization problem can be broken with Shor's algorithm [Sho94]. Notably, this concerns effectively all public-key encryption algorithms, such as RSA or ElGamal, digital signature algorithms, such as RSA or ECDSA, and the Diffie-Hellman key exchange.
Therefore, a new category of public-key cryptographic algorithms that resist attacks using Shor's algorithm needs to be developed—these algorithms are called *post-quantum algorithms* (Section 1.5).

### 1.4.2 Impact of Grover's Algorithm

Shor's algorithm (Section 1.4.1) is not the only quantum algorithm which impacts cryptography. There is also Grover's algorithm [Gro96], which is an algorithm for *unstructured search*. The unstructured search problem is defined as follows: Let $f : \{0,1\}^n \to \{0,1\}$ and assume there exists only one $x^* \in \{0,1\}^n$ s.t. $f(x^*) = 1$. Given oracle access to $f$, find $x^*$. To solve this problem using classical computers, the strategy is to try all possibilities for $x^*$, which *on average* requires $2^{n-1}$ queries to $f$. Grover's algorithm can do it in around $\sqrt{2^n}$ queries.

Grover's algorithm is not relevant for public-key cryptography, which is already insecure due to Shor's algorithm anyway, but it impacts symmetric cryptography. For simplicity, we use the term "symmetric cryptography" to refer to symmetric ciphers as well as hash functions and Message Authentication Codes (MACs). Finding a secret key used in a symmetric cipher is an instance of an unstructured search and so is pre-image finding for hash functions, which also has implications for MACs since they are typically constructed using a hash function.

[12] A zero-knowledge proof is a method that allows one party to prove to another that a statement is true without revealing any additional information beyond the validity of the statement.

**Remark**

In cryptography, the security of primitives is expressed in *bits of security*. Specifically, if a primitive has $n$ bits of security, it means that the best attack requires around $2^n$ *operations*—what an "operation" is depends on the context, sometimes it may be bit operations such as XOR, or AND, or it may refer to the number of invocations of the encryption function.

A symmetric cipher with a key of length 128 bits has at most 128 bits of *classical* security. Using Grover's algorithm, the attack requires "only" $2^{64}$ *quantum* operations, i.e., the cipher has only 64 bits of *quantum* security.

For hash functions the situation is similar. The security of a hash function depends on the length of its output—a secure hash function with an output length of 256 bits has at most 256 bits of *classical* security against *pre-image* attacks and at most 128 bits of *classical* security against *collision* attacks. Grover's algorithm can be used to find a pre-image; therefore, the hash function has 128 bits of *quantum* security. Hash functions also lose a third of collision security in terms of bits, compared to classical collision attacks, due to the BHT algorithm [BHT98], which builds upon Grover's algorithm. For example, a 256-bit hash function has $\approx 85$ bits of *quantum* security against collision attacks.

However, the practical implications of these theoretical results are questionable [Flu17; Zal99; Ber09]. This is mainly because Grover's algorithm is hard to parallelize. Currently, the lowest security parameters recommended by organizations such as BSI [BSI26] or NIST [Bar20] have 128 bits of security. This implies that attacks requiring $2^{64}$ or $2^{80}$ operations are considered feasible in the near future. Such a possibility is a consequence of parallelization because there is no single CPU that can do that many operations in a reasonable amount of time. Modern CPUs operate with a frequency in the order of gigahertz, i.e., they can do around $10^{10} \approx 2^{33}$ operations per second. Since there are around $2^{25}$ seconds in a year, a single CPU core can do $\approx 2^{58}$ operations in a year. However, one can reach $2^{64}$ by merely using 63 more cores in parallel. Reaching $2^{80}$ operations requires around 4 million more cores, which is certainly feasible. The Bitcoin network can be considered the biggest parallel computer on the planet currently. In late 2025 its hash rate hovers around $2^{70}$ hashes per second [Blo25] which corresponds to $2^{95}$ hashes per year. This is over $2^{102}$ *bit* operations per year, if we conservatively assume that a single SHA-256 computation takes at least $2^7$ bit operations—we estimate the real number is around $2^{14}$.

Nonetheless, if we ignore the practical implications, Grover's algorithm effectively halves the bit security of symmetric ciphers and hash functions. However, compared to Shor's algorithm, Grover's algorithm does not give us an exponential speedup—the speedup is only quadratic. This means that we can counter Grover-based attacks by simply doubling the cipher's key size or the hash function's output length to preserve the security levels. Note, however, that Grover-based attacks are generic and *black-box*, meaning that they do not exploit internal structures of ciphers and hash functions. There may be advances in quantum cryptanalysis that exploit specific internal structures of ciphers or hash functions. For example, establishing the quantum security of the sponge construction [Ber+07], which is used in the hash function SHA-3 [NIS15a], requires a separate quantum cryptanalysis [Ala+25b] that considers quantum adversaries capable of making superposition queries to the internal function, rather than only quantum adversaries capable of making superposition queries to the black-box sponge construction.

**Conclusion**

In theory, Grover's algorithm [Gro96] halves the bit security for symmetric primitives, such as symmetric ciphers, hash functions and MACs. Unlike in the case of Shor's algorithm [Sho94], the conservative (in terms of security) mitigation of Grover's algorithm is to double the key size, for symmetric ciphers and MACs, and to double the output length for hash functions.

For example, one must replace AES-128 with AES-256 to retain 128-bit security. SHA-256's security against pre-image attacks is 128 bits in the quantum world, but against collision finding it has only $\approx$ 85 bits of security due to the BHT algorithm [BHT98]. Therefore, the recommendation is to move to SHA-384, SHA-512 or their SHA-3 analogues. SHA-2 [NIS15a] and SHA-3 [NIS15b] are considered equally secure against quantum attacks since the output size is what matters. Compared to SHA-3, SHA-2 suffers from length-extension attacks due to its Merkle-Damgård construction.[a]

However, as discussed, Grover's algorithm may not pose a threat even when quantum computers capable of breaking 2048-bit RSA exist. Nonetheless, BSI [BSI26], NSA [NSA22], or ETSI [ETS17b] advise using 256-bit keys for long-term protection.

[a] An adversary is able to calculate $H(s||m||m')$ given $H(s||m)$ where $s$ is a secret, $m$ is the original message, and $m'$ is a suitably chosen suffix. Note that this does not contradict collision resistance and pre-image resistance, but it implies that SHA-2 does not behave like a random oracle (Section 8.2.3.2).

### 1.4.3 Store Now, Decrypt Later

As discussed at the beginning of this section, to break current cryptographic algorithms using Shor's algorithm, we need a CRQC, i.e., a "large" quantum computer. These do not exist yet, but they are expected to materialize in the coming decades. It may seem that we have plenty of time (decades) to migrate existing systems to quantum-safe cryptography (Section 1.5), but this is not the case. Quantum computers pose a threat even before their existence because of the "store now, decrypt later" (SNDL), also known as "harvest now, decrypt later" (HNDL), attack strategy.

"Store now, decrypt later" refers to a two-stage attack. The adversary is passive, i.e., it can eavesdrop on a secure channel, and its goal is to recover the messages sent over the secure channel. The first stage occurs before the attacker gains access to a CRQC, e.g., now. In the first stage, the adversary records the encrypted data and stores it for later use. The second stage begins when the adversary is able to run Shor's algorithm on a CRQC. The adversary simply takes the captured encrypted communication, and uses the CRQC to decrypt it.

The store now, decrypt later attack is only a threat if the captured communication is considered sensitive at the point in the future when the adversary gets a CRQC. Some data need to be secret only for a few minutes or seconds, such as one-time passwords or codes. Other data, e.g., medical, voting, or governmental data, are considered sensitive for decades. The "threat urgency" is illustrated by Mosca's equation [Mos15], which states: if $x + y > z$, then you are vulnerable—$x$ is "for how many years the data will be sensitive", $y$ is "how many years the migration to quantum-safe cryptography will take" and $z$ is "how many years until CRQCs materialize."

## 1.5 Post-Quantum Cryptography

Recall from Section 1.4 that all classical public-key cryptography is broken due to Shor's algorithm whereas symmetric cryptography is only "weakened." In other words, we can keep using the same symmetric algorithms, except with more secure parameters, but increasing parameters for public-key algorithms is futile.

Therefore, new public-key algorithms are needed which resist Shor's algorithm. Specifically, the security of these algorithms must not rely on the hardness of the integer factorization and discrete logarithm problems. These algorithms are called "post-quantum," "quantum-safe" or "quantum-resistant."

In 2006, the first conference on the topic of post-quantum algorithms, PQCrypto, was organized [PQC06]. This marks the beginning of the deliberate[13] development of post-quantum algorithms. A few years later, the book *Post-Quantum Cryptography* [BBD09] was published which summarized post-quantum cryptography at the time in 2009.

In 2016, the US National Institute of Standards and Technology (NIST) began a standardization process for post-quantum cryptography. NIST's goal was to standardize the following post-quantum cryptographic primitives: a Public-Key Encryption (PKE) scheme, a Key Encapsulation Mechanism (KEM), and a digital signature scheme. The standardization process was open to the public, meaning that anyone could submit their proposal. The process had four rounds in total; each round took 2–3 years during which anyone could provide feedback regarding the algorithms. Initially, there were 69 submissions, but only 15 advanced to the third round [Ala+22]. The reasons for not advancing were typically that the algorithm was broken during the rounds, or that the algorithm's performance was lacking compared to similar algorithms [Ala+19; Moo+20; Ala+22].

Each proposed post-quantum algorithm can be categorized based on its hard assumption (Section 1.3). The majority of the proposals fall into the following categories: *code-based*, *lattice-based*, *multivariate-based*, *isogeny-based* and *hash-based*. We present these categories in more detail in Section 2.4.

At the end of the third round, NIST announced that it has selected four post-quantum algorithms for standardization:

- ML-KEM [NIS24b] (née Kyber [Sch+22b]): a lattice-based post-quantum KEM,
- ML-DSA [NIS24a] (née Dilithium [Lyu+22]): a lattice-based post-quantum signature,
- FN-DSA[14] (née Falcon [Pre+22]): a lattice-based post-quantum signature,
- SLH-DSA [NIS24c] (née SPHINCS+ [Hül+22]): a hash-based post-quantum signature.

However, the standardization continued into the fourth round with four post-quantum KEM candidates: BIKE [Ara+22] (code-based), HQC [Agu+22] (code-based), Classic McEliece [Alb+22] (code-based), and SIKE [Jao+22] (isogeny-based). The motivation for the fourth round was to choose a non-lattice-based KEM alternative to ML-KEM. In 2025, NIST selected HQC for standardization [Ala+25a]. Notably, SIKE was broken at the start of the fourth round [CD23] (BIKE and Classic McEliece are considered safe).

[13] Technically, some post-quantum algorithms, such as McEliece [McE78] and NTRU [HPS98], have been developed before, but they were not consciously designed to resist Shor's algorithm.

[14] NIST has yet to publish an official standard for FN-DSA.

In 2022, NIST started an additional standardization process specifically focused on post-quantum signatures [NIS22]. The motivation behind the additional standardization is to diversify the portfolio of post-quantum signature schemes. ML-DSA and FN-DSA have significantly better performance compared to SLH-DSA (see Section 2.4), but they are both lattice-based, which means that if lattice problems are shown to be easy, then the only alternative is SLH-DSA, which is not practical for certain use cases. In 2024, the first round concluded by selecting 14 candidates into the second round [Ala+24]. The selected signature schemes essentially fall into the above-mentioned categories. NIST began using a new *MPC-in-the-Head* category for some signatures. However, we argue that this category is somewhat misleading since it refers to the "construction framework" rather than the underlying hard problems, which fall into the standard categories, except for PERK [Aar+24a].

The NIST standardization processes serve as the main venue for development of post-quantum cryptography. In Europe, the NIST PQC standards are being adopted, e.g., the German BSI [BSI26] and the French ANSSI [ANS23] recommend them. In addition, while BSI and ANSSI recommend NIST-standardized algorithms, they also recommend FrodoKEM [Fro25] and Classic McEliece [Alb+22], which were part of the NIST competition, but were not selected due to performance reasons—they are planned to be standardized by ISO [Cla23; Fro25]. Some countries have engaged in independent standardization of post-quantum cryptography, e.g., China [Ins25] and South Korea [Kpq25].

Apart from the above-mentioned algorithms, two hash-based signature algorithms, LMS and XMSS, were standardized in 2020 by NIST [Coo+20] following their standardization by IETF [MCF19; Hue+18]. However, these signature schemes are so-called *stateful*, which means that the user needs to keep track of a state after each signing operation, which is not typically needed from a standard signature scheme. This constraint means that these signature schemes cannot serve as a generic replacement for classical signature schemes, although they may be adequate in certain scenarios.

**Conclusion**

In 2025, we already have a wide variety of post-quantum cryptographic algorithms. Several of them have already been standardized, or are going to be standardized in the near future. Notably, we have the following KEMs: ML-KEM [NIS24b], HQC [Agu+22], Classic McEliece [Cla23] and FrodoKEM [Fro25], and the following signature schemes: ML-DSA [NIS24a], FN-DSA [Pre+22] and SLH-DSA [NIS24c].

## 1.6 Post-Quantum Cryptography vs. Quantum Cryptography

Aside from *post-quantum* cryptography, there exists another field of cryptography called *quantum* cryptography. Post-quantum cryptography is about developing *classical* cryptographic algorithms that are resistant to attacks using quantum computers whereas quantum cryptography is about developing *quantum* cryptographic algorithms—these are also resistant to attacks using quantum computers, but they were not intentionally designed to resist Shor's or Grover's algorithms.

Quantum-cryptographic schemes typically exploit the *no-cloning theorem*, which states that unknown quantum states cannot be copied. This allows for building cryptographic primitives

such as *uncloneable encryption* [BL20], which prevents a ciphertext from being decrypted more than once, and *encryption with certified deletion* [BI20], which allows the recipient of a ciphertext to either decrypt it or provably "erase" the encrypted message.

The most prominent representative of quantum cryptography is quantum key distribution (QKD), i.e., quantum key exchange. Here, "QKD" typically refers to the BB84 protocol [BB14]. The main benefit of QKD compared to PQC-based key exchange is that it is information-theoretically secure. Recall from Section 1.3 this means that QKD's security does not depend on the computational capabilities of the adversary—as in the case of the one-time pad.

However, apart from being a quantum algorithm, which itself is a drawback since it means that specialized devices are required, there are many other practical issues with QKD. For example, the BB84 key exchange requires a *classical* authenticated channel to work, which means that 1) the authentication keys need to be pre-shared out of band, or 2) the channel is established using post-quantum cryptography. The first case introduces the problem of key distribution, which is why public-key cryptography was used in the first place; the second case means that the system is no longer information-theoretically secure.

Governmental agencies such as the US NSA [NSA20], the German BSI and the French ANSSI [ANS+24], or the UK's NCSC [NCS25a] have stated that they encourage using PQC instead of quantum cryptography. For an overview of the drawbacks of quantum cryptography and evaluation of practical deployments, see [Aqu+25; PPS04].

**Conclusion**

Quantum cryptography, in its current form, is not a full-fledged replacement for classical public-key cryptography. Even considering QKD, it is unrealistic to consider a mass migration from a classical key exchange to QKD due to its requirement for specialized devices—unlike QKD, PQC can be deployed on existing computers or smartphones.

## 1.7 Migration to Post-Quantum Cryptography

Consider a world where someone comes up with a universal key that unlocks all existing locks, but because the key is so complex, it is hard to manufacture. It will take at least 10 years to build one. The lock manufacturers scramble to design a new lock that cannot be opened by this universal key. After two years, they introduce the new *post-universal* lock, which cannot be opened by the universal key. However, all doors, safes, etc., need to be upgraded to use the post-universal lock. For some things, the upgrade will be straightforward, e.g., the lock has the same dimensions as the previous one, but for some things the upgrade may take some work.

In our world, we are currently in an analogous scenario. We already have *post-quantum* cryptographic primitives, as illustrated in Section 1.5, but now we need to integrate these new primitives into existing quantum-vulnerable cryptographic systems. We call this process the *migration to post-quantum cryptography*. Here are a few examples of systems that need to be migrated to post-quantum cryptography: network security protocols (TLS [Res18], WireGuard [Don17], SSH [LY06]), messaging protocols (Signal [Sig26], Matrix [Mat26]), authentication protocols (FIDO2 [W3C21], SAML [OAS12]), public-key infrastructure (public-key certificates [Boe+08], OCSP [San+13]), cryptographic hardware (HSMs), signature formats (AdES [ETS16a]), or software

signing.

In this thesis, we focus on a few of these critical examples, namely, public-key infrastructure (Chapter 3), TLS (Chapter 4), SAML (Chapter 5), AdES (Chapter 6), and the PAPKE primitive (Chapter 8). The selection of these topics is primarily motivated by the practical demands of our industrial partner LuxTrust S.A., which is a Qualified Trust Service Provider (QTSP). Our focus is on cryptographic systems that are especially relevant to the operation of eIDAS-compliant trust services. Therefore, we address PKI and TLS because they are foundational to secure internet communication, with PKI serving as a fundamental component of many other cryptographic systems. Building on this foundation, we investigate XML security, SAML, and AdES, which have received comparatively little attention in the context of migration to post-quantum cryptography. Although the PAPKE primitive is not an essential component for trust service providers, it represents a complementary research direction that addresses post-quantum migration at the primitive level in the area of password-authenticated cryptography.

We cover the entire migration lifecycle—from threat analysis to solution design and planning—stopping just short of final deployment, although we do provide proof-of-concept implementations for several proposed solutions. We decompose the migration process of the selected cryptographic systems into the following parts:

- **Inventory and Threat Modeling:** We begin by analyzing the classical cryptographic system to identify specific quantum-vulnerable points, e.g., reliance on RSA or Diffie-Hellman. We then evaluate the concrete threats posed by quantum adversaries to the identified vulnerabilities.

- **Design of Post-Quantum Solutions:** Once vulnerable components are identified, we propose post-quantum solutions to address the vulnerabilities. Generally, we focus on designing two distinct variants of post-quantum solutions: purely post-quantum solutions and hybrid post-quantum solutions. We cover both variants because they offer different advantages: hybrid solutions are particularly useful during the transitional phase of migration, whereas purely post-quantum solutions represent the long-term deployment target.

- **Prototyping and Performance Evaluation:** We validate our designs by developing proof-of-concept implementations using established cryptographic libraries. We benchmark these implementations to evaluate their practicality.

- **Migration Planning:** The designed solutions cannot be deployed all at once since they are intertwined. Consequently, a migration plan must be developed that describes the sequence of upgrades while taking into account component dependencies, security threats, and compliance aspects.

In summary, the research questions that this thesis aims to answer are:

- *How and why are widely used cryptographic systems like PKI, TLS, SAML and AdES vulnerable to quantum attacks?* (Chapters 3 to 6)

- *How can we upgrade these cryptographic systems to achieve post-quantum security? What are the possible solutions and what advantages and disadvantages do they bring?* (Chapters 3 to 6)

- *How should an organization plan a migration to post-quantum cryptography for a complex infrastructure with many interdependent quantum-vulnerable components? Which components should be prioritized, how do external factors influence the timeline, and which solutions best satisfy regulatory compliance?* (Chapter 7)
- *In what ways are existing PAPKE constructions inadequate for post-quantum instantiation, and how can we design more efficient, rigorously proven quantum-safe PAPKE constructions?* (Chapter 8)

## 1.8 Structure

This thesis is a monograph composed of nine chapters. The chapters are as follows.

Chapter 1 **(Introduction)** begins with a brief introduction to the fields of cryptography and quantum computers. Then, it describes the main motivation behind the topic of this thesis—the impact of quantum computers on cryptography. It concludes by introducing post-quantum cryptography and the topic of migration.

Chapter 2 **(Preliminaries)** formalizes cryptography-related terms introduced in Chapter 1 and others which we use in subsequent chapters. The chapter concludes with a detailed examination of post-quantum cryptographic primitives from a theoretical and practical (performance) standpoint. We also touch upon the topic of *hybridization* in the context of post-quantum cryptography.

Chapter 3 **(Post-Quantum Public-Key Infrastructure)** starts by introducing current *classical* Public-Key Infrastructure (PKI) and its components such as certificates, revocation checking methods (CRL, OCSP), or timestamping protocols. In the second part of the chapter, we describe two possible post-quantum variants of a PKI and its components. Specifically, we describe a *purely* post-quantum variant and a *hybrid* post-quantum variant. Additionally, we provide a proof-of-concept implementation of a post-quantum PKI—the implementation is part of the post-quantum AdES implementation (Chapter 6), which inherently depends on a PQ PKI.

Chapter 4 **(Post-Quantum TLS)** focuses on the Transport Layer Security protocol, which can be considered one of the most important cryptographic protocols currently in use as it secures the majority of internet communication. We begin the chapter by describing the classical TLS protocol and why the protocol is not secure against quantum attacks. The topic of making TLS post-quantum has been explored in multiple works over the past decade. Therefore, the rest of the chapter presents a survey of these various proposals, which mostly follows our publication titled "A Comprehensive Survey on Post-Quantum TLS" [Aln+24].

Chapter 5 **(Post-Quantum XML and SAML)** presents the Security Assertion Markup Language (SAML) framework and its main building block, the Extensible Markup Language (XML). The main use case of SAML is realizing a Single Sign-On (SSO) authentication, which allows users to authenticate at different websites using the same set of credentials. In the first part of the chapter, we describe the SAML framework and its security mechanisms, which are XML signatures and

XML encryption. Furthermore, we analyze its (in)security against attacks using quantum computers. In the second part of the chapter, we propose post-quantum solutions for XML signatures and encryption. Each solution has two variants: a purely post-quantum variant and a hybrid post-quantum variant. Using these post-quantum building blocks, we build a post-quantum SAML solution, again in two variants. Finally, we provide a proof-of-concept implementation of the described solutions. This chapter mostly follows our publication titled "Post-Quantum XML and SAML Single Sign-On" [MO24].

Chapter 6 **(Post-Quantum Advanced Electronic Signatures)** focuses on Advanced Electronic Signatures (AdES), which is a digital signature format designed to be compliant with the eIDAS regulation. An AdES signature is an augmented version of a regular digital signature, which is technically a collection of random bytes. Notably, AdES defines a mechanism to prolong the signature's lifetime beyond the expiration dates of relevant certificates. There exist four concrete instantiations of AdES signatures: XAdES, CAdES, PAdES and JAdES. First, we present these formats in detail and identify their quantum-vulnerable parts. In the second part of the chapter, we propose how to make AdES purely and hybrid post-quantum secure. At last, we provide a proof-of-concept implementation of post-quantum AdES.

Chapter 7 **(Post-Quantum Migration Planning)** is dedicated to the planning aspects of migrating to post-quantum cryptography. We begin by presenting high-level roadmaps proposed by regulatory bodies such as the European Union. We then present our methodology for constructing a migration plan. We conclude by presenting a concrete migration plan to post-quantum cryptography for a service that uses components discussed in Chapters 3 to 6.

Chapter 8 **(Post-Quantum Password-Authenticated Public-Key Encryption)** introduces a post-quantum variant of the novel Password-Authenticated Public-Key Encryption (PAPKE) primitive introduced in [Bra+19a]. First, we introduce the PAPKE primitive with its classical constructions such as PAPKE-IC. Then, we present our new post-quantum PAPKE-HIC construction utilizing the novel Half-Ideal Cipher (HIC) primitive, which is more suited for post-quantum instantiation. We prove its security in the UC model. Furthermore, we identify and fix several issues with the original PAPKE-IC proof in [Bra+19a]. These issues lead us to define a new public-key encryption property we call *decryption robustness*. We investigate how this property relates to other more-established properties. Finally, we instantiate and implement PAPKE-IC and PAPKE-HIC with post-quantum KEMs (ML-KEM, Classic McEliece and FrodoKEM), to illustrate PAPKE-HIC's performance advantage. This chapter mostly follows our publication titled "HIC Is All You Need: Practical Post-Quantum Password-Authenticated Public Key Encryption" [Arr+26].

Chapter 9 **(Conclusion and Outlook)** summarizes results provided in this thesis and discusses future research directions for the topic of migration to post-quantum cryptography and post-quantum cryptography in general.

## 1.9 Contributions

The work presented in this thesis was conducted in the context of an industrial partnership with LuxTrust S.A.—I refer to the work conducted for LuxTrust as the "LuxTrust project." My work was equally split between the work on the LuxTrust project and the work on publications.

### 1.9.1 LuxTrust Project

The goal of the LuxTrust project was to prepare a migration guide for LuxTrust's infrastructure to post-quantum cryptography. LuxTrust is a Qualified Trust Service Provider (QTSP) which is a designation for an eIDAS-compliant [Eur14] trust provider. LuxTrust provides trust services such as user authentication, the issuance and validation of qualified electronic signatures and seals, and the issuance and validation of qualified timestamps, among others.

The LuxTrust project had two stages: 1) identification of cryptographic components used by LuxTrust and evaluation of security with respect to the threat of quantum computers—the inventorization part, 2) design of post-quantum solutions for identified vulnerable components. Note that the last stage of the migration—the implementation and deployment of the proposed solution—was not part of this work, although I provided proof-of-concept implementations for some components.

The deliverables for the project are comprised of 12 documents in total. Although these deliverables are not directly publishable as research publications due to their nature, I have transformed some of these deliverables into two publications: "A Comprehensive Survey on Post-Quantum TLS" [Aln+24] and "Post-Quantum XML and SAML Single Sign-On" [MO24].

During the development of my proof-of-concept implementations, I have contributed to the open-source cryptographic library BouncyCastle by implementing composite post-quantum signatures [Oup23], proposed by the IETF [Oun+25a]. During that process, I have also assisted the RFC draft authors with validating test vectors [Oup24a]. My proof-of-concept implementations are also publicly accessible along with my post-quantum forks of popular open-source libraries such as Apache Santuario [Apa25], DSS [Eur25b], BouncyCastle [Leg25b] and OpenSAML [Shi25].

### 1.9.2 Publications

Over the course of my studies, I co-authored four publications, serving as the first author on three of them.

- **Post-Quantum XML and SAML Single Sign-On** [MO24]
  - Conference: Privacy Enhancing Technologies Symposium (PETS) 2024, CORE rank: A.
  - Co-Authors: Johannes Müller (CNRS/LORIA).
  - Contribution: I am first author of this work. I designed all the proposed solutions for post-quantum XML and SAML, which I also implemented and benchmarked.
- **A Comprehensive Survey on Post-Quantum TLS** [Aln+24]

  - Journal: IACR Communications in Cryptology, vol. 1, no. 2, 2024, unranked.[15]
  - Conference: ArcticCrypt 2025, unranked.[16]
  - Co-Authors: Nouri Alnahawi (Darmstadt University of Applied Sciences), Johannes Müller (CNRS/LORIA), Alexander Wiesmaier (Darmstadt University of Applied Sciences).
  - Contribution: I am the first author of this work. I conducted the majority of the literature review. Based on that, I classified the collected publications and summarized them. Finally, I conducted the post-quantum TLS 1.3 experiment and presented its results.

- **HIC Is All You Need: Practical Post-Quantum Password-Authenticated Public Key Encryption** [Arr+26]
  - Conference: Cryptographers' Track at the RSAC Conference 2026 (CT-RSAC), CORE rank: B.
  - Co-Authors: Afonso Arriaga (University of Luxembourg), David Mestel (Maastricht University), Peter Browne Rønne (University of Luxembourg), Marjan Škrobot (University of Luxembourg).
  - Contribution: I am the first author of this work. I authored the majority of the proofs; I identified the issues with the original PAPKE-IC proof and investigated the related decryption robustness property. I implemented and benchmarked the post-quantum versions of PAPKE-IC and PAPKE-HIC.

- **Lattice-based Multisignature Optimization for RAM Constrained Devices** [Ric+24]
  - Conference: 4th International Workshop on Security and Privacy in Intelligent Infrastructures (SP2I) (workshop at the 19th International Conference on Availability, Reliability and Security, ARES 2024), (ARES CORE rank: B, SP2I is unranked).
  - Co-Authors: Sara Ricci (Brno University of Technology), Vladyslav Shapoval (Brno University of Technology), Petr Dzurenda (Brno University of Technology), Peter Browne Rønne (University of Luxembourg), Lukas Malina (Brno University of Technology).
  - Contribution: I provided a minor contribution to this work: I conducted a literature review, wrote the related work section and provided security feedback to the proposed optimizations of the DS2 scheme.

The material in Chapter 4, Chapter 5, and Chapter 8 is partly based on the results from [Aln+24], [MO24], and [Arr+26], respectively. Due to the nature of my contribution to [Ric+24], I do not discuss the contents of the paper in this thesis.

[15] The IACR CiC journal was established in 2024. Therefore, at the time of writing, it does not have a ranking.

[16] The ArcticCrypt 2025 conference allowed submissions already published in the CiC journal. It had a separate review process (although CiC reviews were taken into account as well).

# Chapter 2

# Preliminaries


2.1 Notation . . . 21
2.2 Symmetric Cryptography . . . 23
2.3 Asymmetric Cryptography . . . 27
2.4 Post-Quantum Cryptography in Detail . . . 32
2.4.1 NIST Security Levels . . . 32
2.4.2 Categories of Post-Quantum Algorithms . . . 32
2.4.3 Performance of Post-Quantum Primitives . . . 36
2.4.4 Hybrid Post-Quantum Cryptography . . . 40
2.4.5 Example Hybrid Constructions . . . 43
2.5 Threat Model . . . 48


This chapter aims to formalize cryptographic notions introduced in Chapter 1 and beyond. We presume the reader is familiar with the basics of linear and abstract algebra, probability theory, and complexity theory. We begin by defining the notation used throughout the thesis in Section 2.1 and continue with the basics of symmetric and asymmetric cryptography in Sections 2.2 and 2.3. In Section 2.4, we dive into post-quantum cryptography and its related constructions. We also provide a brief performance overview of some post-quantum primitives.

## 2.1 Notation

Let $n \in \mathbb{N}$; $\{0,1\}^n$ denotes the set of bitstrings of length $n$, whereas $\{0,1\}^*$ denotes the set of bitstrings of arbitrary length.

Let $\mathcal{X}$ be a finite set; $x \leftarrow\!\$\ \mathcal{X}$ denotes the process of sampling an element from $\mathcal{X}$ with uniform probability and assigning it to $x$. From a formal standpoint, we treat $x$ as a random variable over $\mathcal{X}$ such that $\forall x' \in \mathcal{X} : \Pr[x = x'] = \frac{1}{|\mathcal{X}|}$.

Let $D$ be a probability distribution on $\mathcal{X}$; $x \leftarrow D$ denotes the process of sampling an element from $\mathcal{X}$ according to $D$ and assigning it to $x$, i.e., $\forall x' \in \mathcal{X} : \Pr[x = x'] = D(x')$. $\emptyset$ denotes the empty set.

Let $X$ be a random variable over $\mathcal{X}$ and $n \in \mathbb{N}$; we say that $X$ has (at least) $n$ bits of *min-entropy* if $\forall x \in \mathcal{X} : \Pr[X = x] \leq \frac{1}{2^n}$.

Let $\mathcal{A}$ be a probabilistic algorithm; $y \leftarrow \mathcal{A}(x)$ denotes running $\mathcal{A}$ on the input $x$ with uniformly random coins and getting $y$ as the output. In case we want to be explicit about the random coins input into the algorithm, we list them as an extra parameter: $y \leftarrow \mathcal{A}(x, r)$ where $r \leftarrow D$ for some probability distribution $D$. *PPT* stands for "Probabilistic Polynomial Time." For the sake of readability, we typically omit the explicit randomness space from descriptions of domains and codomains of PPT algorithms. For example, when we denote $\mathsf{Enc} : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ as a PPT algorithm, we implicitly assume that Enc takes an additional input (the randomness). Strictly speaking, it would be more precise to write $\mathsf{Enc} : \mathcal{M} \times \mathcal{K} \times \mathcal{R} \to \mathcal{C}$, where $\mathcal{R}$ is the randomness space.

An *adversary* is a probabilistic algorithm. We say that an adversary is *efficient* if it runs in PPT, or equivalently, we call the adversary a *PPT adversary*. The running time of the adversary is parametrized by a *security parameter*, typically denoted as $\lambda$—a PPT adversary runs in polynomial time in $\lambda$, i.e., $\mathcal{A} \in \mathcal{O}(\lambda^k)$ for some $k \in \mathbb{N}$.

When we say that there is a *reduction*, we mean a probabilistic polynomial time reduction, unless stated otherwise.

**Cryptographic Security Definitions.** Cryptographic security definitions work with the notion of (security) experiments. The security experiment is a "game" played by an adversary and a *challenger* with a specific winning condition. Formally, the challenger is another probabilistic algorithm that sets up the environment for the adversary and runs it. The adversary's goal is to win the game. The design of the experiment reflects the security properties of the cryptographic scheme in question.

Let $\Pi$ be a cryptographic scheme and let EXP be a security experiment involving $\Pi$, played by an adversary $\mathcal{A}$ parametrized by $\lambda$; $\mathsf{Exp}^{\mathsf{EXP}}_{\mathcal{A},\Pi}(\lambda) = 1$ denotes that $\mathcal{A}$ won the experiment, whereas $\mathsf{Exp}^{\mathsf{EXP}}_{\mathcal{A},\Pi}(\lambda) = 0$ denotes failure.

For experiments where the adversary's winning condition is to guess the challenger's bit, we define $\mathcal{A}$'s *advantage* as follows:

$$\mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\Pi}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{EXP}}_{\mathcal{A},\Pi}(\lambda) = 1\right] - \frac{1}{2}\right|.$$

For other, non-binary, experiments, we define the advantage as:

$$\mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathsf{Exp}^{\mathsf{EXP}}_{\mathcal{A},\Pi}(\lambda) = 1\right].$$

We say that "$\Pi$ is Exp-secure" if $\mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\Pi}(\lambda)$ is negligible in $\lambda$ for any efficient adversary $\mathcal{A}$.

**Cryptographic Game Notation.** In the description of cryptographic games, such as Figure 8.4, lines in gray boxes denote modifications based on the "game version." Specifically, when a line is annotated with $\mathsf{G}_i/\mathsf{G}_{\geq i}$, it denotes that the line is present in the game for $\mathsf{Game}_i/(\mathsf{Game}_i, \mathsf{Game}_{i+1}, \dots)$.

Colored text denotes the "lookup" indices in the pseudocode instruction, e.g., $(a, b, c) \in \mathsf{List}$ means "search for an entry in List where the third entry is equal to $c$ and load the other entries into variables $a$ and $b$."

## 2.2 Symmetric Cryptography

This section presents relevant security definitions related to symmetric cryptography. Throughout the work, the term "symmetric cryptography" is used broadly to encompass not only symmetric ciphers but also message authentication codes and hash functions.

Let us first define a *symmetric encryption scheme*, also known as a *symmetric cipher*, *private key encryption scheme*, or a *data encryption mechanism* (DEM).

**Definition 1** (Symmetric Encryption Scheme). Let $\mathcal{K}, \mathcal{M}, \mathcal{C}$ be finite sets. A *symmetric encryption scheme* is a triple of PPT algorithms (KGen, Enc, Dec) s.t.

- $\mathsf{KGen} : \mathbb{N} \to \mathcal{K}$ is the *key generation algorithm*. It takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs a (symmetric) *key* $k \in \mathcal{K}$.
- $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ is the *encryption algorithm*. It takes a key $k \in \mathcal{K}$ and a *plaintext* $m \in \mathcal{M}$ as inputs and outputs a *ciphertext* $c \in \mathcal{C}$.
- $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ is the *decryption algorithm*. It takes a key $k \in \mathcal{K}$ and a *ciphertext* $c \in \mathcal{C}$ as inputs and outputs a *plaintext* $m \in \mathcal{M}$, or a *decryption error* denoted as $\perp$.

We require that $\forall \lambda \in \mathbb{N}$, every key $k \in \mathcal{K}$ output by $\mathsf{KGen}(\lambda)$, and every plaintext $m \in \mathcal{M}$, it holds that $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$. In other words, every *honestly*[1] encrypted ciphertext decrypts correctly—this property is called (perfect) *correctness*.

$\mathcal{K}$ is called the *key space*, $\mathcal{M}$ is called the *plaintext space* or the *message space*, and $\mathcal{C}$ is called the *ciphertext space*.

There are two standard definitions of security for a symmetric encryption scheme: *indistinguishability under chosen-plaintext attack* (IND-CPA) and *indistinguishability under chosen-ciphertext attack* (IND-CCA). We define their corresponding security experiments in Figure 2.1.

**Definition 2** (IND-CPA and IND-CCA security of a symmetric encryption scheme). Let SE be a symmetric encryption scheme.

- We say that SE is IND-CPA-secure, or simply IND-CPA, if for all PPT adversaries $\mathcal{A}$ it holds that $\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{SE}}(\lambda) \leq \mathsf{negl}(\lambda)$.
- We say that SE is IND-CCA-secure, or simply IND-CCA, if for all PPT adversaries $\mathcal{A}$ it holds that $\mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\mathsf{SE}}(\lambda) \leq \mathsf{negl}(\lambda)$.

[1] An *honest* ciphertext is a ciphertext produced by Enc. However, not every $c \in \mathcal{C}$ has to be produced in this manner—that is why there is a distinction.

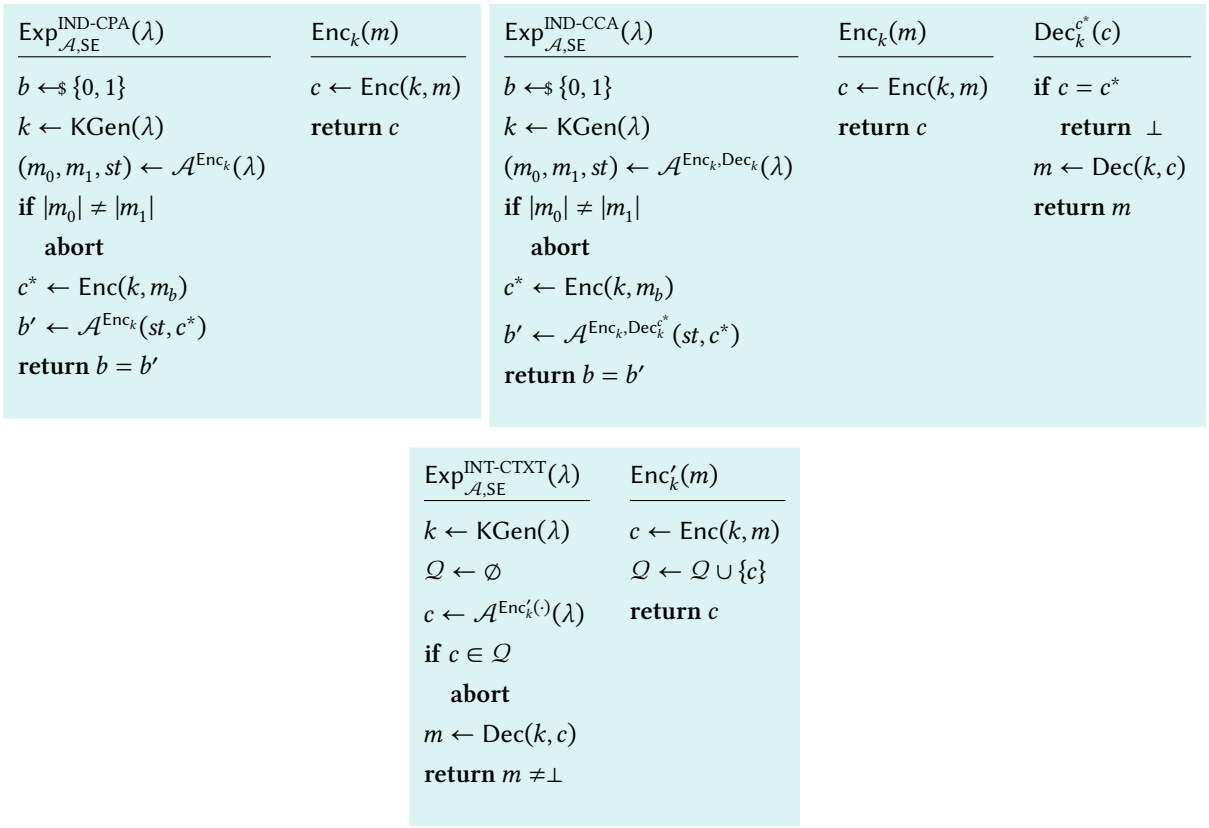



**Figure 2.1:** IND-CPA, IND-CCA, and INT-CTXT security experiments for a symmetric encryption scheme $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$.

**Remark**

In the subsequent definitions of experiments, we omit the definition of oracles when it can be inferred from their names.
Unless stated otherwise, by IND-CCA security we always mean the IND-CCA2 definition where the adversary has access to the decryption oracle throughout the experiment.
For multiple-phase adversaries, i.e., those invoked multiple times throughout the experiment, we omit the state variable *st* and assume it is implicitly passed along.
Additionally, we do not explicitly write out what it means for a scheme to be secure, e.g., that a scheme SE is IND-CPA-secure if, for all PPT adversaries, their advantage is negligible in $\lambda$, as this has already been defined generally at the beginning of Section 2.1.

IND-CPA and IND-CCA definitions assure *confidentiality* of messages. We also want to capture the notion of *authenticity*, i.e., that nobody except the keyholders can produce a valid ciphertext, where "valid ciphertext" means a ciphertext that does not decrypt to $\perp$. This is captured by the notion of *integrity of ciphertexts* (INT-CTXT), defined in Figure 2.1.

**Definition 3** (Authenticated Encryption Scheme). Let SE be a symmetric encryption scheme. We say that SE is an *authenticated encryption scheme* (AE) if it is IND-CCA-secure and INT-CTXT-secure.

Note that some AE schemes have an additional functionality that allows the user to only authenticate (and not encrypt) certain data. Such data are called *additional data* and the AE scheme with this functionality is called *Authenticated Encryption With Additional Data* (AEAD). In our work, we do not distinguish between AE and AEAD unless explicitly stated.

Let us define another symmetric primitive, the message authentication code (MAC).

**Definition 4** (Message Authentication Code). Let $\mathcal{K}, \mathcal{M}, \mathcal{T}$ be finite sets. A *message authentication scheme* is a triple of PPT algorithms (KGen, Sign, Vf) s.t.

- KGen $: \mathbb{N} \to \mathcal{K}$ is the *key generation algorithm*. It takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs a *key* $k \in \mathcal{K}$.
- Sign $: \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ is the *signing algorithm*. It takes a key $k \in \mathcal{K}$ and a *message* $m \in \mathcal{M}$ as inputs and outputs a *tag* $t \in \mathcal{T}$.
- Vf $: \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \{0,1\}$ is the *verification algorithm*. It takes a key $k \in \mathcal{K}$, a tag $t \in \mathcal{T}$, and a message $m \in \mathcal{M}$ as inputs and outputs a bit $b$ indicating whether the tag $t$ is valid for $m$: $b = 1$ means valid, $b = 0$ means invalid.

There are two security notions for MACs: *existential unforgeability under chosen-message attack* (EUF-CMA) and *strong existential unforgeability under chosen-message attack* (SUF-CMA). The corresponding experiments are defined in Figure 2.2.

Next, we introduce one of the most important cryptographic building blocks: a (cryptographic) hash function.

**Definition 5** (Hash Function). Let $l : \mathbb{N} \to \mathbb{N}$ be function. A *(keyed) hash function* is a deterministic polynomial time algorithm $H$ which takes as input a *salt* $s \in \{0,1\}^\lambda$ and an arbitrary-length bitstring $x \in \{0,1\}^*$, and outputs a fixed-length bitstring $H^s(x) \in \{0,1\}^{l(\lambda)}$.

$\underline{\mathsf{Exp}^{\text{EUF-CMA}}_{\mathcal{A},\text{MAC}}(\lambda)}$

$\mathcal{Q} \leftarrow \emptyset$
$k \leftarrow \mathsf{KGen}(\lambda)$
$(m, t) \leftarrow \mathcal{A}^{\mathsf{Sign}'_k}(\lambda)$
**if** $m \in \mathcal{Q}$
  **abort**
**return** $\mathsf{Vf}(k, t, m) = 1$

$\underline{\mathsf{Sign}'_k(m)}$

$t \leftarrow \mathsf{Sign}(k, m)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
**return** $t$

$\underline{\mathsf{Exp}^{\text{SUF-CMA}}_{\mathcal{A},\text{MAC}}(\lambda)}$

$\mathcal{Q} \leftarrow \emptyset$
$k \leftarrow \mathsf{KGen}(\lambda)$
$(m, t) \leftarrow \mathcal{A}^{\mathsf{Sign}'_k}(\lambda)$
**if** $(m, t) \in \mathcal{Q}$
  **abort**
**return** $\mathsf{Vf}(k, t, m) = 1$

$\underline{\mathsf{Sign}'_k(m)}$

$t \leftarrow \mathsf{Sign}(k, m)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, t)\}$
**return** $t$

**Figure 2.2:** EUF-CMA and SUF-CMA security experiments for a MAC MAC.

$\underline{\mathsf{Exp}^{\text{CR}}_{\mathcal{A},H}(\lambda)}$

$s \leftarrow\!\$\ \{0,1\}^\lambda$
$(x, x') \leftarrow \mathcal{A}(s)$
**if** $x = x'$
  **abort**
**return** $H^s(x) = H^s(x')$

$\underline{\mathsf{Exp}^{\text{PR}}_{\mathcal{A},H}(\lambda)}$

$s \leftarrow\!\$\ \{0,1\}^\lambda$
$x \leftarrow\!\$\ \{0,1\}^*$
$y \leftarrow H^s(x)$
$x' \leftarrow \mathcal{A}(s, y)$
**return** $H^s(x) = H^s(x')$

$\underline{\mathsf{Exp}^{\text{SPR}}_{\mathcal{A},H}(\lambda)}$

$s \leftarrow\!\$\ \{0,1\}^\lambda$
$x \leftarrow\!\$\ \{0,1\}^*$
$x' \leftarrow \mathcal{A}(s, x)$
**if** $x = x'$
  **abort**
**return** $H^s(x') = H^s(x)$

$\underline{\mathsf{Exp}^{\text{PRF}}_{\mathcal{A},f}(\lambda)}$

$b \leftarrow\!\$\ \{0,1\}$
$k \leftarrow\!\$\ \{0,1\}^\lambda$
$f' \leftarrow\!\$\ \mathcal{F}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_k}(\lambda)$
**return** $b = b'$

$\underline{\mathcal{O}_k(x)}$

**if** $b = 0$
  $y \leftarrow f(k, x)$
**else**
  $y \leftarrow f'(x)$
**return** $y$

**Figure 2.3:** Collision resistance (CR), preimage resistance (PR), and second-preimage resistance (SPR) definitions for a hash function $H$. Pseudorandomness (PRF) definition for a function $f$, $\mathcal{F}$ denotes the set of all functions with the same signature as $f$.

In practice, we use *unkeyed* hash functions, which we then call just *hash functions*, where the salt is fixed. However, defining hash functions as keyed is necessary to model their security. There are three relevant security definitions for hash functions: *collision resistance* (CR), *preimage resistance* (PR), and *second-preimage resistance* (SPR); see Figure 2.3. Note that CR $\implies$ SPR.

Another important primitive is the pseudorandom function (PRF).

**Definition 6** (Pseudorandom Function). Let $f : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^*$ be a function. We say that $f$ is a *pseudorandom function* (PRF) if it is PRF-secure (Figure 2.3), i.e., if no efficient adversary can distinguish between interaction with $f(k, \cdot)$, where $k \leftarrow\!\$\ \{0,1\}^\lambda$, and a random function $f'(\cdot)$, where $f' \leftarrow\!\$\ \mathcal{F}$ and $\mathcal{F}$ denotes the set of all functions with the same domain and codomain as $f(k, \cdot)$.

Next, we define a key derivation function. Note that we do not define the security of a KDF formally here as it is fairly complex—for a formal treatment see [Kra10].

**Definition 7** (Key Derivation Function). A *key derivation function* (KDF) is a function that takes four inputs: input key material (IKM) $k \in \{0,1\}^*$, length $l \in \mathbb{N}$, salt $s \in \{0,1\}^*$, and context $c \in \{0,1\}^*$, and outputs a bitstring of length $l$.

The main property of a KDF is that it acts as a PRF, i.e., an adversary cannot predict its output. Specifically, the adversary is given information about the distribution of $k$ (but not the value of $k$) and the randomly chosen salt $s$. The adversary then interacts with the "KDF oracle" choosing $l$ and $c$ and is given a challenge output that is real (output of the KDF) or a random bitstring. The goal of the adversary is to decide if the output came from the KDF or not.

In other words, a KDF can be used to derive a pseudorandom bitstring using a secret $k$, which is from a high min-entropy distribution, but not necessarily a uniform one. For example, a finite-field Diffie-Hellman shared secret is a pseudorandom element of some prime-order subgroup of $\mathbb{Z}_p^*$, but not of $\{0,1\}^n$ for some $n$, since $p$ typically is a prime.

Another hash-related primitive is the eXtendable-Output Function (XOF).

**Definition 8** (eXtendable-Output Function). An *extendable-output function* (XOF) is a function with two inputs: an arbitrary-length bitstring $x \in \{0,1\}^*$ and an integer $n \in \mathbb{N}$, and outputs a bitstring of length $n$ (the output length is set by the parameter $n$).

An XOF is effectively a hash function (Definition 5) with a variable-length output. A secure XOF is one that is 1) pre-image resistant and 2) collision resistant. We do not formally define these properties as they are analogous to hash function properties defined in Figure 2.3.

## 2.3 Asymmetric Cryptography

This section presents the relevant definitions in *asymmetric cryptography*, also known as *public-key cryptography*.

Let us first define a public-key encryption scheme (PKE) and establish its security definitions.

**Definition 9** (Public-Key Encryption Scheme). Let $\mathcal{SK}, \mathcal{PK}, \mathcal{M}, \mathcal{C}$ be finite sets. A *public-key encryption scheme* is a triple of PPT algorithms (KGen, Enc, Dec) s.t.

- $\mathsf{KGen} : \mathbb{N} \to \mathcal{SK} \times \mathcal{PK}$ is the *key generation algorithm*. It takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs a *keypair* $(sk, pk) \in \mathcal{SK} \times \mathcal{PK}$; $sk$ is called the *secret key*, or the *private key*, and $pk$ is called the *public key*.
- $\mathsf{Enc} : \mathcal{PK} \times \mathcal{M} \to \mathcal{C}$ is the *encryption algorithm*. It takes a public key $pk \in \mathcal{PK}$ and a *plaintext* $m \in \mathcal{M}$ as inputs and outputs a *ciphertext* $c \in \mathcal{C}$.
- $\mathsf{Dec} : \mathcal{SK} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ is the *decryption algorithm*. It takes a private key $sk \in \mathcal{SK}$ and a *ciphertext* $c \in \mathcal{C}$ as inputs and outputs a *plaintext* $m \in \mathcal{M}$, or a *decryption error* denoted as $\perp$.

$\mathcal{SK}$ is called the *private key space*, $\mathcal{PK}$ is called the *public key space*, $\mathcal{M}$ is called the *plaintext space* or the *message space*, and $\mathcal{C}$ is called the *ciphertext space*.

Unlike with a symmetric cipher, we do not require that a public-key encryption scheme is perfectly correct. In practice, some PKE schemes, especially post-quantum ones, have non-perfect correctness. We define correctness for a PKE scheme as follows:

**Definition 10** (Public-Key Encryption Correctness). We say a public-key encryption scheme (KGen, Enc, Dec) is $\delta$-correct if

$$\mathbb{E}_{(sk,pk)\leftarrow \mathsf{KGen}(\lambda)}\left[\max_{m\in\mathcal{M}} \Pr[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) \neq m]\right] \leq \delta(\lambda)$$

where the probability is taken over the internal randomness of Enc.

We work with many different security properties of a PKE. The well-known definitions are the standard indistinguishability definitions IND-CPA and IND-CCA. However, in Chapter 8, we work with more niche properties:

- *Pseudo-uniformity of public keys* (UNI-PK) captures that the key generation algorithm generates public keys uniformly from the set of all possible public keys.
- *Public key anonymity under chosen-ciphertext attack* (ANO-CCA) captures that ciphertexts reveal no information about the public key used to produce them.
- *Strong robustness under chosen-ciphertext attack* (SROB-CCA) captures that an adversary given two honestly generated public keys is unable to produce a ciphertext that decrypts (to a value $\neq\perp$) under both corresponding private keys.

The corresponding security experiments are listed in Figure 2.4. The UNI-PK property is the most novel property. It was defined for KEMs (Definition 11) in [Arr+25; Beg+23], and we adapted it for PKEs. ANO-CCA was introduced in [Bel+01] and SROB-CCA in [ABN10].

A public-key encryption scheme can be constructed using various methods. One of them is the KEM+DEM construction, which combines a key encapsulation mechanism (KEM) and a data encapsulation mechanism (DEM), i.e., a symmetric cipher. A KEM is defined as follows.

**Definition 11** (Key Encapsulation Mechanism). Let $\mathcal{SK}, \mathcal{PK}, \mathcal{K}, \mathcal{C}$ be finite sets. A *key encapsulation mechanism* is a triple of PPT algorithms (KGen, Encaps, Decaps) s.t.

- $\mathsf{KGen} : \mathbb{N} \to \mathcal{SK}\times\mathcal{PK}$ is the *key generation algorithm*. It takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs a *keypair* $(sk, pk) \in \mathcal{SK}\times\mathcal{PK}$; $sk$ is called the *secret key*, or the *private key*, and $pk$ is called the *public key*.
- $\mathsf{Encaps} : \mathcal{PK} \to \mathcal{K}\times\mathcal{C}$ is the *encapsulation algorithm*. It takes a public key $pk \in \mathcal{PK}$ as input and outputs a *key* $k \in \mathcal{K}$ and a *ciphertext* $c \in \mathcal{C}$.
- $\mathsf{Decaps} : \mathcal{SK}\times\mathcal{C} \to \mathcal{K}\cup\{\perp\}$ is the *decapsulation algorithm*. It takes a private key $sk \in \mathcal{SK}$ and a *ciphertext* $c \in \mathcal{C}$ as inputs and outputs a *key* $k \in \mathcal{K}$, or a *decapsulation error* denoted as $\perp$.

$\mathcal{SK}$ is called the *private key space*, $\mathcal{PK}$ is called the *public key space*, $\mathcal{K}$ is called the *symmetric key space*, and $\mathcal{C}$ is called the *ciphertext space*.

We say that a KEM is *implicitly rejecting* if Decaps never outputs $\perp$; otherwise, we say that the KEM is *explicitly rejecting*.

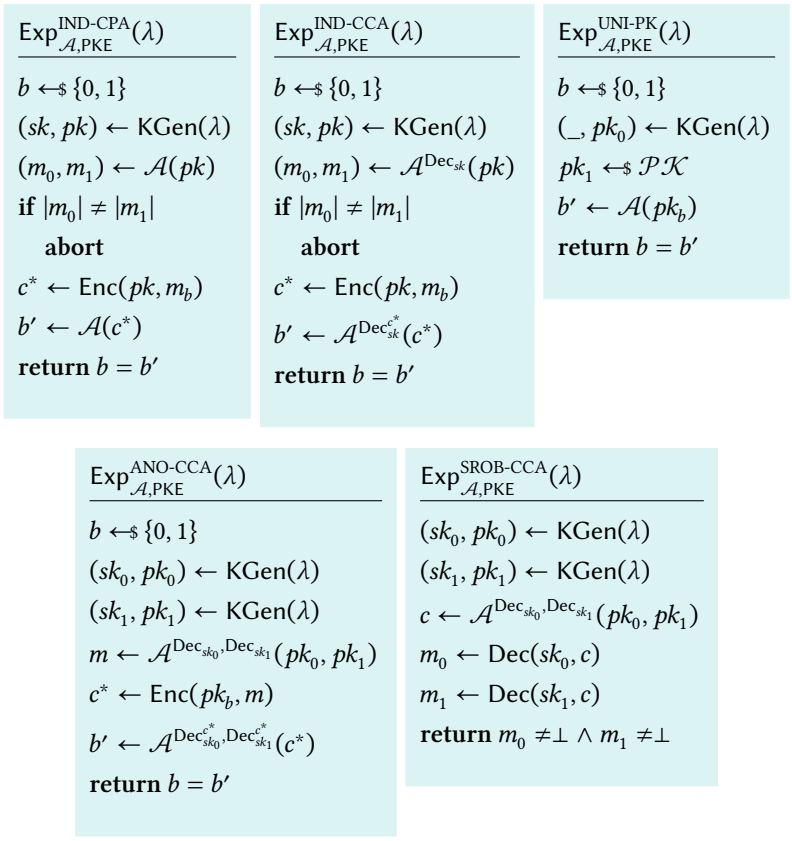



**Figure 2.4:** IND-CPA, IND-CCA, UNI-PK, ANO-CCA, and SROB-CCA security experiments for a public-key encryption scheme $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$.

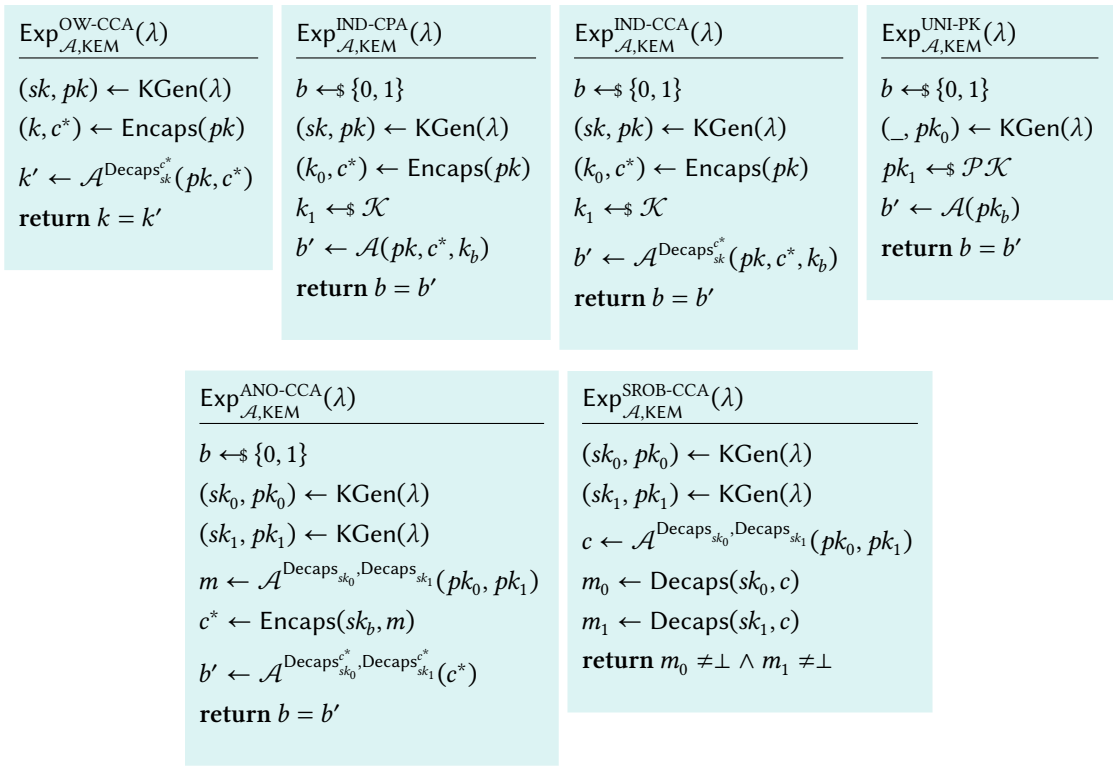



**Figure 2.5:** OW-CCA, IND-CPA, IND-CCA, UNI-PK, ANO-CCA, and SROB-CCA security experiments for a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$.

Apart from the terminology difference (encapsulation/decapsulation vs. encryption/decryption), observe that, compared to a PKE, a KEM does not allow the user to choose "a message" to encapsulate—the message is always a (symmetric) key, i.e., a pseudorandom bitstring. We present security definitions for KEMs in Figure 2.5. The KEM security definitions are analogous to the PKE security definitions given their affinity, however, we present an additional KEM property, *one-wayness under chosen-ciphertext attack* (OW-CCA), which is a weaker version of IND-CCA security.

It is clear that we can construct a KEM from a PKE by modifying the encryption procedure to encrypt a randomly sampled key instead of the message (and analogously the decryption procedure). On the other hand, one can construct a PKE from a KEM and a DEM using the above-mentioned KEM+DEM construction (Definition 12).

**Definition 12** (KEM+DEM Construction). Let $\mathsf{DEM} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a DEM (Definition 1) and $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM (Definition 11). Define the *KEM+DEM* public-key encryption scheme $\mathsf{PKE} = \mathsf{PKE}^{hy}[\mathsf{KEM}, \mathsf{DEM}] = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{PKE.KGen}(\lambda)$: On input $\lambda$, PKE.KGen runs $(sk_{\mathsf{KEM}}, pk_{\mathsf{KEM}}) \leftarrow \mathsf{KEM.KGen}(\lambda)$ and outputs $(sk, pk) = (sk_{\mathsf{KEM}}, pk_{\mathsf{KEM}})$.
- $\mathsf{PKE.Enc}(pk, m)$: On input $(pk, m)$, PKE.Enc runs $(k, c_{\mathsf{KEM}}) \leftarrow \mathsf{KEM.Encaps}(pk)$, then computes $c_{\mathsf{DEM}} \leftarrow \mathsf{DEM.Enc}(k, m)$, and outputs $c = (c_{\mathsf{KEM}}, c_{\mathsf{DEM}})$.

$\underline{\mathsf{Exp}^{\text{EUF-CMA}}_{\mathcal{A},\mathsf{Sig}}(\lambda)}$

$\mathcal{Q} \leftarrow \emptyset$
$(sk, pk) \leftarrow \mathsf{KGen}(\lambda)$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}'_{sk}}(pk)$
**if** $m \in \mathcal{Q}$
  **abort**
**return** $\mathsf{Vf}(pk, \sigma, m) = 1$

$\underline{\mathsf{Sign}'_{sk}(m)}$

$\sigma \leftarrow \mathsf{Sign}(sk, m)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
**return** $\sigma$

$\underline{\mathsf{Exp}^{\text{SUF-CMA}}_{\mathcal{A},\mathsf{Sig}}(\lambda)}$

$\mathcal{Q} \leftarrow \emptyset$
$(sk, pk) \leftarrow \mathsf{KGen}(\lambda)$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}'_{sk}}(pk)$
**if** $(m, \sigma) \in \mathcal{Q}$
  **abort**
**return** $\mathsf{Vf}(pk, \sigma, m) = 1$

$\underline{\mathsf{Sign}'_{sk}(m)}$

$\sigma \leftarrow \mathsf{Sign}(sk, m)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, \sigma)\}$
**return** $\sigma$

**Figure 2.6:** EUF-CMA and SUF-CMA security experiments for a signature scheme Sig.

- $\mathsf{PKE.Dec}(sk, c)$: On input $(sk, c = (c_{\mathsf{KEM}}, c_{\mathsf{DEM}}))$, PKE.Dec runs $k \leftarrow \mathsf{KEM.Decaps}(sk, c_{\mathsf{KEM}})$. If $k = \perp$, then it outputs $\perp$; otherwise, it runs $m \leftarrow \mathsf{DEM.Dec}(k, c_{\mathsf{DEM}})$ and outputs $m$.

**Remark**

An encryption scheme combining a public-key scheme with symmetric-key scheme is called a *hybrid encryption scheme*. Note that this is unrelated to *hybrid post-quantum schemes* (Section 4.3.2).

The KEM+DEM construction preserves IND-CCA security.

**Theorem 1** (Theorem 12.14 [KL20]). *If* KEM *is an* IND-CCA *KEM and* DEM *is an* IND-CCA *DEM, then* $\mathsf{PKE}^{hy}[\mathsf{KEM}, \mathsf{DEM}]$ *is an* IND-CCA *PKE.*

Let us now define (digital) signature schemes.

**Definition 13** (Signature Scheme). Let $\mathcal{SK}, \mathcal{PK}, \mathcal{M}, \Sigma$ be finite sets. A *(digital) signature scheme* is a triple of PPT algorithms (KGen, Sig, Vf) s.t.

- $\mathsf{KGen} : \mathbb{N} \to \mathcal{SK} \times \mathcal{PK}$ is the *key generation algorithm*. It takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs a *keypair* $(sk, pk) \in \mathcal{SK} \times \mathcal{PK}$; $sk$ is called the *secret key*, or the *private key*, and $pk$ is called the *public key*.
- $\mathsf{Sign} : \mathcal{SK} \times \mathcal{M} \to \Sigma$ is the *signing algorithm*. It takes a secret key $sk \in \mathcal{SK}$ and a *message* $m \in \mathcal{M}$ as input and outputs a *signature (value)* $\sigma \in \Sigma$.
- $\mathsf{Vf} : \mathcal{PK} \times \Sigma \times \mathcal{M} \to \{0, 1\}$ is the *verification algorithm*. It takes a public key $pk \in \mathcal{PK}$, a signature $\sigma \in \Sigma$, and a message $m \in \mathcal{M}$ as inputs and outputs a bit $b$ indicating whether the signature $\sigma$ is valid for $m$: $b = 1$ means valid, $b = 0$ means invalid.

$\mathcal{SK}$ is called the *private key space*, $\mathcal{PK}$ is called the *public key space*, $\mathcal{M}$ is called the *message space*, and $\Sigma$ is called the *signature space*.

A signature is a similar primitive to a MAC (Definition 4), except it is a public-key algorithm. Consequently, there are analogous security definitions of unforgeability (Figure 2.6).

## 2.4 Post-Quantum Cryptography in Detail

In this section, we first discuss the basics of post-quantum algorithms and present a selection of the most relevant candidates. Furthermore, we discuss performance characteristics of these post-quantum algorithms. We conclude by introducing the concept of hybrid post-quantum cryptography.

### 2.4.1 NIST Security Levels

As part of the standardization process, NIST defined a metric which is used to evaluate concrete security of proposed algorithms. NIST defined five so-called *security levels* [NIS16]. Each security level corresponds to the difficulty of breaking a specific symmetric primitive. These symmetric primitives were chosen since they are considered secure even against quantum adversaries. Specifically the security levels are defined as follows:

- **Security Level 1**: Any attack 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).
- **Security Level 2**: Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 256-bit hash function (e.g., SHA-256/SHA3-256).
- **Security Level 3**: Any attack 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 192-bit key (e.g., AES-192).
- **Security Level 4**: Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 384-bit hash function (e.g., SHA-384/SHA3-384).
- **Security Level 5**: Any attack 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 256-bit key (e.g., AES-256).

Security level 1 is the weakest security level whereas security level 5 is the most secure. Observe that levels 2 and 4 exist due to the BHT collision finding algorithm [BHT98] whereas levels 1, 3, and 5 consider the best attack to be Grover's search algorithm [Gro96] (see Section 1.4 for details). Another interesting observation is that while in theory AES-128 has only 64 bits of *quantum* security, it is still considered "secure enough" since the security level 1 corresponds to "64 bits of quantum security"—this shows that NIST does not consider Grover's algorithm to be a significant threat in practice due to its difficulty of being parallelized, which was discussed in Section 1.4.

### 2.4.2 Categories of Post-Quantum Algorithms

Recall from Section 1.5 that the majority of post-quantum algorithms can be grouped into five categories based on their underlying hard problem. These categories are: *lattice-based*, *code-based*,

*hash-based*, *multivariate-based*, and *isogeny-based*. Let us briefly present the main underlying problems for each category.

#### 2.4.2.1 Lattice-based Cryptography

The main underlying hard problem in lattice-based cryptography is the *Learning With Errors* (LWE) problem [Reg05] and its related variants, which are the *Ring Learning With Errors* (RLWE) problem and the *Module Learning With Errors* (MLWE) problem. MLWE is a generalization of both LWE and RLWE. Therefore, we present MLWE and discuss how LWE and RLWE are obtained from it. However, let us first introduce a lattice and its related hard problems.

**Definition 14** (MLWE Distribution). Let $n \in \mathbb{N}$, $f(x) \in \mathbb{Z}_q[x]$ be a polynomial of degree $d \in \mathbb{N}$, $s \in (\mathbb{Z}_q[x]/(f(x)))^n$ and let $\mathcal{X}$ be a probability distribution over $(\mathbb{Z}_q[x]/(f(x)))^n$. We define the *MLWE distribution* $\mathcal{A}_{n,d,q,s,\mathcal{X}}$ to be a probability distribution over $(\mathbb{Z}_q[x]/(f(x)))^n \times \mathbb{Z}_q[x]/(f(x))$ s.t. it outputs $(a, b)$ where $a \leftarrow\!\$\ (\mathbb{Z}_q[x]/(f(x)))^n$ and $b = a \cdot s + e$ where $e \leftarrow\!\$\ \mathcal{X}$.

The probability distribution $\mathcal{X}$ is called the *error distribution*.

**Remark**

Elements of the ring $\mathbb{Z}_q[x]/(f(x))$ are polynomials with coefficients in $\mathbb{Z}_q$ with degree at most $d-1$. These polynomials can be viewed as vectors of length $d$ over $\mathbb{Z}_q$. Consequently, $s$ and $a$ from Definition 14 can be viewed as matrices of dimension $d \times n$ over $\mathbb{Z}_q$.
The error distribution $\mathcal{X}$ is typically a distribution over short vectors—vectors with small coefficients.

There are two variants of the MLWE problem: the search variant and the decision variant.

**Definition 15** (Search-MLWE Problem). Let $m \in \mathbb{N}$ and $\mathcal{A}_{n,d,q,s,\mathcal{X}}$ be an MLWE distribution. The *Search-MLWE problem* entails recovering $s \in (\mathbb{Z}_q[x]/(f(x)))^n$ (the parameter of $\mathcal{A}_{n,d,q,s,\mathcal{X}}$) given $m$ samples from $\mathcal{A}_{n,d,q,s,\mathcal{X}}$.

**Definition 16** (Decision-MLWE Problem). Let $m \in \mathbb{N}$ and $\mathcal{A}_{n,d,q,s,\mathcal{X}}$ be an MLWE distribution. The *Decision-MLWE problem* entails distinguishing between $m$ samples from $\mathcal{A}_{n,d,q,s,\mathcal{X}}$ and $m$ uniform samples from $(\mathbb{Z}_q[x]/(f(x)))^n \times \mathbb{Z}_q[x]/(f(x))$.

The hardness of MLWE problems depends on the specific parameters. For example, if $\mathcal{X}$ outputs vectors with coefficients 0 or 1 and $d$ is small, then finding $s$ is feasible by a simple brute-force. It is clear that if one can solve the search problem, then one can solve the decision problem. There is also a reduction in the opposite direction as shown in [Reg05].

The RLWE problems are defined analogously to MLWE problems, except $n = 1$, i.e., $a$ and $s$ are vectors of polynomials of length 1; therefore, $a$ and $s$ are polynomials. The LWE problems are analogous except $d = 1$, i.e., $a$ and $s$ are vectors of polynomials of degree 1; therefore, $a$ and $s$ are vectors with coefficients in $\mathbb{Z}_q$ of length $n$. The name RLWE and MLWE come from the fact that $\mathbb{Z}_q[x]/(f(x))$ is a polynomial *ring* and $(\mathbb{Z}_q[x]/(f(x)))^n$ is a *module* over $\mathbb{Z}_q[x]/(f(x))$ (of rank $n$).

LWE has an important property—it has been shown that solving an average instance of the LWE problem is at least as hard as solving the worst-case instance of the lattice problems GapSVP and SIVP [Reg05]. Note that this property is crucial for building cryptographic systems based on

LWE since we can easily generate instances of the problem and be sure that it is hard—the integer factorization and discrete logarithm problems have the same property. GapSVP and SIVP are considered well-studied problems and the best algorithms have exponential complexity, i.e., they are hard problems. There exist reductions for RLWE [LPR10] and MLWE [LS15] to analogous hard problems; however, note that LWE corresponds to hard problems on *integer* lattices, whereas RLWE corresponds to hard problems on *ideal* lattices and MLWE corresponds to hard problems on *module* lattices. Ideal and module lattices are more "structured" which is considered more risky in terms of security—there may exist an efficient algorithm exploiting such structure which would not work for "plain" LWE.

Let us present the most prominent post-quantum cryptosystems whose security relies on the hardness of lattice-related problems such as LWE. Recall from Section 1.5 that there are three lattice-based algorithms standardized, or planned to be standardized, by NIST. One of them is ML-KEM [NIS24b] (née Kyber [Sch+22b]), which is a KEM whose security relies on the hardness of MLWE (ML stands for "Module Lattice"), and the other two are signatures: ML-DSA [NIS24a] (née Dilithium [Lyu+22]), which relies on hardness of MLWE like ML-KEM, and FN-DSA, which relies on the hardness of the NTRU problem introduced in [HPS98].

Apart from the NIST-standardized algorithms, FrodoKEM [Fro25], which is an LWE-based KEM, is planned to be standardized by ISO [Fro25]. Additionally, a lattice-based signature HAWK [Bos+24] is currently in the second round of the additional NIST standardization project. HAWK has a similar performance profile to FN-DSA with the exception that it does not use floating-point operations, which complicate FN-DSA implementations [Saa23].

Note that LWE-based constructions are less efficient compared to RLWE-based and MLWE-based constructions; however, they are considered to be more conservative options in terms of security because of the above-mentioned structure introduced by a ring, or a module. We discuss their performance in Section 2.4.3.

#### 2.4.2.2 Code-based Cryptography

Code-based cryptography relies on hard problems related to error-correcting codes. The main problem in code-based cryptography is the *syndrome decoding problem* (SDP) which is defined as follows:

**Definition 17** (Syndrome Decoding Problem). Let $n, k, t \in \mathbb{N}$ and let $H \in \mathbb{F}_2^{n-k \times n}$ be a parity-check matrix corresponding to a binary code of length $n$ and dimension $k$ and $c \in \mathbb{F}^{n-k}$. The *syndrome decoding problem* entails finding $e \in \mathbb{F}_2^n$, such that $He = c$, with a Hamming weight $\leq t$.

Note that SDP is NP-hard; however, NP-hardness does not guarantee that the problem is hard *on average* (for a random code). Post-quantum algorithms use certain families of codes for which the average-case hardness of SDP is conjectured, such as Goppa codes used in the original McEliece PKE scheme [McE78].

NIST selected one code-based KEM for standardization: HQC [Agu+25]. Note that HQC relies on the hardness of SDP over Quasi-Cyclic codes. Another prominent code-based KEM is Classic McEliece [Ber+19], which is planned to be standardized by ISO [Cla23]. Classic McEliece's security relies on the hardness of SDP over Goppa codes, which is considered a more conservative assumption compared to SDP over Quasi-Cyclic codes, as Goppa codes have been used in the original McEliece cryptosystem from 1978 [McE78]. Furthermore, there are code-based signature

schemes in the second round of the additional NIST standardization, such as CROSS [Bal+24], SDitH [Agu+24], and RYDE [Ara+24].

#### 2.4.2.3 Hash-based Cryptography

The security of hash-based cryptographic algorithms relies on the security of the specific hash function. In Section 2.2, we introduced security definitions for a hash function: collision resistance, preimage resistance and second-preimage resistance.

SLH-DSA [NIS24c] (née SPHINCS+ [Hül+22]) is a hash-based signature standardized by NIST. Apart from SLH-DSA (SLH stands for "stateless hash-based"), NIST [Coo+20] and IETF have standardized two *stateful* hash-based signatures LMS [MCF19] and XMSS [Hue+18]. These signatures are referred to as stateful because after each signing operation, the private key needs to be updated. This effectively means that they do not conform to the standard signature definition (Definition 13) since Sign outputs two values, the signature value and the updated private key, instead of just the signature value. This hinders practical deployment since the "interface" is not the same as for classical signatures. However, their "stateful nature" makes them more performant compared to the stateless SLH-DSA. Nonetheless, because of that inherent limitation, we do not consider LMS and XMSS as viable for post-quantum migration.

#### 2.4.2.4 Multivariate-based Cryptography

Multivariate cryptography is based on the hardness of solving multivariate systems of equations over a finite field. The main hard problem in multivariate cryptography is the *multivariate quadratic* (MQ) problem.

**Definition 18** (Multivariate Quadratic Problem). Let $n, m, q \in \mathbb{N}$ and define

$$\forall i \in \{1, \dots, m\} \,:\, p^{(i)}(x_1, \dots, x_n) = \sum_{j=1}^{n} \sum_{k=1}^{n} p_{j,k}^{(i)} \cdot x_j \cdot x_k + \sum_{j=1}^{n} p_j^{(i)} \cdot x_j + p_0^{(i)}$$
$$\text{where } \forall i \in \{1, \dots, m\}, \forall j, k \in \{1, \dots, n\} \,:\, p_{j,k}^{(i)}, p_j^{(i)}, p_0^{(i)} \in \mathbb{F}_q.$$

The *multivariate quadratic problem* entails finding $u \in \mathbb{F}_q^n$ s.t. $\forall i \in \{1, \dots, m\} \,:\, p^{(i)}(u) = 0$.

Similar to SDP (Definition 17), the MQ problem is also NP-hard [PG97]. However, as with SDP, this does not tell us much about the security of cryptosystems based on this problem. Multivariate cryptosystems are also not based on this specific version of the problem, but a related one, such as the Oil and Vinegar (OV) problem used in MAYO [Beu+24a].

There have been many multivariate-based signature schemes in the initial NIST standardization process such as Rainbow [Din+20] or LUOV [Beu+19]. However, none of them advanced into subsequent rounds since they got broken [Beu22; Din+21]. There are currently four multivariate-based signatures in the additional NIST standardization process: MAYO [Beu+24a], QR-UOV [Aki+24], SNOVA [Wan+24] and UOV [Beu+24b]. In 2025, a novel attack has been discovered against MAYO and UOV, which undermines the security of proposed parameters [Ran25].[2] For example, the $\text{MAYO}_2$ parameter set lost 28 bits of security due to the attack (141 → 113)—making it no longer security level 1 compliant (Section 2.4.1).

[2] At the time of writing, the paper is not peer-reviewed.

#### 2.4.2.5 Isogeny-based Cryptography

Isogenies are maps between elliptic curves with certain properties. Isogeny-based cryptography relies on hard problems relating to isogenies. The main hard problem in isogeny cryptography is the endomorphism ring problem.

**Definition 19** (Endomorphism Ring Problem). Given $p \in \mathbb{N}$ and a supersingular elliptic curve $E$ over $\mathbb{F}_{p^2}$, the *endomorphism ring problem* entails finding four endomorphisms of $E$ in an efficient representation that generate the endomorphism ring of $E$ as a lattice.

The theory of elliptic curves and isogenies is beyond the scope of this work—just note that an endomorphism is an isogeny from an elliptic curve onto itself. It has been shown that this problem is equivalent to the *l-isogeny finding* problem which is to find an efficient representation of an isogeny of degree $l$ between two supersingular elliptic curves.

During the initial NIST standardization process, there was only one isogeny-based algorithm: a KEM SIKE [Jao+22]. In 2022, SIKE was broken [CD23]. SIKE's security relies on a modified version of the $l$-isogeny finding problem which gave the attacker additional information about the "hidden isogeny." Another isogeny-based key exchange, CSIDH [Cas+18], was proposed outside the NIST standardization process (presumably since it was developed after the start of the standardization process). It was shown that CSIDH's proposed security parameters do not satisfy claimed security [Pei20]. There has been an attempt at proposing a new set of secure parameters for CSIDH, but those parameters make the scheme fairly inefficient [Cam+24; Dar+25].

An isogeny-based signature scheme SQIsign [Aar+24b] is currently in the second round of the additional NIST standardization process.

### 2.4.3 Performance of Post-Quantum Primitives

In this section, we present the performance characteristics of relevant post-quantum algorithms presented in Section 2.4.2. Specifically, we present 1) the sizes of their public keys, ciphertexts (for KEMs), or signatures (for signature schemes), and 2) the speed of operations—key generation, encapsulation, decapsulation, signing, or verification. We do not present sizes of private keys as these values are typically not relevant since private keys are not transferred and their format varies across implementations—some implementations store only a 256-bit seed and others already store (larger) derived values from that seed.

Note that speed of operations is presented in the *(CPU) clock cycle* metric, which indicates the number of CPU clocks used for the computation. The advantage of the clock cycle metric compared to the *time elapsed* metric is that it is independent of the CPU's frequency. Note, however, that it does not translate across different CPU models as CPUs can execute different number of instructions per clock cycle.

#### 2.4.3.1 Performance of Post-Quantum KEMs

This section presents performance metrics of five post-quantum KEMs: lattice-based ML-KEM [NIS24b], NTRU Prime [Ber+20] and FrodoKEM [Fro25]; code-based HQC [Agu+25] and Classic McEliece [Alb+22].

For each KEM we present:

- the size of their public key,
- the size of their ciphertext,
- the speed of key generation,
- the speed of encapsulation,
- and the speed of decapsulation.

As a classical reference point, we include values for the Elliptic Curve Diffie-Hellman (ECDH) key exchange with the NIST curve P-256. Even though ECDH is not a KEM, a DH-based key exchange is set to be replaced with a KEM-based key exchange which involves sending one public key and one ciphertext compared to two public keys in the DH case. Consequently, we consider the "ciphertext size" for ECDH to be its public key size. For encapsulation and decapsulation, we report the shared secret computation, i.e., the exponentiation of a public key by a private key—which is done by both parties during the key exchange.

The KEM performance data, collected from [BL25], is presented in Table 2.1. The size values are in *bytes* and the speed values are in clock *kilocycles*. Since Classic McEliece, FrodoKEM and NTRU Prime have multiple parameters for each security level, we list the chosen parameter sets:

- Classic McEliece parameters: `348864f` (1), `460896f` (3), `6688128f` (5),
- FrodoKEM variant: AES,
- NTRU Prime parameters: `ntrulpr761` (1), `ntrulpr857` (3), `ntrulpr1277` (5).

ML-KEM is the best performing KEM overall—it has the smallest sum of sizes and its operations are even outperforming the classical ECDH. NTRU Prime has a similar performance profile to ML-KEM as it is lattice-based (and not plain LWE). FrodoKEM is the other lattice-based KEM, but its performance is not on the level of ML-KEM, or NTRU Prime, due to being plain LWE-based. On the other hand, FrodoKEM considered a more conservative KEM in terms of security as it is not based on "structured lattices" (MLWE, RLWE, NTRU).

HQC's overall performance is slightly worse compared to its lattice-based counterparts, but it is a code-based KEM, which makes it potentially resistant to advances in lattice cryptanalysis. Classic McEliece, which is a more security-conservative code-based KEM, has the largest public keys of all the KEMs, e.g., its security level 5 public key is over 1 MB whereas ML-KEM's is about 1.5 kB. On the other hand, Classic McEliece has the smallest ciphertexts—only 96 bytes for security level 1—and it has a comparable encapsulation/decapsulation performance to lattice-based KEMs. This makes Classic McEliece the ideal KEM for use-cases where the public key is static.

#### 2.4.3.2 Performance of Post-Quantum Signatures

In this section, we present performance metrics of 10 post-quantum signatures. Three of them were selected by NIST during the initial standardization: lattice-based ML-DSA [NIS24a] and FN-DSA [Pre+22]; hash-based SLH-DSA [NIS24c]. The remaining seven are currently in the second round of the additional standardization by NIST: lattice-based HAWK [Bos+24]; code-based

**Table 2.1:** KEM Performance Metrics

| KEM | Sec. Level | Public Key | Ciphertext | Keygen. | Encaps. | Decaps. |
|---|---|---|---|---|---|---|
| ECDH | 1 | 64 | 64 | 41 | 154 | 154 |
| ML-KEM | 1 | 800 | 768 | 15 | 24 | 18 |
| ML-KEM | 3 | 1 184 | 1 088 | 26 | 37 | 29 |
| ML-KEM | 5 | 1 568 | 1 568 | 35 | 49 | 39 |
| HQC | 1 | 2 249 | 4 433 | 60 | 168 | 290 |
| HQC | 3 | 4 522 | 8 978 | 147 | 385 | 616 |
| HQC | 5 | 7 245 | 14 421 | 294 | 732 | 1 190 |
| Classic McEliece | 1 | 261 120 | 96 | 25 002 | 19 | 94 |
| Classic McEliece | 3 | 524 160 | 156 | 77 813 | 40 | 183 |
| Classic McEliece | 5 | 1 044 992 | 208 | 171 098 | 68 | 215 |
| FrodoKEM | 1 | 9 616 | 9 720 | 889 | 1 115 | 1 062 |
| FrodoKEM | 3 | 15 632 | 15 744 | 1 971 | 2 300 | 1 981 |
| FrodoKEM | 5 | 21 520 | 21 632 | 2 992 | 3 530 | 3 369 |
| NTRU Prime | 1 | 897 | 1 025 | 24 | 45 | 52 |
| NTRU Prime | 3 | 1 184 | 1 312 | 33 | 56 | 67 |
| NTRU Prime | 5 | 1 847 | 1 975 | 44 | 74 | 86 |

Sizes are in bytes and the speed of operations is in clock kilocycles. Data from [BL25], 2023 AMD Ryzen 7 7700; 8 x 3800MHz; hertz, supercop-20251114; median values.

CROSS [Bal+24] and SDitH [Agu+24]; multivariate-based MAYO [Beu+24a], QR-UOV [Aki+24] and SNOVA [Wan+24]; isogeny-based SQIsign [Aar+24b].

For each signature, we present:

- the size of their public key,
- the size of their signature value,
- the speed of key generation,
- the speed of signing,
- and the speed of verification.

The signature scheme performance data, collected from [PQ-25], is presented in Table 2.2. Note that size values are in *bytes* and the speed values are in clock *kilocycles*. Since SLH-DSA, CROSS, SDitH, QR-UOV and SNOVA have multiple parameters for each security level, we list the chosen parameter sets:

- SLH-DSA variant: small, SHAKE, simple,
- CROSS variant: R-SDP, balanced,
- SDitH variant: short,
- QR-UOV parameters: $(31, 600, 70, 10)$ (1), $(31, 890, 100, 10)$ (3), $(31, 1120, 120, 10)$ (5),
- SNOVA parameters: AES, ESK variant, $(24, 5, 16, 4)$ (1), $(24, 5, 16, 5)$ (3), $(29, 6, 16, 5)$ (5).

Note that MAYO has two parameter sets for security level 1; therefore, we chose the $\texttt{MAYO}_1$ parameter set. As a classical reference point, we provide values for the Edwards-curve Digital Signing Algorithm (EdDSA) signature scheme with the curve Curve25519, also known as Ed25519. Note that the Ed25519 values come from a different source than for the post-quantum signature schemes as the source, PQ-SORT [PQ-25], does not provide a classical reference. We have taken the Ed25519 values from [BL25] (Intel Core i7-13700H, P cores; 6 x 4800MHz; raptor, supercop-20231107), which is the closest CPU to the one used by PQ-SORT (Intel Core i7-13700K).

The most size-efficient (the sum of public key and signature value sizes) post-quantum signatures are the isogeny-based SQIsign (213 bytes) and the lattice-based FN-DSA (1563 bytes) and HAWK (1579 bytes). SQIsign is the only post-quantum signature scheme that is on the same level of size efficiency as classical signatures—Ed25519's sum of sizes is 96 bytes, whereas for 2048-bit RSA, the sum of sizes is at least 512 bytes. On the other hand, SQIsign is one of the slowest signature algorithms. The performance profile of FN-DSA and HAWK is similar, however, HAWK is significantly faster compared to FN-DSA, which is most likely due to FN-DSA's use of floating-point operations. Consequently, it makes little sense to prefer FN-DSA if HAWK becomes standardized.

ML-DSA is less size-efficient compared to its other lattice-based counterparts, FN-DSA and HAWK, but it is effectively the most well-rounded signature. ML-DSA's size efficiency is better than the rest of the signature schemes while it is the fastest algorithm if we consider the average speed of all operations.

The hash-based SLH-DSA is essentially the slowest and the most size-demanding signature, however, SLH-DSA's advantage is that it is the most secure option as its security relies only on the security of hash functions, which are also internally used by all other signatures. One thing to note about SLH-DSA is that it actually has the smallest public keys (on par with Ed25519) and its verification speed is fairly quick; therefore, it may be an ideal candidate for a niche use case.

CROSS and SDitH are both code-based signatures. Both have very small public keys (smaller than lattice-based schemes), but their signature sizes are among the largest (CROSS at security level 1 has the biggest signature). It seems that CROSS sacrifices signature size for speed of operations as it is orders of magnitude faster than SDitH, which has 3–5× smaller signatures.

We present results for three multivariate-based signatures: MAYO, QR-UOV and SNOVA. SNOVA's size efficiency is on par with lattice-based schemes; its public keys have a similar size to HAWK's, but its signatures are smaller—almost on par with SQIsign. The same applies for MAYO except its public keys and signatures are about 2× larger compared to SNOVA. The speeds of operations of SNOVA and MAYO are comparable. QR-UOV is the least size efficient signature if we consider the sum of the public key and the signature value, but only its public keys are large—its signature sizes are similar to MAYO's. QR-UOV is also orders of magnitude slower compared to other multivariate schemes; its speed of operations is comparable to SQIsign's.

### 2.4.4 Hybrid Post-Quantum Cryptography

As a consequence of surprising breaks of some post-quantum candidates, notably the breaks of SIKE [Jao+22] and Rainbow [Din+20]. Because these candidates were invented long before the NIST standardization and had survived years of scrutiny, their sudden breaks prompted the cryptographic community to hedge against potential future breaks of, e.g., lattice-based algorithms. The hedging strategy involves combining a post-quantum scheme with a classical scheme. We call this strategy *hybridization*, or that the solution is *hybrid post-quantum* secure. Let us first define formally what classical and post-quantum algorithms are.

**Definition 20** (Classical Cryptographic Algorithm). We say that a public-key cryptographic algorithm is *classical* if its security relies on the hardness of a problem that is efficiently solvable by a quantum computer. Notably, an algorithm is *classical* if its security relies on the hardness of integer factorization (Section 1.3.1) or the discrete logarithm problem (Section 1.3.2).

**Definition 21** (Post-Quantum Cryptographic Algorithm). We say that a public-key cryptographic algorithm is *post-quantum* if its security relies on the hardness of a problem that is not known to be efficiently solvable by a quantum computer. In other words, an algorithm is *post-quantum* if it is not *classical* (Definition 20).

Note that the terms "quantum-safe" and "quantum-secure" are synonyms for "post-quantum." Now let us define a *hybrid post-quantum* algorithm. We also refer to the corresponding hard problems as classical and post-quantum, e.g., integer factorization is a classical (hard) problem whereas LWE is a post-quantum (hard) problem.

**Definition 22** (Hybrid Post-Quantum Cryptographic Algorithm). We say that a public-key cryptographic algorithm is *hybrid post-quantum* if it combines a classical algorithm (Definition 20) and a post-quantum algorithm (Definition 21) in a way that the algorithm remains secure as long as one of the underlying problems is hard.

**Table 2.2:** Signature Performance Metrics

| Signature | Sec. Level | Public Key | Signature Value | Keygen. | Sign. | Verif. |
|---|---|---|---|---|---|---|
| Ed25519 | 1 | 32 | 64 | 39 | 42 | 83 |
| ML-DSA | 2 | 1312 | 2420 | 78 | 193 | 87 |
| ML-DSA | 3 | 1952 | 3309 | 134 | 329 | 140 |
| ML-DSA | 5 | 2592 | 4627 | 213 | 392 | 219 |
| FN-DSA | 1 | 897 | 666 | 19 982 | 667 | 158 |
| FN-DSA | 5 | 1793 | 1462 | 56 814 | 1 324 | 310 |
| SLH-DSA | 1 | 32 | 7856 | 153 431 | 1 166 285 | 1 515 |
| SLH-DSA | 3 | 48 | 16224 | 224 839 | 2 016 929 | 2 131 |
| SLH-DSA | 5 | 64 | 29792 | 146 590 | 1 741 871 | 3 016 |
| HAWK | 1 | 1024 | 555 | 6 263 | 94 | 114 |
| HAWK | 5 | 2440 | 1221 | 29 009 | 195 | 233 |
| CROSS | 1 | 77 | 13152 | 57 | 2 508 | 1 704 |
| CROSS | 3 | 115 | 29853 | 123 | 5 254 | 3 640 |
| CROSS | 5 | 153 | 53527 | 190 | 9 103 | 6 284 |
| SDitH | 1 | 70 | 3705 | 1 405 | 30 287 | 27 214 |
| SDitH | 3 | 98 | 7964 | 4 811 | 185 868 | 176 694 |
| SDitH | 5 | 132 | 14121 | 6 918 | 266 389 | 254 405 |
| MAYO | 1 | 1420 | 454 | 214 | 673 | 278 |
| MAYO | 3 | 2986 | 681 | 492 | 1 324 | 630 |
| MAYO | 5 | 5554 | 964 | 1 242 | 3 320 | 1 414 |
| QR-UOV | 1 | 12266 | 435 | 39 779 | 13 766 | 14 179 |
| QR-UOV | 3 | 34399 | 643 | 143 326 | 41 412 | 41 161 |
| QR-UOV | 5 | 58532 | 807 | 289 804 | 79 559 | 78 123 |
| SNOVA | 1 | 1016 | 248 | 287 | 440 | 254 |
| SNOVA | 3 | 1579 | 379 | 533 | 1 237 | 937 |
| SNOVA | 5 | 2716 | 454 | 938 | 2 000 | 1 562 |
| SQIsign | 1 | 65 | 148 | 43 304 | 101 765 | 5 173 |
| SQIsign | 3 | 97 | 224 | 134 423 | 308 876 | 18 573 |
| SQIsign | 5 | 292 | 421 | 212 255 | 504 885 | 35 809 |

Sizes are in bytes and the speed of operations is in clock kilocycles. Data from [PQ-25]; median values.

We consider a hybrid post-quantum algorithm to always have *two* components: a classical one and a post-quantum one. There can be hybrids composed of more than two components, but this approach has not been implemented in practice since the distrust of post-quantum algorithms is not *that* high. We briefly discuss these hybrids in Chapter 4.

**Remark**

The term "hybrid" in the post-quantum context should not be confused with the older cryptographic term "hybrid encryption" which refers to the method of combining public-key encryption with symmetric-key encryption to produce a public-key encryption mechanism. The KEM+DEM construction (Definition 12) is an example of hybrid encryption.

In other words, a hybrid post-quantum algorithm is secure in the following two scenarios:

1) There is a quantum adversary and the post-quantum algorithm is secure.
2) There is a classical adversary and the post-quantum algorithm is classically broken.

Compared to non-hybrid post-quantum algorithms, hybrids additionally get the 2) guarantee. Hybrids are essentially motivated by the following assumptions:

1) Quantum adversaries will break classical algorithms—Shor's algorithm.
2) Post-quantum algorithms are more likely to be classically broken than classical algorithms—maturity of classical algorithms.

The likelihood of post-quantum algorithms being "quantumly" broken is irrelevant since in that case the solution clearly provides no security—to hedge against that, one needs to use the above-mentioned hybrids with more than two components.

Since a hybrid post-quantum algorithm is a post-quantum algorithm, we need to differentiate between hybrids and non-hybrids. For that purpose, we define *purely post-quantum algorithms*.

**Definition 23** (Purely Post-Quantum Cryptographic Algorithm). We say that a public-key cryptographic algorithm is *purely post-quantum* if it is post-quantum but *not* hybrid post-quantum.

Note that Definitions 20 to 23 do not strictly apply only to *algorithms* but also to protocols, or rather cryptographic solutions in general. The purpose of these terms is to indicate exactly which security properties are preserved in various scenarios.

**Remark**

The IETF has defined terminology related to post-quantum cryptography [DPH25]. We do not adopt their terminology in full, as it is intended for a broader range of use cases and would introduce distinctions that are not needed here. Our definitions of classical (Definition 20) and post-quantum algorithms (Definition 21) are aligned in substance with the IETF terminology; the IETF uses the term "traditional" where we use "classical." Furthermore, our definition of a hybrid post-quantum algorithm (Definition 22) subsumes the IETF notion of a "Post-Quantum/Traditional (PQ/T) hybrid scheme." Depending on the primitive and security goal under consideration, it also captures the associated PQ/T hybrid confidentiality or authentication properties.

#### 2.4.4.1 Advantages and Disadvantages of Hybrid Solutions

From a security perspective, hybrid post-quantum solutions are clearly superior to purely post-quantum solutions as they provide an additional security guarantee. Hybrid post-quantum solutions are expected to be used for a fixed amount of time, we estimate 5–10 years, until the confidence in (purely) post-quantum algorithms is established. Another benefit of hybrid post-quantum solutions is that stakeholders can begin deploying post-quantum cryptography while being compliant with existing regulations requiring classical cryptography, which typically take some time to be updated.

On the other hand, hybrid post-quantum solutions introduce additional complexity since existing solutions (protocols, PKI artifacts, etc.) are generally not designed to accommodate for two different primitives to be used for the same purpose. This complexity affects both the theoretical implementation levels. This is why certain organizations do not recommend hybrid post-quantum solutions but instead recommend using purely post-quantum solutions straight away. For example, the NSA does not recommend using hybrid algorithms [NSA24]; on the other hand, BSI recommends using hybrids [BSI26]. Another disadvantage of hybrids is that at a certain point in the future, there will need to be a migration from hybrid post-quantum solutions to purely post-quantum solutions. That is because hybrid post-quantum algorithms are not expected to be used forever—when quantum adversaries materialize, the classical component becomes redundant.

Hybrid post-quantum solutions are inherently going to be less performant compared to purely post-quantum solutions because of the extra classical component. However, this drawback is generally minor as post-quantum algorithms are the bottleneck since classical algorithms are faster (or around the same speed) and significantly more size-efficient—recall Section 2.4.3.

**Conclusion**

Although hybrid post-quantum solutions have certain drawbacks, we assert that the benefits of hybrid post-quantum solutions outweigh these drawbacks. However, as hybrid solutions are not the "end game," we focus on both purely and hybrid post-quantum solutions in our work.

### 2.4.5 Example Hybrid Constructions

In this section, we introduce a few basic hybrid post-quantum constructions with the aim of giving the reader an idea of how "hybridization" works. We will encounter some of these constructions, such as composite signatures, later on in our work.

#### 2.4.5.1 Hybrid Post-Quantum Signatures

First, we need to define what do we mean by a hybrid post-quantum signature. According to our definitions, we want a signature scheme that is hybrid post-quantum secure. In the context of hybrid post-quantum signatures, we need to take into account the security notion of *non-separability*. This notion was introduced in [Bin+17a]. We do not formally define non-separability in this work, but informally, we say that a signature is *non-separable*, if it cannot be decomposed (separated) into its two components, a classical signature and a post-quantum signature, and these components themselves cannot be used as individual signatures of the signed message. We further

differentiate between so-called *1-non-separability* and *2-non-separability*. These terms specify which component cannot be used as an individual signature, i.e., "separated."

Being able to use these individual components is typically not a desirable property as the hybrid signer inherently produces additional non-hybrid signatures, which may be misused in certain contexts. For example, consider a protocol in which the server signs its responses according to the client's advertised signature capabilities. By stripping the post-quantum component from the client's hybrid signature, an adversary may cause the server to issue only a classically signed response. On the other hand, as we will see in Chapter 5, non-separability allows for backward-compatible hybrid signatures in certain scenarios.

Let us define several hybrid post-quantum signature constructions. We consider two signature schemes, one classical and one post-quantum, $i \in \{1, 2\} : \Sigma_i = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$. We use the dot notation to refer to their specific functions, i.e., $\Sigma_1.\mathsf{Sign}(\cdot)$ refers to the Sign method of the first signature scheme $\Sigma_1$. Note that we also use $\Sigma$ to refer to the set of all possible signature values.

**Parallel Composite Signatures** We begin with the most straightforward hybrid post-quantum signature construction, the *parallel composite signature*. The scheme simply signs the same message "in parallel" with both component schemes. The resulting signature is then a concatenation of the component signatures and the same for its keys. Note that alternative encodings are possible. Since the specific encoding is not relevant, we use the tuple notation $(a, b)$, i.e., $(a, b)$ may refer to concatenation of $a$ with $b$, or some other encoding.

**Definition 24** (Parallel Composite Signature). We define the *parallel composite signature* $\Sigma_{\text{par}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ using components $\Sigma_1$ and $\Sigma_2$ as follows:

- $\mathsf{KGen}(\lambda)$ : Run $(sk_1, pk_1) \leftarrow \Sigma_1.\mathsf{KGen}(\lambda)$ and $(sk_2, pk_2) \leftarrow \Sigma_2.\mathsf{KGen}(\lambda)$. Output $(sk, pk) = ((sk_1, sk_2), (pk_1, pk_2))$.
- $\mathsf{Sign}(sk, m)$ : Decode $(sk_1, sk_2) \leftarrow sk$, run $\sigma_1 \leftarrow \Sigma_1.\mathsf{Sign}(sk_1, m)$ and $\sigma_2 \leftarrow \Sigma_2.\mathsf{Sign}(sk_2, m)$. Output $\sigma = (\sigma_1, \sigma_2)$.
- $\mathsf{Vf}(pk, \sigma, m)$ : Decode $(pk_1, pk_2) \leftarrow pk$, $(\sigma_1, \sigma_2) \leftarrow \sigma$, run $b_1 \leftarrow \Sigma_1.\mathsf{Vf}(pk_1, \sigma_1, m)$ and $b_2 \leftarrow \Sigma_2.\mathsf{Vf}(pk_2, \sigma_2, m)$. Output $b_1 \wedge b_2$.

Note that we use the logical "and" $\wedge$ to indicate that the output is 1 iff both $b_1 = 1$ and $b_2 = 1$. For subsequent definitions, we omit the decoding steps.

Regarding separability, the parallel composite signature is clearly separable as $\sigma_1$ and $\sigma_2$ are valid signatures on their own of the message $m$ under their respective keypairs. We refer to [Bin+17a] for the proof that $\Sigma_{\text{par}}$ achieves hybrid EUF-CMA security, i.e., if one of the component signatures becomes forgeable, $\Sigma_{\text{par}}$ retains unforgeability.

Parallel composite signatures can be made non-separable by introducing domain separation to the message, e.g., by prefixing the message with a component-specific constant. In theory, this solves the problem as we have defined non-separability to not be able to use the component signatures as signatures for that specific message. However, the component signatures are valid signatures for a modified message: the original message with the prefix.

Such an approach is planned to be standardized by the IETF [Oun+25a]—the standard currently supports only ML-DSA-based hybrids, but it can be generalized to other hybrids. The construction

uses a more involved domain separation with pre-hashing. The message signed by the component algorithms has the following format: $m' = \mathrm{c}||\mathrm{l}||\mathrm{len}(ctx)||ctx||\mathsf{H}(m)$ where $c$ is a fixed constant, $l$ is an algorithm-specific constant, $ctx$ is a message-specific domain separation value, $\mathrm{len}(ctx)$ is its length in bytes and $\mathsf{H}(m)$ is a hash of the message to-be-signed. $\mathsf{H}$ is a hash function corresponding to the signing algorithms' security level. Since this is the approach that is to be standardized, we consider this the default approach when referring to composite hybrid signatures in our work. Note that throughout the standardization process, the details of [Oun+25a], such as the specific $m'$ format, have changed, and they may change in the future.

**Remark**

Unless stated otherwise, when we refer to *composite (hybrid) signatures* we mean the *parallel* composite (hybrid) signatures from Definition 24 implemented as specified in the IETF draft [Oun+25a].

Instead of signing in parallel, we can also sign in sequence. We present two approaches: weakly and strongly nested composite signatures. The weak variant does not achieve hybrid security whereas the strong one does.

**Weakly-Nested Composite Signatures** The *weakly-nested composite signature* signs the message with the first component and then the resulting signature value with the second component.

**Definition 25** (Weakly-Nested Composite Signature). We define the *weakly-nested composite signature* $\Sigma_{\text{weak}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ using components $\Sigma_1$ and $\Sigma_2$ as follows:

- $\mathsf{KGen}(\lambda)$ : Run $(sk_1, pk_1) \leftarrow \Sigma_1.\mathsf{KGen}(\lambda)$ and $(sk_2, pk_2) \leftarrow \Sigma_2.\mathsf{KGen}(\lambda)$. Output $(sk, pk) = ((sk_1, sk_2), (pk_1, pk_2))$.
- $\mathsf{Sign}(sk, m)$ : Decode $(sk_1, sk_2) \leftarrow sk$, run $\sigma_1 \leftarrow \Sigma_1.\mathsf{Sign}(sk_1, m)$ and $\sigma_2 \leftarrow \Sigma_2.\mathsf{Sign}(sk_2, \sigma_1)$. Output $\sigma = (\sigma_1, \sigma_2)$.
- $\mathsf{Vf}(pk, \sigma, m)$ : Decode $(pk_1, pk_2) \leftarrow pk$, $(\sigma_1, \sigma_2) \leftarrow \sigma$, run $b_1 \leftarrow \Sigma_1.\mathsf{Vf}(pk_1, \sigma_1, m)$ and $b_2 \leftarrow \Sigma_2.\mathsf{Vf}(pk_2, \sigma_2, \sigma_1)$. Output $b_1 \wedge b_2$.

The weakly-nested composite is clearly 2-non-separable since the second signature value $\sigma_2$ is not a signature of the message but of the first signature value $\sigma_1$. However, the signature scheme is not hybrid secure. It was shown in [Bin+17b] that if the first component signature scheme is EUF-CMA, then the composite scheme is EUF-CMA; however, this does not tell us anything about hybrid "insecurity." We prove the hybrid insecurity in Lemma 1.

**Lemma 1** ($\neg(\Sigma_1$ EUF-CMA or $\Sigma_2$ EUF-CMA $\implies \Sigma_{\text{weak}}$ EUF-CMA)). *There exist signatures $\Sigma_1$ and $\Sigma_2$ s.t. at least one is* EUF-CMA*, but $\Sigma_{weak}$ is not* EUF-CMA.

*Proof.* The authors of [Bin+17b] have shown that if the first component is secure, then the composite scheme is secure. Therefore, we need to consider the case where $\Sigma_1$ is not EUF-CMA, but $\Sigma_2$ is.

Consider the case where $\Sigma_1$ is a degenerate signature scheme where any signature is valid for every message (this scheme obviously satisfies our assumption that it is not EUF-CMA) and

let $\Sigma_2$ be any EUF-CMA scheme. According to the EUF-CMA security definition, we have access to the public key $pk = (pk_1, pk_2)$ and to a signing oracle, which outputs a valid $\Sigma_{\text{weak}}$ signature of our chosen message. Our goal is to produce a valid signature for a message that was not submitted to the oracle. To do that, we simply query the signing oracle for an arbitrary message $m$ and get its signature $\sigma = (\sigma_1, \sigma_2)$. By definition of $\Sigma_{\text{weak}}$ we have $\Sigma_1.\mathsf{Vf}(pk_1, \sigma_1, m) = 1$ and $\Sigma_2.\mathsf{Vf}(pk_2, \sigma_2, \sigma_1) = 1$. Recall that we assume that $\sigma_1$ is a valid signature for any message; therefore, $\sigma = (\sigma_1, \sigma_2)$ is also a valid signature for some other message $m' \neq m$. In other words, the pair $(m', \sigma)$ is a forgery. □

**Strongly-Nested Composite Signatures** As opposed to the weakly-nested composite signature (Section 2.4.5.1), the *strongly-nested composite signature* is hybrid secure. The only difference is that now we sign also the message using the second component.

**Definition 26** (Strongly-Nested Composite Signature). We define the *strongly-nested composite signature* $\Sigma_{\text{str}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ using components $\Sigma_1$ and $\Sigma_2$ as follows:

- $\mathsf{KGen}(\lambda)$ : Run $(sk_1, pk_1) \leftarrow \Sigma_1.\mathsf{KGen}(\lambda)$ and $(sk_2, pk_2) \leftarrow \Sigma_2.\mathsf{KGen}(\lambda)$. Output $(sk, pk) = ((sk_1, sk_2), (pk_1, pk_2))$.
- $\mathsf{Sign}(sk, m)$ : Decode $(sk_1, sk_2) \leftarrow sk$, run $\sigma_1 \leftarrow \Sigma_1.\mathsf{Sign}(sk_1, m)$ and $\sigma_2 \leftarrow \Sigma_2.\mathsf{Sign}(sk_2, (\sigma_1, m))$. Output $\sigma = (\sigma_1, \sigma_2)$.
- $\mathsf{Vf}(pk, \sigma, m)$ : Decode $(pk_1, pk_2) \leftarrow pk$, $(\sigma_1, \sigma_2) \leftarrow \sigma$, run $b_1 \leftarrow \Sigma_1.\mathsf{Vf}(pk_1, \sigma_1, m)$ and $b_2 \leftarrow \Sigma_2.\mathsf{Vf}(pk_2, \sigma_2, (\sigma_1, m))$. Output $b_1 \wedge b_2$.

The proof of hybrid security can be found in [Bin+17a]. Note that similar to weakly-nested composite, the strongly-nested composite is also 2-non-separable.

#### 2.4.5.2 Hybrid Post-Quantum KEM

Similar to hybrid post-quantum signatures (Section 2.4.5.1), we can construct hybrid post-quantum KEMs. The construction is analogous to parallel composite hybrid signatures. If we consider two KEMs, one classical and one post-quantum, and use them in parallel, we obtain two shared secrets. We need to derive a single shared secret from the two component shared secrets. This derivation mechanism is called a *combiner*.

Combiners have been extensively studied [GHP18; Bin+19b]. There exist various combiners but only some of them are hybrid secure. By security, in the KEM context, we mean IND-CCA security (recall from Section 2.3), i.e., the hybrid KEM is IND-CCA as long as one of the component KEMs is IND-CCA. Let us first define the generic hybrid post-quantum composite KEM. Then, we present various combiners. Analogous to hybrid signatures, we consider two (component) KEMs, one classical and one post-quantum, denoted as $\Pi_1$ and $\Pi_2$. We assume the keys output by KEMs have the same length, e.g., 256 bits.

**Definition 27** (Composite KEM). We define the *composite KEM* $\Pi_c = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ using components $\Pi_1, \Pi_2$ and a combiner $\mathsf{C} : \mathcal{C}_1 \times \mathcal{C}_2 \times \mathcal{K}_1 \times \mathcal{K}_2 \to \mathcal{K}$ as follows:

- $\mathsf{KGen}(\lambda)$ : Run $(sk_1, pk_1) \leftarrow \Pi_1.\mathsf{KGen}(\lambda)$ and $(sk_2, pk_2) \leftarrow \Pi_2.\mathsf{KGen}(\lambda)$. Output $(sk, pk) = ((sk_1, sk_2), (pk_1, pk_2))$.

- $\mathsf{Encaps}(pk)$ : Run $(c_1, k_1) \leftarrow \Pi_1.\mathsf{Encaps}(pk_1)$ and $(c_2, k_2) \leftarrow \Pi_2.\mathsf{Encaps}(pk_2)$. Output $(c, k) = ((c_1, c_2), \mathsf{C}(k_1, k_2, c_1, c_2))$.
- $\mathsf{Decaps}(sk, c)$ : Run $k_1 \leftarrow \Pi_1.\mathsf{Decaps}(sk_1, c_1)$ and $k_2 \leftarrow \Pi_2.\mathsf{Decaps}(sk_2, c_2)$. If $k_1 = \perp$ or $k_2 = \perp$, output $\perp$, otherwise output $k = \mathsf{C}(k_1, k_2, c_1, c_2))$.

In Definition 27, we define a combiner as function of the component ciphertexts and the shared secrets since some combiners utilize the ciphertexts in the key derivation. Next, we introduce secure and insecure combiner constructions.

**XOR Combiner** The XOR combiner has been proposed in [GHP18] and shown to be insecure. To be precise, the combiner preserves IND-CPA security, but it does not preserve IND-CCA security, which is considered the standard for KEMs. The combiner XORs the component keys into one, i.e., $\mathsf{C}(k_1, k_2, c_1, c_2) = k_1 \oplus k_2$, ignoring the ciphertexts.

**PRF Combiner** Another naive combiner construction is to use a hash function $\mathsf{H}$ and hash the component keys together, i.e., $\mathsf{C}(k_1, k_2, c_1, c_2) = \mathsf{H}(k_1, k_2)$. This combiner is also not IND-CCA secure. Let us sketch why. In the IND-CCA security game, the adversary is given a challenge ciphertext $c^* = (c_1^*, c_2^*)$ and access to a decapsulation oracle. The adversary cannot query the challenge ciphertext to the oracle, but if one of the component KEMs, e.g., the first one, is insecure in such a way that the adversary is able to craft a ciphertext $c_1' \neq c_1^*$ that decapsulates to the same key, the adversary can query the modified challenge $(c_1', c_2^*)$ to obtain the shared secret.

**Split-key PRF Combiner** A combiner that is hybrid post-quantum secure uses a *split-key pseudorandom function* (split-key PRF). This notion, introduced in [GHP18], is an extension of the standard PRF notion (Definition 6) where we assume the first argument to the PRF—the random key—can be split into two components. Then, the PRF is a split-key PRF if the function remains a PRF as long as one of the component keys is random. This directly corresponds to hybrid security where we want the function to produce random keys even if one of the component keys is compromised.

The combiner is therefore this split-key PRF, which takes as keys the component keys and the second argument is the concatenation of the component ciphertexts. Specifically, let $F : \mathcal{K}_1 \times \mathcal{K}_2 \times \{0,1\}^* \to \mathcal{K}$ be a split-key PRF, then the shared secret is computed as $k = F(k_1, k_2, c_1 \| c_2)$. If we consider $\mathsf{H}$ to be a random oracle, then the function which hashes the concatenation of $k_1, k_2, c_1, c_2$ with $\mathsf{H}$ is a splitPRF as shown in [GHP18, Lemma 6].

This combiner (with additional input for better domain separation) was proposed by IETF at the beginning of its standardization [OWK24]. Since then, the focus shifted from a general hybrid KEMs to an ML-KEM + ECDH specific hybrid [Oun+25b]. The downside of this approach is that post-quantum KEM ciphertexts are large and hashing them hinders performance. It has been shown in [Bar+24] that it is not necessary to hash the ML-KEM ciphertext for the KEM hybrid composed of ML-KEM and X25519. Consequently, the X-Wing combiner derives the shared key from the component keys, the classical ciphertext and the classical public key (as opposed to the post-quantum ciphertext). The X-Wing combiner is currently planned to be standardized by IETF [Oun+25b].

## 2.5 Threat Model

An essential part of migrating to post-quantum cryptography is to identify which cryptographic components or systems are vulnerable to the quantum computer attacks described in Section 1.4. We define a *quantum adversary*, or a *quantum attacker*, as an attacker with access to a cryptographically relevant quantum computer (CRQC), i.e., we assume that the attacker can *break* any non-quantum-safe public-key algorithm in "reasonable time." By "reasonable time", we mean seconds, days, or months, depending on the context.

By "breaking an algorithm," we specifically mean that the attacker is able to recover the private key from the public key, which implies the compromise of standard security guarantees provided by public-key algorithms, such as unforgeability for signature algorithms and confidentiality for public-key encryption and KEMs. In general, we do not consider more formal or fine-grained security analyses, e.g., the impact on EUF-CMA security compared to SUF-CMA security, because of the overwhelming capabilities of quantum adversaries, which render such analysis moot—computing the private key from the public key is the most straightforward approach to breaking any quantum-vulnerable public-key algorithm. Unless stated otherwise, we naturally assume that the attacker has access to the public keys. For simplicity, we model the CRQC as an abstract *quantum-computer oracle* that can be queried by the quantum adversary. For example, on input of a 2048-bit RSA public key, the oracle outputs the corresponding private key. We express the difficulty of attacks in terms of the number of queries to this oracle.

# Chapter 3

# Post-Quantum Public-Key Infrastructure


3.1 **Public-Key Infrastructure Components** . . . . . 50
3.1.1 Motivating Example . . . . . 50
3.1.2 PKI Terminology . . . . . 52
3.1.3 Certificates . . . . . 53
3.1.4 Revocation Checking . . . . . 59
3.1.5 Certificate Management . . . . . 65
3.1.6 Timestamping . . . . . 72
3.2 **Purely Post-Quantum Public-Key Infrastructure** . . . . . 76
3.2.1 Post-Quantum PKI Terminology . . . . . 76
3.2.2 Purely Post-Quantum Certificates . . . . . 78
3.2.3 Transition to a Purely Post-Quantum PKI . . . . . 81
3.2.4 Purely Post-Quantum Certificate Revocation Lists . . . . . 82
3.2.5 Purely Post-Quantum Online Certificate Status Protocol . . . . . 83
3.2.6 Purely Post-Quantum Certificate Management Protocol . . . . . 83
3.2.7 Purely Post-Quantum CMS EnvelopedData . . . . . 85
3.2.8 Purely Post-Quantum Time-Stamp Protocol . . . . . 86
3.3 **Hybrid Post-Quantum Public-Key Infrastructure** . . . . . 86
3.3.1 Parallel Hybrid Post-Quantum Public-Key Infrastructure . . . . . 86
3.3.2 Composite Hybrid Post-Quantum Public-Key Infrastructure . . . . . 92
3.4 **Conclusion** . . . . . 95


In this chapter we explore the topic of Public-Key Infrastructure (PKI). PKI is an essential part of most cryptographic systems. Its purpose is to provide a scalable and practical way to authenticate public keys—be they public keys of PKEs, KEMs, or signature schemes.

We begin this chapter with an overview of PKI components while assessing its security with respect to quantum adversaries in Section 3.1. Subsequently, in Section 3.2, we describe a purely post-quantum PKI—a PKI secure against quantum adversaries. Furthermore, Section 3.3 expands on this by discussing hybrid post-quantum PKI, which serves as a stepping stone between classical and purely post-quantum PKIs during migration.

## 3.1 Public-Key Infrastructure Components

Public-Key Infrastructure (PKI) is an umbrella term for various components involved in managing public-key certificates. The purpose of a PKI is to provide a way for entities communicating over insecure networks to authenticate public keys. These public keys may be used to establish a secure channel over the network and for various other purposes.

We begin with a motivating example that introduces the core PKI concepts in Sections 3.1.1 and 3.1.2. In Section 3.1.3, we describe the structure of (public-key) certificates, how they are created and how they are validated. The validation process is where most of the other PKI components play a role, such as revocation checking, to which we dedicate Section 3.1.4. Specifically, we present two common methods of revocation checking: Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP). Another important PKI-related component is management protocols, i.e., protocols used by PKI entities to communicate PKI-related information. We present one such protocol in detail—the Certificate Management Protocol (CMP)—and discuss its alternatives in Section 3.1.5. In Section 3.1.6, we focus on timestamping, which is the process of binding data to a point in time.

At the end of each section, we analyze the security of the presented PKI components with respect to a quantum attacker described in Section 2.5.

For each protocol and format discussed, we cover the basics—the scope is defined by the purpose of this document, which is to analyze the security against quantum adversaries. Consequently, we focus on the cryptographic parts of the PKI and less on the security engineering parts, which are agnostic to the cryptographic primitives used, e.g., how to build the certificate validation path based on various policies.

### 3.1.1 Motivating Example

Let us motivate the various PKI components using a real-world example: a user securely accessing their bank account over the internet.

Suppose a user wants to access their bank account from their personal device at home using a web browser. When the user types `https://www.mybank.com` into the search bar and hits "Enter," the TLS protocol [Res18] is executed (see Chapter 4 for a detailed treatment of the TLS protocol). For illustrative purposes, our description of TLS is simplified.

The purpose of TLS is to establish a secure channel between the bank and the user even in the presence of active adversaries on the network. The user wants to be certain that they are really communicating with the bank and not some man-in-the-middle attacker. Without going into detail, TLS works by deriving a shared key that is then used to encrypt the traffic between the user and the bank. To derive the shared key, the user only needs to know the bank's (Diffie-Hellman) public key.

The problem is how to securely transport the bank's public key to the user over the insecure channel. One option would be to use another *secure* channel to obtain the key, e.g., going physically to the bank. Clearly, this option has many disadvantages, but the main one is scalability—imagine needing to do that for every website you connect to. PKI provides a solution to this problem.

The bank sends the user a *certificate* containing the bank's Diffie-Hellman public key.[1] A (public-key) certificate binds a public key to an identity. How can the user trust what the certificate says, which is "this public key belongs to the bank"? PKI solves this by establishing a *certification path* or a *certificate chain*. Essentially, a public-key certificate contains these data points: 1) the public key, 2) the identity of the owner, and 3) the identity and signature of the entity that verified that 1) belongs to 2). If the user *trusts* the entity that certified the binding (by applying its signature), then such a certificate is enough for the user to trust that the public key belongs to the claimed owner. The entity that certifies such bindings, i.e., issues certificates, is called a *certification authority* (CA).

Let us elaborate on what *trust* means in this context. The user trusts the information in the certificate only if it was signed (certified) by a CA that the user inherently trusts. A CA that the user inherently trusts is called a *trust anchor* or sometimes a *root CA*. Specifically, we assume that the user has obtained the trust anchor's public key by using other secure channels.

In practice, there are additional links in the certification chain. These links are called *Intermediate* Certification Authorities (ICAs). Intermediate CAs improve PKI scalability. The certification chain from our example (with an intermediate CA) is depicted in Figure 3.1. Note that if an intermediate CA is part of the certification chain, then the bank needs to send the CA's certificate together with its own certificate so that the user can verify them both against their trust anchor.

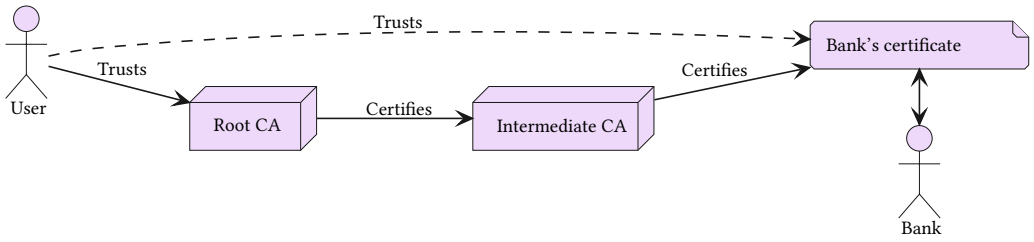



**Figure 3.1:** An example of trust establishment using a PKI.

We described a process by which the user can verify that the public key belongs to the bank without needing to physically go to the bank. Note that we did not explain in detail how the user obtains the trust anchor's public keys, or rather the certificates containing them, that are part of the certification chain. In practice, these so-called *root* certificates are typically included in the operating system or browser. For example, the certificates trusted by Mozilla Firefox are listed at [Moz25].

Observe that a PKI is essentially a centralized version of the initial approach in which is the user going physically to the bank to get their public key. We still need to obtain some keys (in the form of root certificates) using a trusted channel, e.g., by purchasing a phone with the operating system installed. However, that process is streamlined in such a way that it is unnoticeable.

[1] Modern TLS (version 1.3 [Res18]) sends a certificate containing a public key corresponding to a signing key that is used to sign a one-time (ephemeral) Diffie-Hellman public key used to derive the shared key. The use of ephemeral keys provides *forward-secrecy*—leaking the signing (private) key does not compromise past sessions.

### 3.1.2 PKI Terminology

In the motivating example (Section 3.1.1), we introduced many PKI-related terms. We aggregate them here. Note that the majority of the definitions are taken from [ETS16a].

**Definition 28** (Public-Key Certificate). A *(public-key) certificate* contains a public key of an entity together with some other information rendered unforgeable by a digital signature with the private key of the certification authority which issued it.

**Definition 29** (Trust Anchor). A *trust anchor* is an entity that is trusted by a relying party and used for validating certificates in certificate chains.

**Definition 30** (End-Entity). An *end-entity* is an entity in the PKI that does not issue certificates.

**Definition 31** (Certification Path). A *certification path*, or a *certificate path*, is a sequence of certificates that link a certificate, referred to as the *leaf* or *end-entity* certificate, to a trust anchor. Each certificate in the chain, except the trust anchor certificate, is signed by a private key corresponding to a public key in another certificate in the chain. Such a chain is called a *certificate chain*.

**Definition 32** (Certification Authority). A *certification authority* (CA), or a *certificate authority*, is an authority trusted by one or more users to create and assign public-key certificates.

**Definition 33** (Root Certification Authority). A *root certification authority* is a CA that is a trust anchor. A *root certificate* is a certificate belonging to a root CA.

**Definition 34** (Intermediate Certification Authority). An *intermediate certification authority* (ICA) is a CA that is not a trust anchor.

In the motivating example (Figure 3.1), we only scratched the surface of the PKI ecosystem. One important detail we omitted is that certificates expire—they have a validity period during which this certificate is considered valid. In practice, it means that a verifier checking the CA signature checks if the current time is in the validity period. If not, the certificate is considered invalid even if the CA signature is valid.

The existence of the validity period is motivated by security concerns such as leakage of private keys and the fact that cryptographic algorithms are not expected to be secure forever. Estimating when a certain cryptographic algorithm becomes insecure is not easy, but it is somewhat doable;[2] however, estimating private key leaks is impossible as they can occur at any time. That is why there is a need for another mechanism using which a verifier can check if a private key is not compromised before its expiration date. This is what certificate *revocation* is for.

Certificate revocation is the process of invalidating a certificate before its validity period expires. There are essentially two methods by which a verifier can check if a certificate has been revoked: 1) using a Certificate Revocation List (CRL) [Boe+08] and 2) using the Online Certificate Status Protocol (OCSP) [San+13]. Section 3.1.4 is dedicated to these mechanisms.

In addition to that, there are PKI-related protocols for automating the creation of certificates such as the Certificate Management Protocol (CMP) [Mon+05] and the Automatic Certificate Management Environment (ACME) [Bar+19]. We examine these so-called certificate management protocols in Section 3.1.5.

[2] For example, by considering the cost of existing attacks and estimating computing power advances.

### 3.1.3 Certificates

In this work, our focus is on X.509 public-key certificates defined in [Boe+08] as this is the most widely used certificate format. Note that most of the content of this section also applies to *attribute certificates*, which do not contain public keys, but attributes instead. More precisely, only end-entity attribute certificates do not contain public keys. CA certificates must contain public keys since attribute certificates are still signed.

#### 3.1.3.1 Certificate Structure

X.509 certificates are defined using the Abstract Syntax Notation One (ASN.1) notation [ITU21b], and they are encoded using the Distinguished Encoding Rules (DER) [ITU21a].

A certificate has the following structure:

```
Certificate ::= SEQUENCE {
   tbsCertificate TBSCertificate,
   signatureAlgorithm AlgorithmIdentifier,
   signatureValue BIT STRING
}
```

- `tbsCertificate`: This field contains the *To-Be-Signed* (TBS) certificate which contains all certificate data except the CA's signature.
- `signatureAlgorithm`: This field identifies the signature algorithm used to produce the signature in `signatureValue`.
- `signatureValue`: This field contains the raw signature of `tbsCertificate`.

`TBSCertificate` has the following structure:

```
TBSCertificate ::= SEQUENCE {
   version [0] EXPLICIT Version DEFAULT v1,
   serialNumber CertificateSerialNumber,
   signature AlgorithmIdentifier,
   issuer Name,
   validity Validity,
   subject Name,
   subjectPublicKeyInfo SubjectPublicKeyInfo,
   issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,
   subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,
   extensions [3] EXPLICIT Extensions OPTIONAL
}
```

- `version`: This field describes the version of the X.509 standard. The typical value is version 3.

- `serialNumber`: This field contains the serial number of the certificate. The value is a positive integer. Serial numbers are unique per CA so that a serial number and the CA's identity uniquely identify a certificate.
- `signature`: This field identifies the signature algorithm used by the CA. This value is identical to the value of `signatureAlgorithm` in `Certificate`. Unlike `signatureAlgorithm`, this value is protected by the signature.
- `issuer`: This field contains the identity of the issuer, i.e., the CA that signed the certificate. The identity is in the form of a distinguished name—see Section 3.1.3.3.
- `validity`: This field defines the validity period of the certificate using two timestamps: `notBefore` and `notAfter`.
- `subject`: This field contains the identity of the entity associated with the public key (the subject) stored in the certificate (inside `subjectPublicKeyInfo`). The identity format is identical to `issuer`.
- `subjectPublicKeyInfo`: This field contains the public key. Specifically, the field contains information about the algorithm with which the key is used, and the encoded key as a bitstring. The encoding format varies based on the algorithm.
- `issuerUniqueID`: This field may be present in case the `issuer` value is not unique. This field is typically omitted.
- `subjectUniqueID`: This field is analogous to `issuerUniqueID`.
- `extensions`: This field contains a sequence of extensions, which allow for extending the certificate with other fields.

#### 3.1.3.2 Certificate Extensions

The `extensions` field allows for adding extra information into the certificate. The standard [Boe+08] defines a few standard extensions, but anyone can define their own.

Each extension can be marked as *critical*, or *non-critical*. If an extension is marked as critical, the verifier must reject the certificate if it cannot process such extension, e.g., because the verifier does not recognize it. Non-critical extensions must be correctly processed if supported by the verifier, but they can be ignored if the verifier cannot process them. A certificate cannot contain duplicate extensions.

The most common standardized extensions are:

- `authorityKeyIdentifier`: This extension contains information identifying the CA's public key used to sign the certificate. This is useful in cases where the same issuer may possess multiple signing keypairs. The key identifier is typically a truncated hash of the public key.
- `subjectKeyIdentifier`: This extension is analogous to `authorityKeyIdentifier` except that it identifies the subject's public key that is inside `subjectPublicKeyInfo`.

- `keyUsage`: This extension specifies the intended purpose, such as encryption or signing, of the keypair corresponding to the certificate.
- `certificatePolicies`: This extension identifies policies that apply to the certificate. Policies are typically indicated using a unique identifier or a URI.

  Certificate policies can be additional processing rules, or they may refer to legal aspects of the signatures corresponding to the certificate.
- `subjectAlternativeName`: This extension describes alternative identifiers of the subject. A common use case is a single website having multiple addresses, e.g., with or without "www."
- `basicConstraints`: This extension indicates two data points. The first is whether the subject is a CA, i.e., whether it can issue certificates. The second, which is optional, indicates the maximum depth of a valid certification path for the certificate.
- `extendedKeyUsage`: This extension essentially extends the `keyUsage` extension. It allows specifying more fine-grained usages as `keyUsage` defines only nine possible values.
- `cRLDistributionPoints`: This extension describes where to retrieve certificate revocation lists (Section 3.1.4.1), which is typically in the form of a URI.
- `authorityInfoAccess`: This extension describes where to retrieve information and services for the issuer. For example, this may contain a URI pointing to an OCSP resolver (Section 3.1.4.2).

#### 3.1.3.3 Distinguished Name

A *Distinguished Name* (DN) is the format used in the PKI context to identify an entity. A distinguished name is a sequence of identifying attributes. For example, the distinguished name of an employee named John Doe working as a cybersecurity researcher at the University of Luxembourg would be `CN=John Doe, OU=Cybersecurity research, O=University of Luxembourg, C=LU` where `CN` stands for "Common Name," `OU` for "Organizational Unit," `O` for "Organization," and `C` for "Country." Of course, each PKI may have different requirements on the specific DN structure.

#### 3.1.3.4 Certificate Creation

In this section, we describe how certificates are created, i.e., how they are issued. Note that we mainly focus on the cryptographic aspect of the process.

Recall that the entity issuing certificates is called the Certification Authority (CA). To be able to issue certificates, the CA needs to have its own keypair and the corresponding certificate. Therefore, before being able to issue other certificates, the CA needs to create its own keypair and issue a certificate to itself.

**Root Certification Authority Initialization.** Generating a keypair is trivial from a cryptographic point of view. In practice, the keypair is typically generated using a Hardware Security Module (HSM). Once the CA has the keypair, it creates a certificate for itself. Specifically, the CA creates the `TBSCertificate` structure and signs it using the generated private key. This certificate is

called a *self-signed* certificate, i.e., a self-signed certificate is a certificate where the issuer matches the subject. Consequently, the verification of the signature of a self-signed certificate is done using the public key in the certificate. Note that a self-signed certificate must serve as a trust anchor, i.e., it needs to be trusted implicitly, since anyone can create such a certificate.

**Remark**

Depending on the importance of the keypair, the keypair generation and subsequent private key manipulation may be audited/observed by multiple parties. Such a supervised keypair generation procedure is called a *key ceremony*.

**Subject Registration.** Once the CA has its own keypair, the CA can begin issuing certificates for its "subjects." The process starts with the subject registering at the CA. The registration process involves the subject proving its identity to the CA—this may require the subject to physically visit the CA and present its national identity card.[3] Additionally, depending on the specific CA's procedures, the CA may give the subject a token, e.g., in the form of a password, which may be used later by the subject to prove its identity over the internet.

**Remark**

We use the term "token" as an umbrella term for various scenarios. Some CAs may provide an interface for subjects to request certificates using a web application. In that case, the subject registers at the CA and receives its username-password pair which serves as the token. Later, the subject logs into the web application using the username-password pair (establishing a secure channel) and requests a certificate via the web interface. Another scenario may be that the CA provides an API endpoint which is secured using mutually authenticated TLS and the subject receives a client TLS certificate (with the corresponding private key) which it then uses to authenticate at the API endpoint to make certificate requests.

**Subject Certificate Creation.** After the registration process, the subject establishes a secure channel to the CA (using the token from the registration process) and sends a *certificate request* containing the subject's identity information, in the form of a distinguished name, and the public key of the subject. The CA must verify two things: 1) that the requestor (identified by the token) is truly the subject specified in the request and 2) that the requestor has the private key corresponding to the public key in the request. The verification of the requestor's identity is done by verifying the token owner. The second process of verifying the possession of the private key is done using a "Proof-of-Possession" (PoP) mechanism. For example, the requestor signs the request using the key. We discuss various PoP mechanisms in detail in Section 3.1.5.

Once the verification is done, the CA proceeds to create the `TBSCertificate` with the appropriate values, signs it with its private key and sends the certificate to the subject.

[3] In some sources [BKW13; Boe+08], the identity verification is said to be done by a *registration authority* (RA), which may be a separate entity from the CA. We do not make this distinction as the identity verification is not the focus of this work.

See Section 3.1.5 for implementation details regarding the certificate request and response procedures.

**Remark**

In our description, the keypair is assumed to be generated by the subject before making the request for the certificate creation; however, the keypair may be generated by the CA which then transfers the control of the private key to the subject. This may be useful in cases where certain regulations require that the keypair is generated by a certified device, e.g., to ensure proper key generation procedure with enough entropy [ETS23c, Section 6.5].

#### 3.1.3.5 Certificate Verification

Note that some sources distinguish between certificate *verification* and *validation*: verification refers to checking the signature of the certificate whereas validation refers to the complete process of establishing a valid certificate path. We use these terms interchangeably.

We sketch a simplified certificate path validation algorithm analogous to the one in [Boe+08, Section 6.1]. The exact algorithm is fairly involved mainly due to the complexity introduced by policies and constraints, which we mostly omit. Figure 3.2 provides the high-level description of the verification algorithm. We describe the algorithm by describing its sub-procedures: *Initialization*, *Process certificate*, *Prepare next certificate* and *Wrap-Up*.

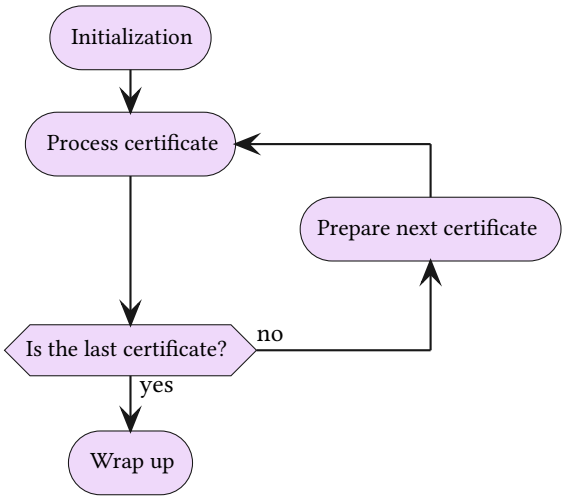



**Figure 3.2:** A high-level description of the certificate path validation algorithm from [Boe+08].

The goal of the certificate path validation algorithm is to verify the binding between the subject's identity and the subject's public key. At the beginning, we have the (unverified) certificate path/chain, which is a sequence of $n$ unique certificates $(c_1, \ldots, c_n)$, where $c_1$ is the trust anchor certificate, $c_n$ is the subject's certificate and $c_2, \ldots, c_{n-1}$ are ICA certificates. The algorithm verifies: 1) that $\forall i \in \mathbb{N} : 1 \leq i \leq n-1$ the subject of $c_i$ is the issuer of $c_{i+1}$, 2) that $c_1$ is the trust anchor certificate, and 3) that $\forall i \in \mathbb{N} : 1 \leq i \leq n : c_i$ was valid at the verification time.

**Remark**

Constructing a candidate certificate path is not trivial and there are algorithms for building one [Hes+05]. We omit this process as it is not cryptographic in nature and is therefore irrelevant to the security analysis of PKI in the context of quantum adversaries.

The inputs into the algorithm are: the certificate path without the trust anchor $(c_2, \cdots, c_n)$, the current time $t$ and the trust anchor $c_1$.

**Initialization.** At the start, the algorithm sets variables *pk* and *issuer* to values corresponding to the trust anchor $c_1$ and sets $i = 1$. It then proceeds with the next step of processing the certificate $c_i$ (*Process certificate*).

**Process Certificate.** This sub-procedure validates the certificate $c_i$ with respect to values *pk* and *issuer*. Specifically, it checks that $c_i$ is signed by *pk*, that `issuer` (the value in $c_i$) matches *issuer*, that the validity period includes the current time $t$, i.e., that `notBefore` $\leq t \leq$ `notAfter`, and that the certificate is not revoked (using methods described in Section 3.1.4).

If any check fails, the algorithm outputs "invalid path," otherwise the algorithm branches based on the value of $i$. If $i = n$, the algorithm goes to the *Wrap-Up* sub-procedure, otherwise it goes to the *Prepare next certificate* sub-procedure.

**Prepare Next Certificate.** In preparation for processing the next certificate, the algorithm updates the values *pk* and *issuer*: *pk* is set to the value of `subjectPublicKey` from $c_i$ and analogously for *issuer*. The algorithm makes additional checks verifying that $c_i$ is a CA certificate—by checking the values of `basicConstraints` and (`extended`)`keyUsage`. Additionally, it verifies other extensions and aborts if any critical extensions fail to validate. At the end, the algorithm increments $i$ by 1.

If any checks fail, the algorithm outputs "invalid path," otherwise it goes to *Process certificate*.

**Wrap-Up.** This sub-procedure is called after the last certificate $c_n$ was processed. It validates extensions and aborts if any critical extensions fail to validate. If there were no errors, the algorithm outputs "valid path," otherwise it outputs "invalid path."

#### 3.1.3.6 Quantum Threat to Certificates

The two cryptographic primitives used in certificates are signatures schemes and hash functions.

Recall from Section 2.5 that all classical signature schemes are insecure in the presence of a quantum adversary. Specifically, the quantum adversary is assumed to be able to compute the private key from a public key in "reasonable time." Consequently, all security guarantees provided by certificates are broken since the crucial assumption is that the underlying signature scheme is EUF-CMA-secure (Figure 2.6) which means that nobody, except the private key owner, can create signatures on their behalf (assuming the owner did not leak the private key).

For example, a quantum attacker could impersonate a user's bank website by forging its own certificate which would effectively give the attacker control of the user's bank account as it could man-in-the-middle the connection without anyone noticing.

A quantum attacker could also easily compromise any trust service provider by computing the root certificate's private key. Even though root private keys are "more secure," i.e., they have larger key sizes, in practice compared to intermediate, or end-entity certificates, Shor's algorithm runs in polynomial time (Section 1.4.1) so the attack difficulty increases negligibly.

The security of hash functions is also reduced by quantum computers due to Grover's algorithm (Section 1.4.2); however, the security loss is not catastrophic as in the case of signature schemes. Note that hash functions in this context are either used internally as part of signature schemes or they are used separately, e.g., to derive an extension value (`authorityKeyIdentifier`).

In summary, a PKI using a classical signature scheme is insecure—it does not provide any security guarantees against quantum adversaries. This includes all signature schemes listed in the current ETSI standard [ETS23a].

### 3.1.4 Revocation Checking

This section presents two main approaches to determining the revocation status of a certificate. The first method is using the Certificate Revocation List (CRLs), which are lists of revoked certificates periodically published by the CA (Section 3.1.4.1). The second method is using the Online Certificate Status Protocol (OCSP) which is a protocol for determining the revocation status in real-time (Section 3.1.4.2).

#### 3.1.4.1 Certificate Revocation List (CRL)

A Certificate Revocation List (CRL) is a list which includes information about all certificates revoked by a particular CA. Each CA publishes the list concerning all certificates issued by that CA. The CRL is either signed by the CA, or the CA can delegate the issuance of CRLs to other entities. Consequently, a verifier can verify its authenticity the same way it verifies any certificate's authenticity.

During certificate verification, the verifier checks if the certificate is not present in the list. If the certificate is not in the list, the verifier has proof that the certificate was not revoked at the time the list was issued.

A CRL has a similar structure to a certificate:

```
CertificateList ::= SEQUENCE {
   tbsCertList TBSCertList,
   signatureAlgorithm AlgorithmIdentifier,
   signatureValue BIT STRING
}
```

- `tbsCertList`: This field contains the *To-Be-Signed* (TBS) CRL which contains all revocation data except the CA's signature.
- `signatureAlgorithm`: This field identifies the signature algorithm used to produce the signature in `signatureValue`.
- `signatureValue`: This field contains the raw signature of `tbsCertList`.

The `TBSCertList` has the following structure:

```
TBSCertList ::= SEQUENCE {
   version Version OPTIONAL,
   signature AlgorithmIdentifier,
   issuer Name,
   thisUpdate Time,
   nextUpdate Time OPTIONAL,
   revokedCertificates SEQUENCE OF SEQUENCE {
      userCertificate CertificateSerialNumber,
      revocationDate Time,
      crlEntryExtensions Extensions OPTIONAL
   } OPTIONAL,
   crlExtensions [0] EXPLICIT Extensions OPTIONAL
}
```

- `version`: This optional field describes the version of the X.509 standard. If present, the version must be version 2.
- `signature`: This field identifies the signature algorithm used by the CA. This value is identical to the value of `signatureAlgorithm`.
- `issuer`: This field contains the identity of the issuer, i.e., the CA that signed the CRL. Analogous to certificates (Section 3.1.3.1), it is typically a triplet of strings describing the issuer's common name, organization and country.
- `thisUpdate`: This field indicates the creation/issuance time of the CRL.
- `nextUpdate`: This field indicates when the next CRL will be issued by the CA. The next CRL may be issued earlier, but it will not be issued later than indicated.
- `revokedCertificates`: This field contains the actual list of the revoked certificates. The list is in the form of a sequence of sub-sequences where each sub-sequence contains three data points:
  - `userCertificate`: This field contains the revoked certificate's serial number.
  - `revocationDate`: This field contains a timestamp indicating the time of revocation of the certificate.
  - `crlEntryExtensions`: This field contains an optional sequence of extensions containing additional data regarding the revocation. For example, it may contain the reason of revocation or the invalidity date, which may predate `revocationDate`.

  If there are no revoked certificates, the field is omitted.
- `crlExtensions`: This optional field contains a sequence of extensions containing additional data regarding the list. The standard [Boe+08] mandates that the extensions `authorityKeyIdentifier` and `cRLNumber` are present.

  `authorityKeyIdentifier` serves the analogous purpose as in a certificate.

  `cRLNumber` indicates the sequence number of the CRL, which must be larger than that in the previous CRL issued by the CA.

**Delta CRLs.** At the beginning of this section, we stated that a CRL contains information about all revoked certificates issued by a given CA. This is not entirely correct as CAs can also issue so-called *delta* CRLs, which contain only the changes from the last CRL. Delta CRLs are clearly more space efficient compared to regular (full) CRLs which consistently increase in size. The downside of delta CRLs is that the verifier needs to keep track of the changes. A delta CRL contains the extension `baseCRLNumber` which identifies the base CRL (the starting point) using the `cRLNumber` extension. The delta CRL combined with the base CRL forms the full CRL. The "reconstructed" CRL gets the updated `cRLNumber` of the delta CRL.

**Distribution of CRLs.** We did not explicitly mention how a verifier obtains a CRL. Recall from Section 3.1.3.2 that the certificate may contain the extension `cRLDistributionPoints`, which contains information about the location of CRLs issued by the CA. Typically, CRLs are distributed using HTTP at the URI location specified in the extension. Note that the transport protocol may be the *insecure* HTTP (as opposed to HTTPS) since the authenticity and integrity of the CRL is guaranteed by the signature in the CRL.

#### 3.1.4.2 Online Certificate Status Protocol (OCSP)

The Online Certificate Status Protocol (OCSP) [San+13] is a request-response protocol used by the verifier to obtain real-time revocation status of a certificate. The protocol is between two parties: the client (the verifier), who sends requests and the server (the responder), who issues responses.

First, the client must know how to communicate with the OCSP responder. The client either has the information preconfigured, or the common approach is to include the information inside the certificate using the extension `authorityInfoAccess`, which typically contains a URI of the responder (Section 3.1.3.2).

A CA may directly act as an OCSP responder for certificates it issued, signing the OCSP responses with the same key. Alternatively, the CA may delegate the issuance of OCSP responses to another entity. The CA issues a certificate for the delegated OCSP responder with the value `id-kp-OCSPSigning` in the `extendedKeyUsage` extension (Section 3.1.3.2).

**OCSP Request.** An OCSP request has the following structure:

```
OCSPRequest ::= SEQUENCE {
   tbsRequest TBSRequest,
   optionalSignature [0] EXPLICIT Signature OPTIONAL
}
```

where

```
TBSRequest ::= SEQUENCE {
   version [0] EXPLICIT Version DEFAULT v1,
   requestorName [1] EXPLICIT GeneralName OPTIONAL,
   requestList SEQUENCE OF Request,
   requestExtensions [2] EXPLICIT Extensions OPTIONAL
}

Signature ::= SEQUENCE {
```

```
   signatureAlgorithm AlgorithmIdentifier,
   signature BIT STRING,
   certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
}
```

Similarly to a certificate or a CRL structure, the outer layer is combining the internal `TBSRequest` with a potential signature. Note that unlike a certificate or a CRL structure, the OCSP request's signature is optional.

`TBSRequest` has the following fields:

- `version`: This optional field describes the version of the protocol. The default value is version 1.
- `requestorName`: This optional field identifies the client.
- `requestList`: This field contains a sequence of certificates for which the client is requesting the revocation data.

  Each certificate is identified, in the form of `CertId`, by the triplet (`issuerNameHash`, `issuerKeyHash`, `serialNumber`). `issuerNameHash` and `issuerKeyHash` are hashes of the `issuer` and `subjectPublicKeyInfo` fields from the certificate. The hash function used is specified in the request. `serialNumber` contains the serial number of the certificate.
- `requestExtensions`: This optional field contains extensions to include additional data related to the request. For example, one standard extension is used to indicate to the OCSP responder the preferred signature algorithm used to sign the OCSP response.

**OCSP Response.** An OCSP response has the following structure:

```
OCSPResponse ::= SEQUENCE {
   responseStatus OCSPResponseStatus,
   responseBytes [0] EXPLICIT ResponseBytes OPTIONAL
}

ResponseBytes ::= SEQUENCE {
     responseType OBJECT IDENTIFIER,
     response OCTET STRING
}
```

`responseStatus` is one of seven possible constants indicating whether there were any errors during response creation. Note that this is not the revocation status of the requested certificate(s).

`responseBytes` contains the actual response data. The standard is designed to support alternative response types, so there is an extra "wrapper" layer. The standard type is the *basic* type which has the structure of `BasicOCSPResponse`. The DER-encoded `BasicOCSPResponse` is the value of `response` in the `ResponseBytes` wrapper.

`BasicOCSPResponse` has the following structure:

```
BasicOCSPResponse ::= SEQUENCE {
   tbsResponseData ResponseData,
```

```
    signatureAlgorithm AlgorithmIdentifier,
    signature BIT STRING,
    certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
}
```

- `tbsResponseData`: This field contains the *To-Be-Signed* response in the form of `ResponseData`.
- `signatureAlgorithm`: This field identifies the signature algorithm used to produce the signature in the `signature` field.
- `signature`: This field contains the raw signature of `tbsResponseData`.
- `certs`: This optional field may contain certificates useful to the verifier to authenticate the response.

`ResponseData` has the following structure:

```
ResponseData ::= SEQUENCE {
    version [0] EXPLICIT Version DEFAULT v1,
    responderID ResponderID,
    producedAt GeneralizedTime,
    responses SEQUENCE OF SingleResponse,
    responseExtensions [1] EXPLICIT Extensions OPTIONAL
}
```

- `version`: This optional field describes the version of the protocol. The default value is version 1.
- `responderID`: This field identifies the OCSP responder either using a name or hash of its public key. This field is used to identify the public key used to verify the response.
- `producedAt`: This field indicates the timestamp when this response was created.
- `responses`: This field contains a sequence of individual responses corresponding to each request in `requestList`. Each individual response, in the form of `SingleResponse`, contains the certificate identifier (`certId`) (analogous to the one in the request), the revocation status of the certificate (`certStatus`), the most recent time at which the revocation status is known to have been correct (`thisUpdate`) and the time when newer revocation information will be available (`nextUpdate`). The response also may contain various extensions.

#### 3.1.4.3 Alternative Revocation Approaches

Both presented methods for revocation have downsides. CRLs are inefficient due to their large size, or they require keeping track of changes (delta CRLs). The OCSP protocol does not have these issues, but it introduces new ones. One issue with OCSP is that it requires the CA to have a robust infrastructure to keep up with the OCSP requests, which can be a single point of failure. Another issue is privacy, at least in the web context—OCSP leaks to the CA which websites the user is visiting. These issues motivate a shift to alternative methods for revocation checking.

**Short-Lived/Short-Term Certificates.** The current trend is to remove the need for revocation by issuing certificates with short lifetimes. In the Web PKI context, the suggested lifetimes are days or weeks [Let25; App24]. The argument for doing this is that certificate issuance has been largely automated using protocols such as ACME [Bar+19] (Section 3.1.5.4). In the eIDAS context, Trust Service Providers (TSPs) may also utilize so-called *short-term certificates* that are defined to be "certificates whose validity period is shorter than the maximum time to process a revocation request as specified in the certificate practice statement." [ETS23c] These short-term certificates may be valid for just a few seconds since they are created "on-the-fly" by the TSP for a specific operation, such as the production of an AdES signature for a customer.

**OCSP Stapling.** OCSP Stapling [Eas11, Section 8] is an alternative method by which to transport OCSP responses to the client in the web context. Specifically, the proposal extends TLS such that the TLS server sends an OCSP response to the client during the TLS handshake. Consequently, the client does not have to query the OCSP responder itself, which is the step that leaks privacy. The client can trust the OCSP response provided by the TLS server since it is signed by the OCSP responder.

#### 3.1.4.4 Quantum Threat to Revocation

We briefly analyze the threats posed by quantum attackers to revocation methods presented in Sections 3.1.4.1 and 3.1.4.2. Of course, in the broader context, where we assume the quantum attacker has access to a quantum computer, it does not make sense for the attacker to bother with falsifying revocation data for an existing expired/revoked certificate, if the attacker can simply issue itself a valid certificate. However, it is important to consider different scenarios, e.g., where the other parts of a PKI are already migrated to quantum-safe cryptography whilst the revocation checking component is still not.

Considering the threat to CRLs, the client trusts the content of a CRL because it is signed by the trusted issuer (typically the CA). If the CRL is signed by a classical signature, the quantum attacker can simply forge that signature, i.e., it can issue its own CRL.

Analogously to CRL, the OCSP client trusts the content of an OCSP response because it is signed by the trusted OCSP responder. In case the OCSP responses are signed by a classical signature scheme, then they can be forged by a quantum attacker.

In practice, the quantum attacker would need to additionally compromise the delivery channel of the revocation information. For example, CRLs/OCSP responses are typically distributed over insecure HTTP (since the CRL/OCSP response signature should already provide authenticity and integrity) and insecure HTTP is fairly easy to man-in-the-middle. In the case of "offline" signatures such as AdES signatures (Chapter 6), the adversary can easily replace the revocation data included in the signature for levels **B-LT**. In the case of **B-LTA** AdES signatures, the adversary would need to additionally compromise the timestamp signatures which cover the included revocation data—see attacks on timestamps in Section 3.1.6.

In summary, if CRLs or OCSP responses are signed using classical signature schemes, then these revocation methods are insecure, and they do not provide any security guarantees in the presence of a quantum adversary.

### 3.1.5 Certificate Management

In this section, we focus on protocols that are part of so-called certificate management, i.e., protocols that facilitate certificate issuance, certificate revocation, and similar operations. We primarily focus on the Certificate Management Protocol (CMP) [Mon+05] and its related format—Certificate Request Message Format (CRMF) [Sch05]. We also briefly present alternative certificate management protocols.

#### 3.1.5.1 Certificate Management Protocol (CMP)

Certificate Management Protocol (CMP) [Mon+05] is a protocol for certificate creation and management, i.e., it defines a common message structure which participants in a PKI can use to communicate. Among other things, it defines how to request a new certificate from a CA, how to manage key escrow[4] or how to request revocation of a certificate. We first present the common message structure and then dive into how one uses them to request and manage certificates.

**CMP Message Structure.** The CMP protocol is a request-response protocol between two entities in the PKI, where at least one is a CA. All CMP messages share a common structure called *PKIMessage*:

```
PKIMessage ::= SEQUENCE {
   header PKIHeader,
   body PKIBody,
   protection [0] PKIProtection OPTIONAL,
   extraCerts [1] SEQUENCE SIZE (1..MAX) OF CMPCertificate OPTIONAL
}
```

- `header`: This field contains the header, in the form of the `PKIHeader` structure, which always contains names (in the same format as in a certificate) of the sender and the intended recipient. The header can optionally contain other metadata such as public-key identifiers of the sender and the recipient, and nonces. For authenticated PKI messages it also contains the identifier of the protection algorithm.
- `body`: This field contains the body of the message. The content of the body depends on the specific CMP operation—we present a few later in this section.
- `protection`: This optional field either contains a Message Authentication Code (MAC) or a signature providing integrity and authenticity of the PKI message. Specifically, `header` and `body` are protected.
- `extraCerts`: This optional field contains certificates that may help the recipient with authentication of public keys inside `header`/`body`.

[4] Key escrow refers to the practice where an entity's private key is stored by a third party (CA).

**Certificate Request Message Format (CRMF).** Before we introduce the individual operations supported by CMP, we present the Certificate Request Message Format (CRMF) [Sch05]. This is a standalone format defining the structure of a certificate request, which is sent by an entity to a CA. The standard also defines the structure of the corresponding response from the CA.

The standard itself does not define a specific protocol; therefore, the CRMF messages can be used by any protocol. We present it since CMP utilizes CRMF messages for its certification operations.

The certificate request message `CertReqMsg` has the following structure:

```
CertReqMsg ::= SEQUENCE {
   certReq CertRequest,
   popo ProofOfPossession OPTIONAL,
   regInfo SEQUENCE SIZE(1..MAX) of AttributeTypeAndValue OPTIONAL
}
```

- `certReq`: This field contains the certificate template for the requested certificate and additional metadata, such as the request ID. We do not present the template format as it is out of scope but note that the template contains the entity's public key, which is meant to be certified by the CA.
- `popo`: This optional field contains the Proof-of-Possession (PoP), which is a proof that the requestor possesses the private key corresponding to the public key inside `certReq`. There are various ways the requestor can prove that.

  (a) *Encrypted private key method*: The requestor encrypts the private key and sends it to the CA. This method has an obvious disadvantage—it leaks the private key to the CA. The standard does not specify which key is used to encrypt the private key; it only states that the format is the CMS `EnvelopedData` [Hou09] (see Section 3.1.5.3). This method is technically possible for all kinds of private keys.

  (b) *Request signing method*: The requestor signs the request using the private key. Since the request contains the public key, the CA uses it to validate the signature. This method is only possible for signature private keys.

  (c) *Request DH-MAC method*: In the case that the requestor has the CA's public Diffie-Hellman key, it derives a shared key using its private key. The shared key is used to calculate a Message Authentication Code (MAC) over the request, proving its authenticity. The CA derives the same key using the requestor's DH public key and validates the MAC. This method is only possible for DH private keys.

  (d) *Encrypted challenge*: The requestor indicates in the request that it wants to prove possession interactively. The CA responds with an encryption of a random value using the public key inside the request. The requestor then sends the hash of the decrypted value. This method is possible for all public-key encryption private keys and also for Diffie-Hellman keys. The downside is that it requires an additional message exchange.

  (e) *Encrypted certificate*: This method is similar to the *Encrypted challenge* except that the CA encrypts the certificate instead of a random value. This indirectly proves

possession of the private key since only the private key owner will be able to get the plaintext certificate.

- `regInfo`: This optional field contains additional metadata related to the request.

Note that CRMF does not define how the response should look as it is meant to be defined by the protocol itself, which is in our case CMP.

**CMP Operations.** The CMP supports the following PKI management operations, each of which can be viewed as individual sub-protocols.

*Entity Initialization/Registration.* In the CMP context, entity initialization or registration is the process where an entity for the first time makes contact with the CA. At the end of this process, the CA issues the requested certificate for the entity.

The request bears the CRMF format presented in Section 3.1.5.1 (`CertReqMsg`). Specifically, the standard defines the request to be a sequence of `CertReqMsg` called `CertReqMessages`, i.e., the requestor can make multiple certificate requests in one CMP message (`CertReqMessages` is the `PKIBody` of `PKIMessage`).

The response `CertRepMessage` has the following format:

```
CertRepMessage ::= SEQUENCE {
   caPubs [1] SEQUENCE SIZE (1..MAX) OF Certificate OPTIONAL,
   response SEQUENCE OF CertResponse
}

CertResponse ::= SEQUENCE {
   certReqId INTEGER,
   status PKIStatusInfo,
   certifiedKeyPair CertifiedKeyPair OPTIONAL,
   rspInfo OCTET STRING OPTIONAL
}
```

- `caPubs`: This optional field contains CA-related certificates which may be of use to the requestor.
- `response`: This field contains a sequence of responses each corresponding to a `CertReqMsg` request. The sub-response has the structure of `CertResponse`:
  - `certReqId`: This field corresponds to the ID of the request.
  - `status`: This field indicates the status of the request. The status has multiple values indicating success, rejection, waiting etc.
  - `certifiedKeyPair`: This optional field contains the (potentially encrypted) certificate in case the request was successful. Additionally, it may contain an encrypted private key corresponding to the certificate in case the key generation was delegated to the CA.

– `rspInfo`: This optional field contains additional metadata related to the response.

For some certificates, e.g., those that use "encrypted certificate" as a PoP method, the CA may require the entity to send a certificate confirmation message proving that the entity received the correct certificate. CMP defines the `CertConfirmContent` structure to communicate that information in the form of a hash of the received certificate.

*Certification Request/Response.* In the CMP context, a certification request is made by an entity already known to the CA, compared to the initialization/registration process. This process is, however, essentially identical to the initialization/registration.

*Key Update.* This operation is invoked in the case that an entity wants to update the key inside its existing certificate. Due to its nature, the process is essentially identical to the certification request.

*Key Recovery.* A CA may escrow an entity's private key. In that scenario, CMP provides a way for the entity to request a copy of the private key from the CA. The request uses the CRMF structure containing the corresponding public key and its ID. The response uses its own structure `KeyRecRepContent`, which we do not present since it is out of scope. The important thing is that the private key is encapsulated inside the CMS structure [Hou09] `EnvelopedData` (see Section 3.1.5.3).[5]

*Revocation.* An entity may also request to revoke a certificate using CMP. It defines custom request/response structures `RevReqContent`/`RevRepContent` which are out of scope for this document. Note that the authentication mechanism is not specified—most likely since that depends on specific PKI policies—but it is presumably handled on a higher level using `PKIMessage` protection structures.

*Cross-Certification Request/Response.* A cross-certification request is essentially a certification request between two CAs. Therefore, the message structure is identical to standard certification request/response.

*Announcements.* The protocol supports various "announcement" messages such as announcing the issuance of a new CRL or that the CA updated their key (see Section 3.1.5.1 for details regarding a CA key update).

**Root CA Key Update Procedure.** Another mechanism that relates to CMP is the update of the root CA's keypair. Specifically, what steps the CA takes and how they are communicated to other entities in the PKI. A CA's keypair typically lasts longer than end-entity keypairs—its lifetime ranges from a couple of years to decades.

[5]Note that the CMP RFC [Mon+05] does not mention `EnvelopedData`, but a custom CRMF [Sch05] `EncryptedValue` structure. However, the CRMF RFC [Sch05] states that `EncryptedValue` was deprecated in favor of `EnvelopedData`.

The first step of this process is that the CA generates its new keypair $k_n$ to replace the old keypair $k_o$. The current certificate known to all PKI entities is the CA's self-signed certificate $C_{k_o}^{k_o}$ corresponding to $k_o$. To ease the transition, the CA creates three certificates:

- An *old-with-new* certificate $C_{k_o}^{k_n}$: This certificate contains the old public key corresponding to $k_o$, but it is signed using the new private key corresponding to $k_n$.
- A *new-with-old* certificate $C_{k_n}^{k_o}$: This certificate contains the new public key corresponding to $k_n$, but it is signed using the old private key corresponding to $k_o$.
- A *new-with-new* certificate $C_{k_n}^{k_n}$: This certificate is a self-signed certificate corresponding to the new keypair $k_n$.

Note that there is a transition period during which $C_{k_o}^{k_o}$ and $C_{k_n}^{k_n}$ are both valid. The purpose of these three certificates is to ease the keypair transition in various scenarios.

- Consider a verifier that still only trusts the old certificate $C_{k_o}^{k_o}$, which is still valid. However, the verifier may also need to correctly verify signatures produced with the new keypair $k_n$. It may do that if it receives the new-with-old certificate $C_{k_n}^{k_o}$. Thanks to $C_{k_n}^{k_o}$, the verifier can assemble a valid certification path ending with the old trust anchor ($C_{k_o}^{k_o}$).
- Consider a verifier that is in the opposite situation compared to the previous one—it replaced its trust anchor with the new certificate $C_{k_n}^{k_n}$. However, the verifier may still need to correctly verify old signatures produced with the old keypair $k_o$. It may do that if it receives the old-with-new certificate $C_{k_o}^{k_n}$. Analogously, such a certificate allows the verifier to build a certification path to the new trust anchor.

Note that this procedure is not directly related to the CMP protocol, but the CMP RFC [Mon+05] presents this process, and it also defines messages, which the CA may send, indicating the key update event to other PKI entities.

#### 3.1.5.2 PKCS #10

An alternative to CRMF (Section 3.1.5.1), which is also supported by CMP, is the PKCS #10 certificate request format. PKCS #10 is older compared to CRMF, but it is simpler since it only supports signature keypairs.

The certification request `CertificationRequest` has the following structure:

```
CertificationRequest ::= SEQUENCE {
   certificationRequestInfo CertificationRequestInfo,
   signatureAlgorithm AlgorithmIdentifier,
   signature BIT STRING
}

CertificationRequestInfo ::= SEQUENCE {
```

```
   version INTEGER { v1(0) } (v1,...),
   subject Name,
   subjectPKInfo SubjectPublicKeyInfo{{ PKInfoAlgorithms }},
   attributes [0] Attributes{{ CRIAttributes }}
}
```

where

- `certificationRequestInfo`: This field contains the entity identifier (name), the public key to be certified and potentially other attributes to be included into the certificate.
- `signatureAlgorithm`: This field identifies the signature algorithm used to sign the certification request.
- `signature`: This field contains the signature of `certificationRequestInfo`. The signature is made with the private key corresponding to the public key in `certificationRequestInfo`.

One of the disadvantages of the PKCS #10 format is that it requires that the keypair can be used as a signature keypair as it is used to sign the request, which serves as a PoP. Compared to CRMF, it does not support any other types of PoP.

#### 3.1.5.3 Cryptographic Message Syntax (CMS)—EnvelopedData

The Cryptographic Message Syntax (CMS) standard [Hou09] defines an "envelope" for encrypted data. The purpose of such an envelope structure is to combine a ciphertext with its metadata, e.g., which algorithm was used to produce the ciphertext and using which key. In the CMP context, the envelope, called `EnvelopedData`, is used to encrypt entities' private keys.

From a theoretical perspective, encrypting data as `EnvelopedData` is often an instance of hybrid encryption, which is a combination of a symmetric cipher with an asymmetric cipher: the creator encrypts the plaintext with a random symmetric key, referred to as the *content-encryption key*, and then encrypts the content-encryption key using the asymmetric cipher. However, this is not the only supported method to protect the content-encryption key as the standard [Hou09] also allows for encrypting the content-encryption key with another pre-shared symmetric key, a password-derived key, or a Diffie-Hellman shared key. We present these methods below. This hybrid design allows the sender to create a single envelope intended for multiple recipients, e.g., the sender encrypts the content-encryption key with each recipient's public key.

The envelope `EnvelopedData` has the following structure:

```
EnvelopedData ::= SEQUENCE {
   version CMSVersion,
   originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
   recipientInfos RecipientInfos,
   encryptedContentInfo EncryptedContentInfo,
   unprotectedAttrs [1] IMPLICIT UnprotectedAttributes OPTIONAL
}
```

- `version`: This is a CMS version identifier.

- `originatorInfo`: This optional field identifies the sender. The identifiers are in the form of certificates with relevant revocation data.
- `recipientInfos`: This field contains the encrypted content-encryption key using the recipient-specific method(s). The supported methods are the following:
  - *Key transport*: This method uses a public-key encryption scheme to encrypt the key. The corresponding structure `KeyTransRecipientInfo` contains the public-key encryption ciphertext along with data identifying the public key, e.g., the recipient's certificate, and the key-encryption algorithm.
  - *Key agreement*: This method uses a key agreement scheme, such as Diffie-Hellman, to derive a shared secret which is used to encrypt the content-encryption key. The corresponding structure `KeyAgreeRecipientInfo` contains the sender's public key (or its identifying information), identification of the key-encryption algorithm, identification of the recipient's public key, and the ciphertext containing the content-encryption key.
  - *Pre-shared key*: This method uses a pre-shared symmetric key as the key-encryption key which is used to encrypt the content-encryption key. The corresponding structure is `KEKRecipientInfo`, which contains the corresponding ciphertext and metadata identifying the key-encryption key.
  - *Pre-shared password*: This method is analogous to the pre-shared key method except the pre-shared information is a password which is used to derive a key-encryption key using a Key Derivation Function (KDF). Accordingly, the corresponding structure `PasswordRecipientInfo` must additionally contain identification of the KDF.
- `encryptedContentInfo`: This field contains the ciphertext containing the content. The structure also identifies the encryption algorithm used.
- `unprotectedAttrs`: This optional field may contain additional unencrypted metadata, e.g., the content-type.

#### 3.1.5.4 Alternative Certificates Management Protocols

We briefly present alternatives to CMP as it is not the only "certificate management" protocol.

**Simple Certificate Enrollment Protocol (SCEP).** The Simple Certificate Enrollment Protocol (SCEP) protocol [Gut20] is essentially a lightweight version of CMP. It utilizes PKCS structures: PKCS #10 (Section 3.1.5.2) for certificate requests and PKCS #7 (CMS) for certificate responses. Compared to CMP, the protocol is focused on certificate issuance and not certificate management in general.

**Certificate Management over CMS (CMC) and Enrollment over Secure Transport (EST).** The Certificate Management over CMS (CMC) protocol [MS08] is a more robust alternative to CMP while utilizing standardized structures from the CMS standard [Hou09]. The protocol is not as widely adopted as CMP.

The Enrollment over Secure Transport (EST) protocol [PYH13] builds upon CMC, and it serves as a modern alternative to SCEP.

**Automatic Certificate Management Environment (ACME).** The Automatic Certificate Management Environment (ACME) protocol is a modern web-PKI-focused alternative to CMP. It was primarily designed to automate certificate issuance for websites. Under the hood it uses the PKCS #10 certificate request format, HTTPS and JWS [JBS15a].

#### 3.1.5.5 Quantum Threat to Certificate Management Protocols

In this section, we presented the CMP protocol along with some of its building blocks—CRMF (Section 3.1.5.1), PKCS #10 (Section 3.1.5.2) and CMS EnvelopedData (Section 3.1.5.3). These protocols/standards utilize many cryptographic primitives. Specifically:

- CMP messages (`PKIMessage`) may be authenticated and integrity protected using a MAC or a signature scheme.
- The proof-of-possession (PoP) mechanisms used in CRMF and PKCS #10 rely on the security of the corresponding schemes. For example, signature key proof-of-possession relies on the signature scheme property that nobody without the private key can create a valid signature. Similarly, public-key encryption PoP methods rely on the PKE property that nobody without the private key can decrypt the ciphertext.
- The key escrow/retrieval mechanism in CMP (and the "encrypted private key" PoP method) use the CMS `EnvelopedData` structure which relies on the security of the public-key encryption scheme, or the key agreement method and the symmetric cipher (and potentially also on the security of the KDF or the MAC).

Note that the same applies to other certificate management protocols listed in Section 3.1.5.4 as they typically utilize similar methods for proof-of-possession etc.

Consequently, if these mechanisms rely on classical public-key cryptography, the quantum attacker will be able to compromise all security properties relying on those methods. Note, however, that the practical impact varies. For example, it does not make much sense for an attacker to forge the proof-of-possession since a quantum attacker is assumed to be able to calculate the corresponding private key from the public key. Furthermore, the reasonable strategy for the attacker is to attack the CA's private key directly instead, thereby circumventing the need to attack certificate management protocols.

### 3.1.6 Timestamping

A common service typically offered by trust service providers, along with a "regular" PKI, is a timestamping service. The service provider is called a Time-Stamping Authority (TSA). The purpose of a timestamping service is to provide cryptographic proof that certain data existed at a specific point in time. The proof is a signature of the TSA over the particular data and a timestamp. The TSA may be part of a PKI, or it may be a separate trust anchor. In our description, we consider that the TSA is an entity within a PKI.

#### 3.1.6.1 Time-Stamp Protocol (TSP)

To the best of our knowledge, the only widely standardized timestamping-related protocol is the Time-Stamp Protocol (TSP) [Zuc+01]. The standard defines PKI-related requirements for the TSA and also defines a request-response protocol, which can be used by other entities in the PKI to request timestamps on data. The standard also defines the structure of the timestamp proof, which is called a *timestamp token*.

A TSA is a special entity in a PKI whose certificate contains the extended key usage extension (Section 3.1.3.2) with the value `id-kp-timeStamping`[6], which indicates that the entity is authorized (in the PKI context) to produce timestamps.

**TSP Request** The protocol is a request-response protocol where an entity sends a timestamp request using a transport protocol to the TSA and the TSA responds with a timestamp response. The timestamp request `TimeStampReq` has the following structure:

```
TimeStampReq ::= SEQUENCE {
   version INTEGER { v1(1) },
   messageImprint MessageImprint,
   reqPolicy TSAPolicyId OPTIONAL,
   nonce INTEGER OPTIONAL,
   certReq BOOLEAN DEFAULT FALSE,
   extensions [0] IMPLICIT Extensions OPTIONAL
}
```

- `version`: This field indicates the protocol version.
- `messageImprint`: This field contains the hash of the data requested to be timestamped alongside the hash function identifier.
- `reqPolicy`: This optional field is used to indicate a policy identifier under which the timestamp token is to be produced. Similar to certificates, the policy number is to be included in the resulting timestamp token—assuming the TSA allows it.
- `nonce`: This optional field contains a nonce. The nonce is used to verify the freshness of the timestamp token in the case where the requestor has no notion of time. If included, the nonce is expected to be included in the timestamp token.
- `certReq`: This field indicates if the timestamp token should contain the signing (TSA's) certificate.
- `extensions`: This optional field contains additional metadata relating to the request.

[6] Of course, depending on the PKI setup and policies not any entity may be a trusted TSA.

**Remark**

Observe that only the hash of the data to be timestamped is sent to the TSA. This is because sending the entire data to the TSA is typically inefficient, and in practice a signature of a (collision-resistant) hash of data is considered equivalent to a direct signature over the data itself. Additionally, the sender may not want to disclose the data to the TSA.

**TSP Response** The timestamp response `TimeStampResp` has the following structure:

```
TimeStampResp ::= SEQUENCE {
   status PKIStatusInfo,
   timeStampToken TimeStampToken OPTIONAL
}
```

- `status`: This field indicates the status of the request, i.e., if the request was successful and the timestamp token issued, or if there was an error.
- `timeStampToken`: This optional field contains the resulting timestamp token if successful.

**TSP Token** The resulting token `TimeStampToken` is an instance of the CMS `SignedData` structure [Hou09].

The `eContentType` is set to the `id-ct-TSTInfo` OID indicating it is a timestamp token. The encapsulated content (`eContent`) is the `TSTInfo` structure described below. The `SignedData` must contain only one signature (an instance of `SignerInfo`) — the TSA's signature.

The `TSTInfo` has the following structure

```
TSTInfo ::= SEQUENCE {
  version INTEGER { v1(1) },
  policy TSAPolicyId,
  messageImprint MessageImprint,
  serialNumber INTEGER,
  genTime GeneralizedTime,
  accuracy Accuracy OPTIONAL,
  ordering BOOLEAN DEFAULT FALSE,
  nonce INTEGER OPTIONAL,
  tsa [0] GeneralName OPTIONAL,
  extensions [1] IMPLICIT Extensions OPTIONAL
}
```

- `version`: This field indicates the protocol version.
- `policy`: This field contains the policy identifier under which the timestamp token is produced.
- `messageImprint`: This field contains the hash of the timestamped data. This field is identical to the one in the request.

- `serialNumber`: This field contains the serial number of the timestamp token. Similar to certificates, this number must be unique per TSA.
- `genTime`: This field contains the actual timestamp when the timestamp token was produced, i.e., indicating that the data "in" `messageImprint` existed at time `genTime`.
- `accuracy`: This optional field indicates the accuracy of `genTime`.
- `ordering`: This optional field indicates if every timestamp token issued by the TSA can be ordered based on `genTime` alone regardless of `accuracy`.
- `nonce`: This optional field contains the nonce value specified in the request.
- `tsa`: This optional field additionally identifies the TSA.
- `extensions`: This optional field contains additional metadata relating to the timestamp token.

**Transport Protocols** The standard [Zuc+01] provides guidance on how to send TSP requests and responses using various transport protocols. Specifically, it provides guidance on how to transport them via email, any file transfer protocol such as FTP, a socket-based protocol such as TCP, or HTTP.

#### 3.1.6.2 Quantum Threat to Timestamping

A timestamp token is essentially a certificate, not a public-key certificate, but rather a "time certificate." In other words, it binds a point in time to some data and the binding is notarized in the form of a signature by a trusted third party, the TSA. Consequently, if the TSA is using a classical signature scheme, then its timestamps can be forged by a quantum adversary analogously to forging any other certificates.

For example, timestamps are commonly used to extend the validity period of signatures such as in the case of **B-LTA** AdES signatures presented in Chapter 6. The process assumes there is a signature which is produced using a private key whose certificate expires in the near future. The solution to extend its lifetime beyond the lifetime of the signer's certificate is to apply a timestamp over it before the certificate expires. The timestamp proves that the signature was created before the certificate expired (and therefore the private key is assumed to not be compromised at the time of signing). Consequently, a future verifier can verify the applied timestamp (we assume the timestamp has not expired at that point, otherwise the process can be repeated) and verify the original signature in the time-context before the original certificate expired.

In the context of quantum attackers, if the timestamp contains a classical signature, the verifier cannot trust the time indicated in the timestamp as the timestamp itself could be forged. A quantum adversary could for example present signatures made with a compromised signing key to be valid.

Consider the scenario where a signing certificate $c$ gets revoked at time $t$ due to private key compromise, i.e., any signature made with the private key dated after $t$ should be considered invalid. Since the quantum adversary is able to forge timestamps, it can produce a forged AdES signature (on behalf of $c$) with the compromised key and mark it as produced before the time $t$.

For example, the quantum adversary is able to produce a "valid" **B-LTA** signature as follows. It uses the compromised signing key to produce the initial basic signature under the identity of $c$, it includes revocation data issued prior to $t$ which does not contain information about revocation of $c$, and it includes a forged timestamp token dating the existence of the **B-LTA** signature before $t$. Note that in this scenario, the adversary does not need to attack the signer's keypair because we assume it is compromised already, and it does not even need to forge revocation data assuming it can retrieve older records before time $t$.

## 3.2 Purely Post-Quantum Public-Key Infrastructure

Section 3.1 presented components of a Public-Key Infrastructure (PKI) and analyzed their security with respect to quantum adversaries. The conclusion of the section is that classical PKI, i.e., a PKI relying on classical signatures, is inherently vulnerable to quantum adversaries.

To address the issue, we introduce a *post-quantum PKI*, i.e., a PKI that is resistant to quantum adversaries. We propose three post-quantum PKI variants: the *purely post-quantum PKI* in this section and two hybrid variants in Section 3.3.

Recall from Section 2.4 that purely post-quantum solutions utilize solely post-quantum algorithms. Consequently, purely post-quantum solutions are typically simpler to implement compared to hybrid post-quantum solutions; however, their threat model does not consider the risk of the post-quantum algorithm being broken.

On the other hand, hybrid post-quantum solutions combine classical and post-quantum algorithms. This approach essentially hedges against potential new attacks against post-quantum algorithms. Note that hybrid post-quantum solutions become redundant in the presence of a quantum adversary since we know that the classical algorithm will be broken. However, they are an ideal solution for the transition phase during which cryptographically relevant quantum computers do not yet exist and post-quantum algorithms have not yet been battle-tested. Another benefit of hybrid solutions is that they allow deployment of quantum-safe cryptography while being compliant with existing regulations, which typically lag behind the state of the art and still require classical algorithms.

Taking this into account, defining a "purely post-quantum PKI" is straightforward—it is the same as a traditional (classical) PKI except that classical signatures, such as ECDSA or RSA, are replaced with post-quantum signatures, such as ML-DSA or SLH-DSA.

Defining a "hybrid post-quantum PKI" is not as simple. Recall that the main purpose of a PKI is to provide a mechanism to authenticate public keys. Consequently, a *classical PKI* provides an authentication mechanism that relies solely on classical cryptography. Analogously, a *purely post-quantum PKI* relies solely on post-quantum cryptography. The purpose of a *hybrid post-quantum PKI*, as with any hybrid post-quantum solution, is to hedge against two threats—the *expected* compromise of classical algorithms by quantum adversaries and an *unexpected* compromise of a post-quantum algorithm by any (even classical) adversary.

### 3.2.1 Post-Quantum PKI Terminology

Let us define these PKI-related terms more formally.

**Definition 35** (Classical PKI). We say a PKI is *classical* if its security depends on the security of classical cryptography.

Note that the definition is intentionally vague since PKI is essentially an umbrella term for many components as described in Section 3.1. Our aim with these definitions is to give the reader an idea of what we mean when we say *classical PKI*. We define the *purely post-quantum PKI* analogously.

**Definition 36** (Purely Post-Quantum PKI). We say a PKI is *purely post-quantum* if its security depends *solely* on the security of post-quantum cryptography.

In other words, purely post-quantum PKI does not mix classical and post-quantum cryptography, which is what *hybrid post-quantum PKI* does.

**Definition 37** (Hybrid Post-Quantum PKI). We say a PKI is *hybrid post-quantum* if it utilizes both classical and post-quantum cryptography and is constructed such that it remains secure as long as at least one of the cryptographic components remains secure.

Analogously, we define post-quantum certificates.

**Definition 38** (Classical/Post-Quantum Certificate). A *post-quantum* certificate is a certificate that is signed using a post-quantum signature scheme (Definition 21). Similarly, a *classical* certificate is a certificate that is signed using a classical signature scheme (Definition 20).

The important implication of Definition 38 is that a classical certificate is forgeable by a quantum adversary whereas a post-quantum certificate is not (assuming the post-quantum signature scheme is not broken).

Furthermore, we distinguish between *purely post-quantum* certificates and *hybrid post-quantum* certificates.

**Definition 39** (Purely/Hybrid Post-Quantum Certificate). We call a certificate *purely post-quantum* if it is signed using only one post-quantum signature scheme. We call a certificate *hybrid post-quantum* if it is signed by one classical signature scheme and one post-quantum signature scheme.

The implication of Definition 39 is that a purely post-quantum certificate becomes forgeable in the event that there exists a classical attack against the post-quantum scheme. On the other hand, a hybrid post-quantum certificate does not become forgeable during such an event (assuming a classical adversary).

However, if the post-quantum signature scheme gets classically broken *and* a quantum adversary is already present, then the hybrid post-quantum certificate becomes forgeable. Therefore, the benefit of hybrid post-quantum certificates, or rather hybrid post-quantum authentication in general, is to minimize the risk before the existence of quantum adversaries.

**Remark**

Recall from Section 2.4.4 that the IETF has defined terminology related to post-quantum cryptography [DPH25], which we do not adopt in full in this thesis.
In particular, our definitions of classical and (hybrid) post-quantum certificates (Definitions 38 and 39) classify a certificate according to the signature scheme used by its issuer, rather than according to the public key contained inside the certificate. This choice reflects the security property considered in this thesis: protection of the binding between the subject identity and its public key against a quantum adversary. Under the IETF terminology, a certificate containing a post-quantum public key but signed with a classical signature scheme may be called a post-quantum certificate, even though a quantum adversary could forge that certificate. In our terminology, such a certificate is classical, since its authenticity relies on a classical signature scheme.
Finally, the parallel PKI construction discussed in Section 3.3.1 corresponds closely to the IETF notion of a parallel PKI.

### 3.2.2 Purely Post-Quantum Certificates

Let us begin by describing purely post-quantum X.509 certificates (Definition 39). A classical certificate is forgeable since the signature of the issuer is made using a classical signature scheme. Specifically, the `signatureValue` is the raw signature of the `TBSCertificate` made using the algorithm specified in `signatureAlgorithm`.

Therefore, a purely post-quantum certificate must be signed by a post-quantum signature algorithm. Specifically, the `signatureValue` will contain the raw post-quantum signature of the `TBSCertificate` made by the issuer with the post-quantum signature algorithm specified in `signatureAlgorithm`.

There are no restrictions on the contents of `subjectPublicKeyInfo` in purely post-quantum certificates. The end-entity's public key may correspond to a post-quantum or a classical public key primitive (a signature scheme, a KEM, etc.). On the other hand, intermediate and root certificates must contain a public key corresponding to a post-quantum signature scheme as in the classical case.

#### 3.2.2.1 Real-World Instantiation

Even though post-quantum signature schemes are equivalent, in terms of their cryptographic functionality, to classical signature schemes, they significantly differ in terms of performance—primarily in terms of bandwidth and size. Recall from Section 2.4.3.2 that almost all post-quantum signature schemes have at least 10× larger public keys or signatures.

One question is whether the current X.509 standard [Boe+08] imposes any size restrictions on the signatures and public keys present in the certificates, which would complicate migration by requiring a modification of the standard. Thankfully, the standard does not limit the size, however, implementations may impose their own non-standard size limits.

The other question is the performance impact in terms of 1) the increase certificate size 2) the potential slowdown of signing and verification. One may infer the impact by simply looking at the differences, e.g., a typical certificate contains one public key and one signature.

Therefore, the increase in the size of a post-quantum certificate, compared to a classical certificate, is $(pk_{PQ} - pk_C) + (sig_{PQ} - sig_C)$ where $pk_{PQ}, pk_C$ are the sizes of the post-quantum and classical public key in bytes, respectively, and analogously for $sig_{PQ}, sig_C$. For example, by taking numbers from Table 2.2, we get that a purely post-quantum certificate containing an ML-DSA-44 (security level 2) public key and signature is 3636 bytes larger than a classical certificate containing an Ed25519 public key and signature.

It is also important to evaluate the relative increase in size. To address this, we present sizes of certificates signed by (and containing the public keys of) various signature algorithms in Table 3.1. Note that the relative increase depends on the "complexity" of the certificate, e.g., which optional fields and extensions are present. In our measurements, we consider a self-signed certificate with the following extensions: `authorityKeyIdentifier`, `subjectKeyIdentifier`, `keyUsage`, `basicConstraints`, `cRLDistributionPoints` and `authorityInfoAccess`. We can estimate the overhead caused by all non-signature-related fields using values from Table 3.1—the overhead is around 500 bytes.

Recall from Table 2.2 that FN-DSA is the most size efficient post-quantum signature (meaning it has the smallest combined size of the public key and the signature). We see that, even assuming the typical certificate overhead, an FN-DSA security level 1 certificate is roughly 3× the size of a classical ECDSA certificate and 1.6× the size of a classical RSA certificate. Compared to the second most size efficient post-quantum signature, ML-DSA, an FN-DSA certificate is roughly 2× smaller compared to ML-DSA. An ML-DSA-44 (security level 2) is more than 6× the size of an ECDSA certificate and over 12× the size at the highest security level 5.

Even though SLH-DSA's public key sizes are on par with classical signature schemes, its significant signature size dominates the size of the certificate. Even the most size-efficient (and security level 1) parameter set $(128, f)$ produces a certificate which is roughly 2.3× the size of an ML-DSA-87 certificate (the highest security parameter set).

**Remark**

See [Oup25a] for an experimental post-quantum PKI implementation. Although it is focused on post-quantum Advanced Electronic Signatures, discussed in Chapter 6, a post-quantum PKI is a necessary component of such an implementation.

**Example Purely Post-Quantum Certificate.** Figure 3.3 shows an example of a purely post-quantum certificate using ML-DSA-44 as the signature. The textual representation of the certificate was created using OpenSSL.[7] The only differences compared to a classical certificate are the signature algorithm field and the actual values for the subject public key info and the signature value.

#### 3.2.2.2 Hash Functions in Certificates

Recall from Section 3.1 that, apart from signature schemes, certificate creation relies on hash functions as well, and not only "indirectly" in signature schemes. For example, the standard extensions `authorityKeyIdentifier` and `subjectKeyIdentifier` contain hashes of the issuer's

[7] Using the command `openssl -x509 -in <certificateFile> -text`.

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 110 (0x6e)
      Signature Algorithm: ML-DSA-44
      Issuer: CN=Purely PQ CA 1, O=University of Luxembourg, OU=APSIA, C=LU
      Validity
         Not Before: Nov 27 14:05:12 2024 GMT
         Not After : Sep 27 13:05:12 2026 GMT
      Subject: CN=Jan Oupický, O=University of Luxembourg, OU=APSIA, C=LU
      Subject Public Key Info:
         Public Key Algorithm: ML-DSA-44
            ML-DSA-44 Public-Key:
            pub:
               <public key bytes>
      X509v3 extensions:
         X509v3 Key Usage: critical
            Non Repudiation
         X509v3 CRL Distribution Points:
            Full Name:
              URI:http://pq.pki/pkicrl/purelypq.crl
         Authority Information Access:
            OCSP - URI:http://pq.pki/pkiocsp/purelypq
            CA Issuers - URI:hhttp://pq.pki/pkicrt/purelypqca1.crt
         X509v3 Subject Key Identifier:
            91:3A:4E:FA:30:0A:E0:77:E0:D0:5F:AF:90:23:F4:EC:3F:C1:2C:C9
   Signature Algorithm: ML-DSA-44
   Signature Value:
      <signature bytes>
```

**Figure 3.3**: ML-DSA-44 Purely Post-Quantum Certificate Example

**Table 3.1:** Post-Quantum Certificate Sizes (bytes)

| Signature Algorithm | Certificate Size | Relative Increase w.r.t. ECDSA | Relative Increase w.r.t. RSA |
|---|---|---|---|
| ECDSA (P-256) | 627 | 0% | −50% |
| RSA (3072 bits) | 1 278 | 103% | 0% |
| ML-DSA-44 | 4 222 | 573% | 230% |
| ML-DSA-65 | 5 751 | 817% | 350% |
| ML-DSA-87 | 7 709 | 1 129% | 503% |
| FN-DSA (512) | 2 030 | 223% | 58% |
| FN-DSA (1024) | 3 545 | 465% | 177% |
| SLH-DSA (128,f) | 17 606 | 2 707% | 1 277% |
| SLH-DSA (128,s) | 8 374 | 1 235% | 555% |
| SLH-DSA (256,f) | 50 406 | 7 939% | 3 844% |
| SLH-DSA (256,s) | 30 342 | 4 739% | 2 274% |

and subject's public key, respectively. Even though the purpose of these extensions is not security-critical—they are meant to aid verifiers with identification of public keys—it is important to note that in the case that a certain extension relies on, e.g., pre-image resistance or collision-resistance of a hash function, the hash function must be upgraded accordingly by doubling its output length (Section 1.4.2).

The same applies to other cryptographic primitives used in extensions, e.g., web PKI certificates compliant with Certificate Transparency [LLK13] (CT) include Signed Certificate Timestamps (SCTs) which themselves contain signatures by the CT log. These extensions and related systems will also need to be migrated to achieve a fully post-quantum PKI.

### 3.2.3 Transition to a Purely Post-Quantum PKI

In this section, we discuss the process of transitioning to a purely post-quantum PKI. There are two possible scenarios:

1) Transitioning from a classical PKI to a purely post-quantum PKI.

2) Transitioning from a hybrid post-quantum PKI to a purely post-quantum PKI.

#### 3.2.3.1 Transition From a Classical PKI

A transition from a classical PKI to a purely post-quantum PKI assumes that there was never a hybrid period. This is likely to occur in some scenarios as migration to post-quantum authentication may not be as time-sensitive as migration to post-quantum key exchange and the adopters wait until the confidence in (purely) post-quantum cryptography rises.

In theory, the transition is no different from a regular PKI transition to a newer signature scheme. In the past, there have been migrations from RSA to ECDSA, or key sizes have simply been increased. The only fundamental difference is that post-quantum signatures and their public keys

are larger, which may introduce some unexpected issues due to non-compliant implementations—as discussed in this document, no PKI-related standard imposes size restrictions.

#### 3.2.3.2 Transition From a Hybrid Post-Quantum PKI

The transition from a hybrid post-quantum PKI to a purely post-quantum PKI depends on the type of hybrid post-quantum PKI. We consider two types of hybrid post-quantum PKI: 1) a parallel hybrid PKI (Section 3.3.1) where two separate PKIs are used in parallel and 2) a composite hybrid PKI (Section 3.3.2) which utilizes hybrid post-quantum certificates.

The parallel (hybrid) PKI model inherently provides a smooth transition from hybrid post-quantum PKI to purely post-quantum PKI. Since the two PKIs are separate and independent of each other, the transition "only" requires the deactivation of the classical PKI and the post-quantum PKI can remain in place unchanged. This deactivation can be realized in the form of not issuing new certificates, waiting for the expiration of the root certificate, and instructing end-users to skip the "classical verification step" while relying only on the post-quantum certificates.

The composite hybrid PKI is harder to transition than the parallel PKI. One might, analogously to the parallel PKI transition, leave the PKI in place while instructing users to disable the classical verification step of the composite signatures. However, this is practically infeasible as, on an implementation level, this operation is typically handled by a low-level library, whereas the modification required for a parallel PKI transition is typically handled at a level higher. Another disadvantage is that the hybrid certificates contain redundant information — the classical signatures and public keys.

A more realistic approach to the transition from a composite hybrid PKI is to handle the transition analogously to the transition from a classical PKI to a purely post-quantum PKI. This transition is essentially identical, since in theory a composite hybrid PKI is simply a PKI whose signature algorithm happens to be hybrid post-quantum.

### 3.2.4 Purely Post-Quantum Certificate Revocation Lists

Recall from Section 3.1.4.1 that Certificate Revocation Lists (CRLs) provide one of the most common ways to perform certificate revocation checking.

A CRL must be signed by the CA to ensure the validity of its contents—a list of revoked certificates. A CRL structure is similar to a certificate: the field `signatureValue` contains the raw signature of `tbsCertList` made using the algorithm specified in `signatureAlgorithm`.

Consequently, it is straightforward to make a purely post-quantum CRL—sign it with a post-quantum signature. In a purely post-quantum CRL, the `signatureValue` will contain the raw post-quantum signature of `tbsCertList` from the CA with the post-quantum signature algorithm specified in `signatureAlgorithm`.

As in the certificate case, the standard [Boe+08] does not impose any size restrictions. Therefore, nothing restricts the use of post-quantum signatures.

Purely post-quantum CRLs are obviously going to suffer from an increase in size, just like purely post-quantum certificates. However, the increase in size is not practically relevant as CRLs themselves are not designed to be size-efficient.

**Delta CRLs.** The same applies to "partial" CRLs such as delta CRLs. The only difference is that delta CRLs are smaller compared to regular CRLs and consequently the relative size increase may be more noticeable.

### 3.2.5 Purely Post-Quantum Online Certificate Status Protocol

Online Certificate Status Protocol (OCSP) is an alternative way to perform certificate revocation checking. Recall from Section 3.1.4.2 that the OCSP protocol is a request-response protocol where the requests are optionally signed, but responses must be signed by the OCSP responder.

OCSP requests are typically not signed by the requestor. However, upgrading OCSP requests to purely post-quantum security is a straightforward process, as the structure is analogous to certificates and CRLs.

OCSP responses *must* be signed by the OCSP responder to ensure the validity of their contents. `BasicOCSPResponse` follows the same envelope-like structure as certificates and CRLs. Therefore, the replacement of a classical signature with a post-quantum one is analogous.

OCSP was designed to be a lightweight alternative to CRLs. Consequently, the size of OCSP responses is an important metric. Unsurprisingly, OCSP responses signed with a post-quantum signature are larger since they contain the signature value and potentially optional certificate(s) related to the OCSP responder, which contain a public key and a signature.

In Table 3.2 we present sizes of OCSP responses signed by various algorithms. The OCSP responses measured contained only a single response, i.e., they did not correspond to multiple OCSP requests, and did not contain any optional certificates from the responder. We see that, compared to certificate sizes presented in Table 3.1, the gap between FN-DSA and ML-DSA widens. This is because certificates contain both the public key and the signature, whereas an OCSP response contains only the signature. FN-DSA's signatures are slightly smaller than its public keys whereas ML-DSA's signatures are roughly double the size of its public keys (Table 2.2). This suggests that signature schemes with small signatures and potentially large public keys, such as QR-UOV or MAYO, are ideal for OCSP responses. However, the OCSP responder's public key needs to be distributed somehow to the requestor. Most likely, the public key is part of the validated certificate which prompted the OCSP request. Therefore, in the broader context, using such signature schemes does not improve performance.

### 3.2.6 Purely Post-Quantum Certificate Management Protocol

In Section 3.1.5.1 we have presented the basics of the Certificate Management Protocol and analyzed its vulnerabilities with respect to quantum adversaries. We have concluded that there are two parts of the protocol that need to be upgraded.

#### 3.2.6.1 PKIMessage Authentication

All CMP messages share a common structure `PKIMessage`. A `PKIMessage` can be optionally "signed" by either a signature scheme or a message authentication code (MAC) to provide integrity and authenticity of the content.

The standard supports two methods for deriving the symmetric MAC key. The first method is using a pre-shared secret and the second is to derive a shared key using Diffie-Hellman—it

Table 3.2: Post-Quantum OCSP Response Sizes (bytes)

| Signature Algorithm | OCSP Response Size | Relative Increase w.r.t. ECDSA | Relative Increase w.r.t. RSA |
|---|---|---|---|
| ECDSA (P-256) | 243 | 0% | −57% |
| RSA (3072 bits) | 566 | 133% | 0% |
| ML-DSA-44 | 2 600 | 970% | 359% |
| ML-DSA-65 | 3 489 | 1 336% | 516% |
| ML-DSA-87 | 4 807 | 1 878% | 749% |
| FN-DSA (512) | 829 | 241% | 46% |
| FN-DSA (1024) | 1 439 | 492% | 154% |
| SLH-DSA (128,f) | 17 268 | 7 006% | 2 951% |
| SLH-DSA (128,s) | 8 036 | 3 207% | 1 320% |
| SLH-DSA (256,f) | 50 036 | 20 491% | 8 740% |
| SLH-DSA (256,s) | 29 972 | 12 234% | 5 195% |

is assumed the sender knows the recipients DH public key. The first method does not rely on public-key cryptography; therefore, at most it requires doubling of the pre-shared secret. The second method relies on DH and is therefore insecure. However, it is possible to replace Diffie-Hellman with a KEM. The sender is assumed to have the recipient's public key and can therefore run encapsulation and use the resulting key as the MAC key. Note that the MAC algorithm will also need to be updated to quantum-safe parameters to counteract Grover's algorithm impact (Section 1.4.2).

Apart from "signing symmetrically" using a MAC, we can sign using a signature. Using a post-quantum signature scheme is straightforward as in the other cases.

#### 3.2.6.2 Purely Post-Quantum Certificate Requests

Recall from Section 3.1.5 that there are two commonly used certificate request formats, Certificate Request Message Format (CRMF) and PKCS #10, and both of these formats rely on cryptographic mechanisms to prove the possession of the private key (PoP).

**PKCS #10.** PKCS #10 provides only one method for PoP, namely signing the request using the private key. This PoP method can clearly be used with any post-quantum signature as well.

**CRMF.** CRMF supports five PoP methods, and only some of them can be used with post-quantum algorithms:

1) *Encrypted private key method*: This method simply encrypts the private key under a shared secret. This method is agnostic of the private key's purpose and can therefore be used with any post-quantum algorithm. Note that the encryption method must be quantum-safe; see Section 3.2.7.

2) *Request signing method*: Identical to PKCS #10's PoP.

3) *Request DH-MAC method*: For this method, there is no post-quantum alternative since it relies on the non-interactivity property of Diffie-Hellman. In a KEM scenario, the sender knowing the receiver's public key cannot derive a shared key that also depends on the sender's private key. Note that it is possible to derive a shared key, but that requires an additional round of communication which becomes essentially the "encrypted challenge" PoP.

4) *Encrypted challenge*: This method works for any public-key encryption scheme, i.e., it can be used with post-quantum KEMs.

5) *Encrypted certificate*: This method is similar to the encrypted challenge, and therefore it is also compatible with any post-quantum KEM.

Note that for all methods there are no size restrictions which would make the existing standards incompatible with post-quantum algorithms with large sizes.

In summary, four of the five classical CRMF PoP methods can be used with post-quantum algorithms. The one method that cannot be translated to the post-quantum case since it relies on unique Diffie-Hellman properties, which standardized post-quantum algorithms do not possess, is the DH-MAC method.

**Remark**

The PoP method of signing the request is intended to be used for signing keys only, however, an RSA keypair can be used for encryption and signing. This led some real-world systems to use this method of PoP even for RSA encryption keys (during issuance of a PKE certificate) [OC25]. Note that no post-quantum signature scheme has this RSA-specific property.

### 3.2.7 Purely Post-Quantum CMS EnvelopedData

Recall from Section 3.1.5.3 that the CMS standard [Hou09] defines an envelope format for encrypted data that is used in some PKI contexts. For example, this is the format used to encrypt the private key during the "Encrypted private key method" PoP (Section 3.2.6.2).

The basis of the encryption part of the standard is the `EnvelopedData` structure which primarily contains 1) the ciphertext containing the data encrypted using a symmetric cipher under a symmetric key and 2) the ciphertext containing the symmetric key encrypted using public-key encryption or information identifying the symmetric key using other means. The condition 2) is intentionally vague since the standard supports multiple ways to determine the symmetric key.

To make `EnvelopedData` quantum-safe there are two things to migrate. The first one is the symmetric cipher. Recall from Section 1.4.2 that it is recommended to use ciphers with 256-bit keys, such as AES-256. The second thing to migrate is public-key encryption used to encrypt the symmetric key.

As mentioned, the standard supports four methods for handling the symmetric key:

1) *Key transport*: This method uses a public-key encryption algorithm to encrypt the symmetric

key using the recipient's public key. This method can be used with any post-quantum KEM.[8]

2) *Key agreement*: This method relies on non-interactivity of Diffie-Hellman. Consequently, there is no post-quantum alternative for this method.

3) *Pre-shared key*: This method does not rely on any public-key cryptographic primitives, so there is no quantum threat (expect for doubling the key size where necessary).

4) *Pre-shared password*: Analogous to the pre-shared key method.

### 3.2.8 Purely Post-Quantum Time-Stamp Protocol

The Time-Stamp Protocol (TSP) presented in Section 3.1.6.1 essentially relies on two cryptographic primitives which need to be migrated. One is the hash function used to generate the message fingerprint inside `messageImprint`. The other primitive is the signature scheme used by the Time-Stamping Authority (TSA). Also recall that timestamp tokens are fairly similar to public-key certificates except they bind a point in time to some data.

Migrating to a quantum-safe hash function for the message imprint is straightforward as with other symmetric components.

The timestamp token `TimeStampToken` uses the CMS signature structure, which we discuss in detail in Chapter 6, and consequently there are also no issues with simply using a post-quantum signature scheme instead of a classical one to provide post-quantum security.

## 3.3 Hybrid Post-Quantum Public-Key Infrastructure

In Section 3.2, we have already defined and hinted at possible realizations of a "hybrid post-quantum PKI." In contrast to the purely post-quantum PKI, whose security relies purely on post-quantum algorithms, the hybrid post-quantum PKI hedges against an expected compromise of classical algorithms by quantum adversaries or an unexpected compromise of novel post-quantum algorithms by even classical adversaries. The *parallel hybrid post-quantum PKI* presented in Section 3.3.1 is one realization of a hybrid post-quantum PKI. We present an alternative, referred to as the *composite hybrid post-quantum PKI*, in Section 3.3.2.

### 3.3.1 Parallel Hybrid Post-Quantum Public-Key Infrastructure

The parallel hybrid post-quantum PKI design is fairly straightforward. The idea is to use two separate PKIs in parallel where one PKI is classical, and the other one is purely post-quantum. The classical PKI can be "brand new," or it may be an already existing PKI which gets complemented by a purely post-quantum one. Recall from Section 3.2 that setting up the purely post-quantum PKI is straightforward.

#### 3.3.1.1 Parallel PKI Variants

We distinguish between two variants of a parallel hybrid post-quantum PKI.

[8] A minor technical difference is that a KEM generates the symmetric key during the encapsulation whereas using a PKE the encrypting party must generate the key "on its own."

**Same Subject, Same Public Key.** The first approach considers that there is a single public key that is to be authenticated in a hybrid manner, i.e., there exist two certificates for the same public key and subject. Observe that this is essentially a form of cross-signing except "at the end-entity level." This approach does not have many realistic use cases. Consider the possible public keys:

1) Classical signing key: A post-quantum certificate containing a classical public key is superfluous, because if the classical signature scheme is compromised, the public key inside the certificate must not be used to validate signatures in any case. Consequently, hybridization does not provide any benefit.

2) Post-quantum signing key: Analogous to the previous point.

3) Classical PKE/DH key: First, we consider quantum adversaries. In the presence of quantum adversaries, the classical certificate is compromised and the classical PKE/DH keypair must not be used anyway. Now, if we consider a compromise of the post-quantum signature, then the post-quantum certificate is compromised. Consequently, in both scenarios the post-quantum certificate is superfluous.

4) Post-quantum KEM key: Analogous to the previous point.

**Same Subject, Different Public Keys.** This second approach considers that the subject has two distinct keypairs, a classical keypair and a (purely) post-quantum keypair, and each keypair has its own corresponding certificate. Specifically, a classical certificate corresponding to the classical keypair and a post-quantum certificate corresponding to the post-quantum keypair. The relying party that wants to "hybridly" authenticate the subject receives both certificates and validates each certificate independently.

Note that this approach inherently assumes that both keypairs are used in some protocol. For example, there is a proposal [She+25] for hybrid authentication in TLS which uses this approach—the TLS transcript is signed in parallel by both keys and the server provides both certificate chains. Our hybrid post-quantum SAML proposal (Chapter 5) also utilizes parallel PKI for authentication. Hybrid AdES signatures (Chapter 6) may also rely on the parallel PKI (in the case they are hybrid at issuance).

**Conclusion**

As discussed, the first approach does not provide any meaningful benefits. Consequently, we focus on the second approach, i.e., we consider that in a parallel PKI there is a classical certificate and a post-quantum certificate with the same subject, but different public keys.

#### 3.3.1.2 Parallel PKI Example

Figure 3.4 illustrates the parallel PKI in action. The relying party (RP) wants to verify that the classical and post-quantum public keys, *cPubKey* and *pqPubKey*, belong to John Doe. The RP is provided with two certificates: a classical one binding *cPubKey* to John Doe and a purely post-quantum one binding *pqPubKey* to John Doe. The RP trusts both PKIs' root certificates. To authenticate John Doe in a hybrid post-quantum sense, the RP must verify both certificates independently and consider the binding to be valid if and only if both certificates verify successfully.

Note that the verification process for each certificate is unchanged and proceeds in a standard non-hybrid manner including revocation checking as both PKIs are independent of each other. For the classical PKI, this is a standard procedure, and we have shown in Section 3.2 that for the purely post-quantum PKI the procedure is essentially the same, except with post-quantum signatures.

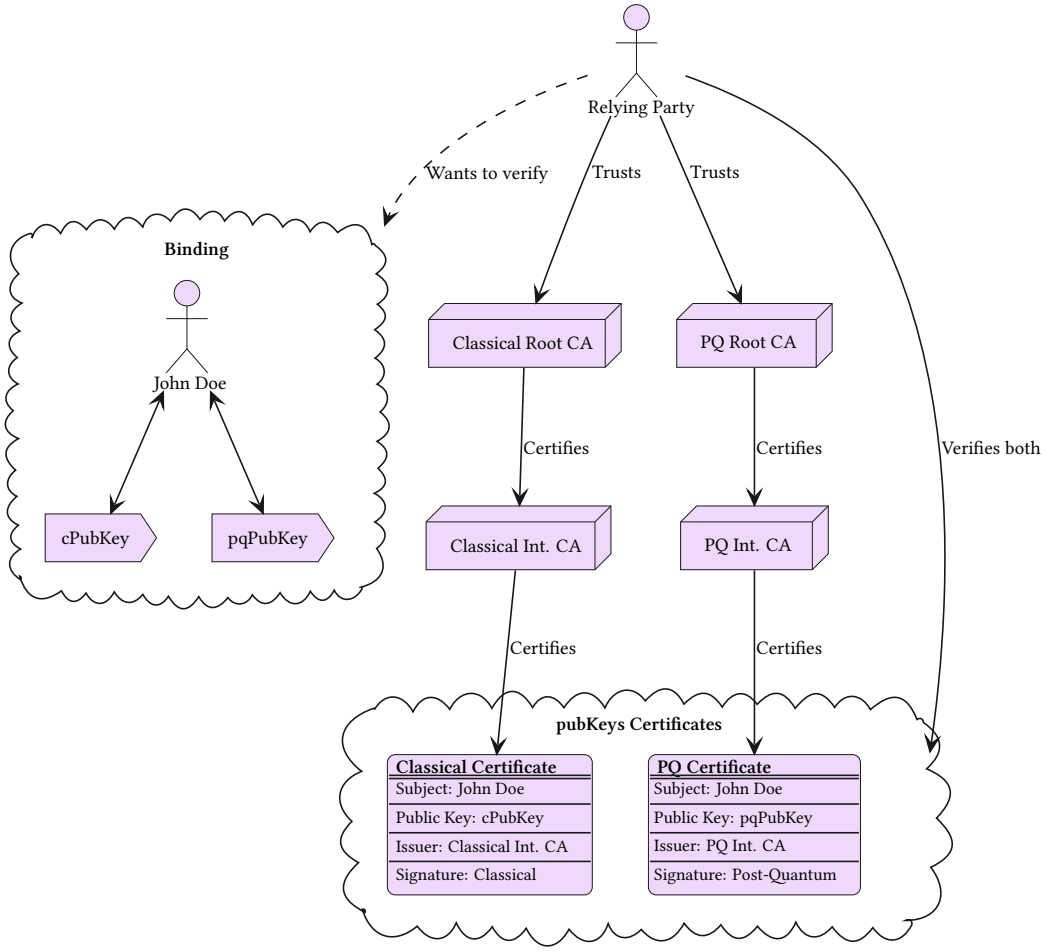



**Figure 3.4:** Parallel Hybrid Post-Quantum PKI Example

#### 3.3.1.3 Impact of Parallel PKI on Participants

Due to its nature, using the parallel hybrid post-quantum PKI is not as simple as using the purely post-quantum PKI discussed in Section 3.2. We discuss the impact on each participant in the parallel PKI.

From the CA's standpoint, operating a parallel hybrid PKI is technically equivalent to operating two independent PKIs. The bigger impact of parallel PKI is on the PKI users. We consider two kinds of PKI users: subscribers and relying parties. A subscriber is the one who is issued the certificate, i.e., the holder of the private key corresponding to the public key in the certificate. The relying party is the one that utilizes the PKI to verify the binding between the identity and the public key.

**Impact on Subscribers.** A subscriber of a parallel hybrid PKI must request a certificate from each component PKI (instead of just one). Furthermore, the subscriber must provide both certificates (instead of just one) where necessary. During revocation, the subscriber may revoke only the one impacted certificate and continue using the other one since they do not correspond to the same keypair.

**Impact on Relying Parties.** The RP is required to do twice as many steps to authenticate the binding—it must verify the classical certificate chain and also the post-quantum certificate chain. To achieve hybrid post-quantum security, the RP must consider the binding to be valid if and only if both certificates are valid (and their subjects match). It may be that due to its policies, the RP runs in a non-hybrid mode, i.e., it requires only one of the certificates to be valid. In that case, the relying party ignores one of the parallel PKI components at the cost of losing hybrid post-quantum security.

#### 3.3.1.4 Transition Timeline for Parallel Hybrid PKI

As with other hybrid solutions, the hybrid post-quantum PKI is meant to be used before the purely post-quantum PKI (Section 3.2). In this section we present the transition timeline for the parallel hybrid PKI.

Let us define several points in time:

- $T^s_{PQ}$: the `notBefore` date of the purely post-quantum root certificate,
- $T^e_C$: the `notAfter` date of the classical root certificate,
- $T_{QC}$: the point in time when quantum adversaries become a threat.

Then we call the time period from $T^s_{PQ}$ to $T^e_C$ the *parallel hybrid PKI period*, meaning that this is the period during which the parallel hybrid post-quantum PKI can be used. Note that $T^e_C$ also marks the beginning of the purely post-quantum period since only the purely post-quantum PKI can be used. Note that in our diagram $T_{QC}$ occurs after $T^e_C$, but it may also be the case that $T_{QC}$ occurs during the parallel hybrid PKI period in which case the post-quantum component of the parallel PKI is crucial for security.

The expected transition timeline is shown in Figure 3.5. At the start, there is only a classical PKI, which corresponds to the situation before PQC migration begins. Sometime during the classical PKI's lifetime, the purely post-quantum PKI begins its operation (at $T^s_{PQ}$). After this point, subscribers and relying parties can begin using the parallel hybrid PKI—the subscribers can request their "extra" post-quantum certificates and the relying parties can begin verifying the certificates. At a later point, a few years before $T^e_C$,[9] the classical PKI stops issuing classical certificates due to the proximity of the root certificate expiration date ($T^e_C$).

After the classical root certificate expires at $T^e_C$, all classical certificates are invalid and subscribers and relying parties must switch to purely post-quantum mode, i.e., subscribers need to only provide the post-quantum certificate and relying parties need to validate only the post-quantum certificate.

[9] "Few years" depends on the specific PKI policy that asserts the lifetime of issued certificates.

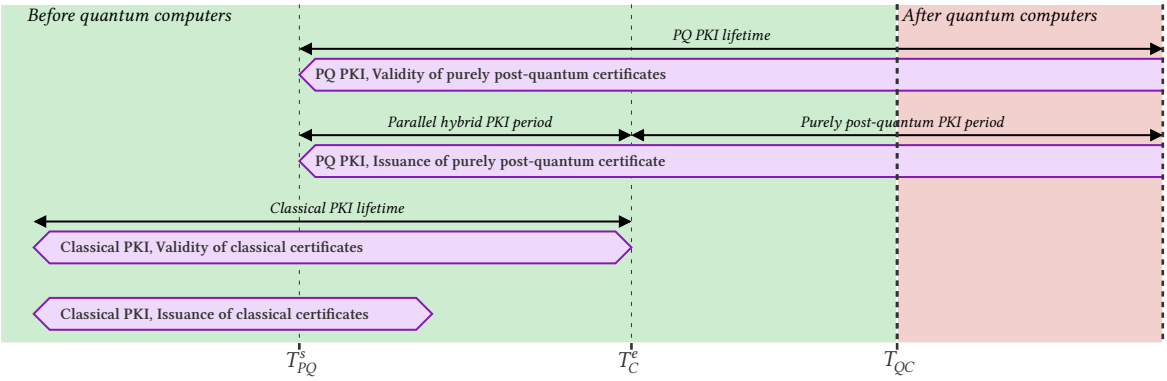



**Figure 3.5:** Parallel Hybrid PKI Transition Timeline

#### 3.3.1.5 Backward Compatibility

One of the main benefits of the parallel PKI approach is that, compared to the composite hybrid post-quantum approach (Section 3.3.2), it allows for "backward-compatible" solutions in certain cases. Specifically, certain cryptographic protocols/mechanisms can be made hybrid post-quantum secure while also allowing them to be processed by classical-only verifiers, e.g., see our post-quantum SAML solution in Chapter 5 or hybrid post-quantum AdES signatures in Chapter 6 which may ignore the post-quantum timestamp.

#### 3.3.1.6 Possible Modifications

We presented one flavor of the parallel hybrid PKI, but there are possible modifications of this approach. The first makes the two certificates explicitly bound together, and the second uses one *hybrid* certificate instead of two regular certificates.

**Dual Certificate Binding.** The two certificates used during hybrid authentication share the same subject, but they are not explicitly bound together, which may be necessary in certain scenarios. [BGJ25] proposes to solve this using a new certificate extension which binds one certificate to another. The binding is one way, i.e., certificate A can be bound to certificate B, or vice versa, but they cannot both be bound to each other. This is because the binding relies on the hash of the certificate which is unpredictable before the certificate is issued.

**Hybrid Certificates.** The parallel hybrid post-quantum PKI we presented relies on two separate certificates. Effectively, the only difference between these two certificates is the subject public key value and the signature value of the issuer. Because having two separate certificates is sometimes inconvenient, there are proposals to "merge" these certificates into one [ITU19; Bon+25].

*Alternative Signature Extension.* The ITU-T X.509 standard from 2019 [ITU19] defines three new extensions:

- `subjectAltPublicKeyInfo`: This extension contains an alternative subject public key for the certificate.

- `altSignatureValue`: This extension contains a signature by the issuer made with an alternative signature algorithm specified in `altSignatureAlgorithm`.
- `altSignatureAlgorithm`: This extension contains the identifier of the alternative signature algorithm.

During certificate issuance, the issuer first signs the certificate with the alternative signature algorithm, and then it signs the certificate with the classical algorithm. Specifically, the alternative signature is applied over the `TBSCertificate` with the `signature` field and the `altSignatureValue` extension excluded. The classical signature is applied in a standard manner over the whole `TBSCertificate` (with the alternative signature present). Note that this extension approach is sometimes referred to as "Catalyst"—the name given by ISARA Corporation, which held a patent for it, but today it is in the public domain [ISA22].

Having one hybrid certificate allows easier migration in certain scenarios since modifying existing solutions to accommodate two certificates instead of one may not be straightforward. The approach is also backward compatible: a classical-only verifier expects one certificate and the PQ-related certificate extensions will be ignored since they are not critical extensions.

The downside is that it makes an eventual migration to purely post-quantum PKI more complex than the approach using two certificates. Consider the scenario in which quantum adversaries become active during the parallel hybrid period. With two-certificate parallel PKI, the users simply agree to stop sending and processing the redundant classical certificate and keep using the purely post-quantum one. If users only have these hybrid certificates, they can agree to ignore the classical part of the certificate. However, it still forces them to use certificates that contain redundant classical data.

Furthermore, one must be careful when deploying this standardized solution for several reasons:

- The standard says that the alternative signature is "to be checked instead of the native digital signature." This suggests that it is not required to verify the default signature. This behavior—verifying *only* the alternative signature—is expected since the original purpose of this extension is to ease migrating from an old signature scheme to a new one, not to provide hybrid security. This implies that verifiers that want hybrid post-quantum security must set a policy to verify both signatures.
- The "order" of signature algorithms matters:
  - If the default signature is classical and the alternative signature is post-quantum, then the certificate is backward compatible, meaning that classical-only verifiers that do not even support the (alternative) extensions are able to verify the certificate.
  - If the default signature is post-quantum and the alternative signature is classical, then the certificate is technically backward-compatible in the same way as the previous case; however, it will most likely not be in practice. This is because this extension is fairly novel and classical-only verifiers, which are unlikely to adopt the latest standards quickly, will most likely not support the extensions. Consequently, they will not be able to verify the certificate as it is a purely post-quantum certificate in their view. Additionally, consider a non-hybrid PQ verifier, e.g., a verifier that prefers PQ

signatures, but for backward-compatibility reasons still supports classical signatures. Such a verifier will treat the hybrid certificate as classical since only the alternative signature shall be verified according to the standard. This results in an "unintentional downgrade."

*Chameleon Certificates.* Similar to the alternative extension approach, [Bon+25] proposes a mechanism to reconstruct the second (post-quantum) certificate from a "difference/delta information" provided in a single extension.

### 3.3.2 Composite Hybrid Post-Quantum Public-Key Infrastructure

Compared to the parallel hybrid post-quantum PKI (Section 3.3.1), the composite hybrid post-quantum PKI is a single PKI in which the signature algorithm itself is hybrid post-quantum secure. Recall from Section 2.4.5.1 that a composite hybrid signature is a combination of a classical signature and a post-quantum signature.

The main advantage of the composite hybrid approach is that it provides hybrid security using a lower-level component—the composite hybrid signature scheme (Definition 24). This generally makes migrations simpler, since we exchange one signature scheme with another, unlike in the case of parallel PKI where the hybrid security needs to be handled on a higher level. Consequently, the migration process is essentially identical to the migration to a purely post-quantum signature.

Figure 3.6 illustrates the composite hybrid PKI in action. Observe that it is identical to how a classical PKI or a purely post-quantum PKI is utilized to verify a public key's authenticity.

Consequently, we do not go into the details of the composite hybrid post-quantum PKI as the solutions are analogous to those of a purely post-quantum PKI which has been discussed in detail in Section 3.2. However, we still highlight a few differences between these two approaches and present an example composite hybrid post-quantum certificate in Section 3.3.2.1.

#### 3.3.2.1 Example Composite Hybrid Post-Quantum Certificate

Figure 3.7 shows an example of a composite hybrid post-quantum certificate with the composite signature from ECDSA (P-256) + ML-DSA-44. The textual representation of the certificate was created using OpenSSL.[10] Note that compared to the purely post-quantum certificate presented in Section 3.2.2.1, OpenSSL (version 3.6) does not recognize the composite signature and consequently prints its raw Object Identifier (OID), which in this case is `2.16.840.1.114027.80.9.1.9`.

As we can see, the only difference compared to a classical (or a purely post-quantum) certificate is that the signature algorithm field is different along with the actual values for the subject's public key and the signature value.

**Remark**

See [Oup25a] for an experimental post-quantum PKI implementation. Although it is focused on post-quantum Advanced Electronic Signatures, discussed in Chapter 6, a post-quantum PKI is a necessary component of such an implementation.

[10] Using the command `openssl -x509 -in <certificateFile> -text`.

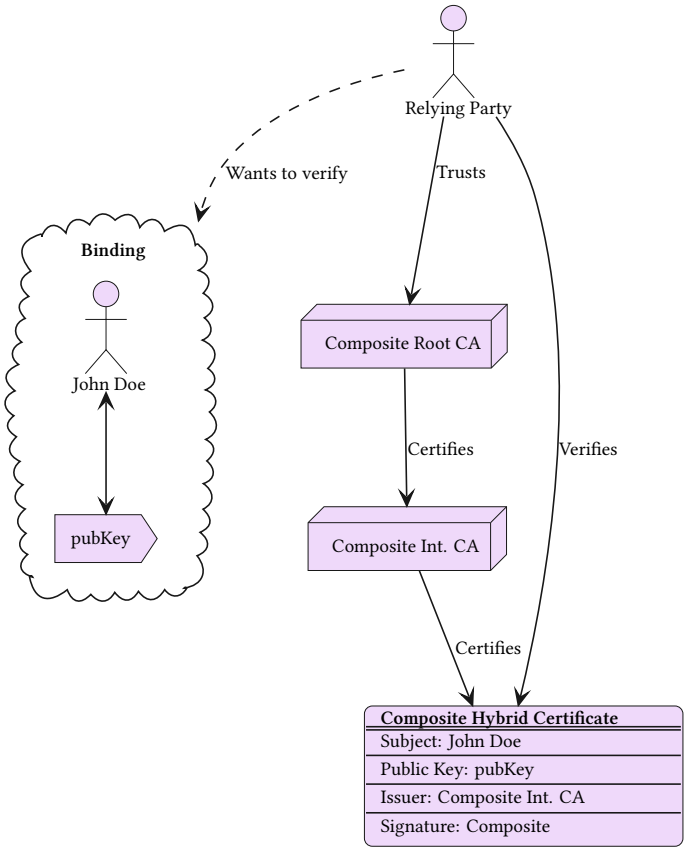



**Figure 3.6:** Composite Hybrid Post-Quantum PKI Example

#### 3.3.2.2 Composite Hybrid Performance

We discuss the performance of composite hybrids compared to purely post-quantum solutions and parallel PKI solutions.

**Composite PQ vs. Purely PQ.** Composite hybrid post-quantum signatures are inherently less performant than purely post-quantum signatures since they employ the additional classical component.

Considering size, the difference is relatively small because classical signatures are significantly more size efficient. For example, consider the (Ed25519, ML-DSA-44) composite hybrid. An Ed25519 signature is 64 bytes whereas an ML-DSA-44 signature is 2 420 bytes which makes the relative size increase around 3%. An Ed25519 public key is 32 bytes whereas an ML-DSA-44 public key is 1 312 bytes which also makes the relative size increase around 2%.

Regarding speed of operations, the slowdown of composite hybrid signatures, compared to purely post-quantum signatures, is potentially more significant since the speed of some post-quantum signatures is on par with classical signatures. For example, considering the (Ed25519, ML-DSA-44) composite hybrid and results from Table 2.2, we see that ML-DSA-44 is around 5× slower at signing compared to Ed25519, but it is effectively equal at verification. Consequently, the composite hybrid is around 22% slower at signing compared to pure ML-DSA-44 signing and around 95% slower at verification. If we average it out, we can roughly say that the composite

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 111 (0x6f)
      Signature Algorithm: 2.16.840.1.114027.80.9.1.9
      Issuer: CN=Composite PQ CA 1, O=University of Luxembourg, OU=APSIA, C=LU
      Validity
         Not Before: Nov 27 14:06:40 2024 GMT
         Not After : Sep 27 13:06:40 2026 GMT
      Subject: CN=Jan Oupický, O=University of Luxembourg, OU=APSIA, C=LU
      Subject Public Key Info:
         Public Key Algorithm: 2.16.840.1.114027.80.9.1.9
            2.16.840.1.114027.80.9.1.9 Public-Key:
            pub:
               <public key bytes>
      X509v3 extensions:
         X509v3 Key Usage: critical
            Non Repudiation
         X509v3 CRL Distribution Points:
            Full Name:
              URI:http://pq.pki/pkicrl/compositepq.crl
         Authority Information Access:
            OCSP - URI:http://pq.pki/pkiocsp/compositepq
            CA Issuers - URI:http://pq.pki/pkicrt/compositepqca1.crt
         X509v3 Subject Key Identifier:
            91:3A:4E:FA:30:0A:E0:77:E0:D0:5F:AF:90:23:F4:EC:3F:C1:2C:C9
   Signature Algorithm: 2.16.840.1.114027.80.9.1.9
   Signature Value:
      <signature bytes>
```

**Figure 3.7**: ECDSA (P-256) + ML-DSA-44 Composite Hybrid Post-Quantum Certificate Example

hybrid is 1.45× slower compared to the purely post-quantum signature. However, if we consider less performant post-quantum signature schemes, such as SLH-DSA, then the slowdown is not as significant.

**Composite PQ vs. Parallel PQ.** On paper, the verification of a composite certificate requires the same number of signature operations as the verification of two (non-composite) certificates using the parallel PKI.

However, the parallel PKI inherently brings some overhead. For example, the two certificates used in the parallel PKI contain redundant information, such as the subject; both certificates need to be hashed during verification instead of just one; revocation checking requires consulting two separate CRLs/OCSP responders, and much more.

We created a composite hybrid post-quantum certificate and an OCSP response for the composite (ECDSA (P-256), ML-DSA-44) using the same conditions as described in Section 3.2.2.1. The composite certificate's size is 4 388 bytes whereas the combined size of separate ECDSA and ML-DSA-44 certificates is $627 + 4\,222 = 4\,849$ bytes, which means that in this setup the parallel PKI has an overhead of $\approx 11\%$. For OCSP responses, the overhead is smaller ($\approx 6\%$) since in an OCSP response the signature value is typically a larger percentage of the total size compared to a public-key certificate.

On a lower level, signing and verification of composites is also slightly faster than performing signing and verification separately for each component twice as demonstrated, e.g., in Chapter 5.

#### 3.3.2.3 Backward Compatibility

Compared to the parallel hybrid PKI approach, the composite hybrid PKI approach is inherently not backward compatible. This is because it relies on a new signature scheme, which classical-only verifiers cannot handle. Even though under-the-hood the composite hybrid signature contains a classical signature, the classical-only verifier does not know how to parse it.

## 3.4 Conclusion

This chapter investigated post-quantum PKI approaches. Because PKI is a dependency for virtually all other cryptographic protocols discussed in this thesis—including TLS, SAML, and AdES—its migration is a critical prerequisite.

We began by recalling PKI components such as certificates, revocation, and timestamping, which helped us identify the quantum-vulnerable parts of PKI.

In Section 3.2, we demonstrated that setting up a purely post-quantum PKI is similar to setting up a new classical PKI—it is simply the replacement of one signature scheme with another. One notable difference is that post-quantum artifacts, such as certificates and CRLs, are significantly larger, which may introduce technical challenges in use cases where size efficiency is crucial. We also identified certain PKI-related mechanisms that do not migrate straightforwardly because of their reliance on Diffie-Hellman properties, such as some proof-of-possession methods, but these can easily be replaced by alternative methods. Another non-technical aspect is that setting up a purely post-quantum PKI may be difficult in the early stages of post-quantum adoption due to the unavailability of necessary hardware or outdated regulations.

The last section, Section 3.3, was dedicated to the topic of hybrid post-quantum PKI. We presented two approaches: the parallel hybrid post-quantum PKI (Section 3.3.1) and the composite hybrid post-quantum PKI (Section 3.3.2).

The parallel hybrid post-quantum PKI refers to the paradigm of using a classical PKI in tandem with a purely post-quantum PKI. Consequently, the PKI setup essentially reduces to setting up a purely post-quantum PKI given that the classical PKI already exists. This approach shifts the handling of hybrid post-quantum authentication to the higher-level cryptosystem or protocol—the verification of hybrid signatures or the handling of two separate certificate chains. On the other hand, this approach allows for an easier future transition to purely post-quantum PKI (Section 3.3.1.4) compared to the composite hybrid post-quantum approach.

The composite hybrid post-quantum PKI is a single PKI that uses a composite hybrid post-quantum signature scheme. Since a composite hybrid post-quantum signature is a regular signature, from an interface perspective, setting up a composite hybrid post-quantum PKI is identical to setting up a purely post-quantum PKI. Compared to the parallel hybrid post-quantum PKI approach, which shifts the handling of hybrid authentication to the higher level of the cryptosystem or protocol, this approach does the opposite; it shifts handling to the lower level—the hybrid post-quantum authentication is handled by the cryptographic primitive itself.

# Chapter 4

# Post-Quantum TLS


4.1 **Overview of TLS** . . . 98
4.1.1 Cryptographic Building Blocks . . . 99
4.1.2 Handshake Protocol . . . 99
4.1.3 Quantum Threat to TLS . . . 101
4.2 **Post-Quantum Authentication in TLS** . . . 102
4.2.1 Purely Post-Quantum Authentication in TLS . . . 102
4.2.2 Hybrid Post-Quantum Authentication . . . 104
4.2.3 Merkle Tree Certificates . . . 106
4.3 **Post-Quantum Key Exchange in TLS** . . . 107
4.3.1 Purely Post-Quantum Key Exchange in TLS . . . 107
4.3.2 Hybrid Post-Quantum Key Exchange in TLS . . . 111
4.4 **Post-Quantum Authenticated Key Exchange in TLS** . . . 113
4.4.1 KEMTLS . . . 113
4.4.2 Mutually Authenticated Key Exchange . . . 116
4.5 **Symmetric Primitives in Post-Quantum TLS** . . . 117
4.5.1 Symmetric Ciphers . . . 117
4.5.2 Hash Functions . . . 118
4.6 **Post-Quantum TLS Performance Experiment** . . . 118
4.6.1 Methodology . . . 118
4.6.2 Results . . . 119
4.7 **Conclusion** . . . 126


Transport Layer Security (TLS) is the backbone security protocol of the internet. It is arguably the most prominent cryptographic protocol in the world. Everyone on the internet is most

likely connected using TLS. At the time of writing, Cloudflare reports [Clo26a] that over 98% of internet traffic is secured using TLS.

TLS is a cryptographic protocol, which means that its security will be impacted by quantum computers. Since TLS is *the* cryptographic protocol, there are numerous works analyzing the impact of quantum computers and proposing solutions achieving post-quantum security for TLS. However, the large number of contributions makes it difficult to understand the state of the art. Indeed, the existing proposals differ in many aspects: The security properties they seek to protect (confidentiality, authenticity or both), their underlying post-quantum hardness assumptions (lattice, code, isogeny, multivariate or hash-based), their specific application scenarios (unilateral or mutual authentication, pre-shared keys, etc.), the corresponding TLS versions (1.2 or 1.3), the different setups for their experiments and the resulting benchmarks (e.g., idealized or real networks), and their type of protection (purely post-quantum or combined with classical schemes).

This confusing state of affairs raises several questions: Which approaches actually achieve the desired properties? Which proposals are efficient? Which post-quantum assumptions are trustworthy? What application scenarios are realistic? Which research problems have been solved, and which are still open or need more attention? Answering these questions is crucial for making TLS post-quantum on a large scale in practice. This chapter aims to answer these questions.

In Section 4.1, we present the TLS protocol and summarize the impact of quantum computers. The TLS protocol's quantum vulnerabilities can be put into three categories: authentication, key exchange and symmetric primitives. Many of the aforementioned post-quantum proposals focus only on one category. Consequently, we classify these proposals into these categories. Section 4.2 focuses on post-quantum *authentication*, Section 4.3 focuses on post-quantum *key exchange*, Section 4.4 focuses on proposals that combine the two former categories into a post-quantum *authenticated key exchange* and Section 4.5 discusses the quantum computer impact on the symmetric primitives used by TLS. Finally, in Section 4.6, we present performance results from our post-quantum TLS experiment, which addresses gaps in the existing literature.

**Remark**

The content of this chapter is based on our published work titled "A Comprehensive Survey on Post-Quantum TLS" [Aln+24].

## 4.1 Overview of TLS

TLS is a protocol for establishing a secure channel between two peers, called the *client* and the *server*. The security features of TLS are confidentiality, authenticity, and integrity of the transmitted messages.

Throughout the years, there have been many versions of the protocol. At the time of writing, the latest version, 1.3 [Res18], accounts for over 96% TLS-secured connections [Clo26a]. The other 4% use the previous version, 1.2 [RD08]. Because of the overwhelming adoption of TLS 1.3, we do not present details of TLS 1.2 in our work. Furthermore, in the context of migration to post-quantum TLS, users of TLS 1.2 can directly migrate to the post-quantum version of TLS

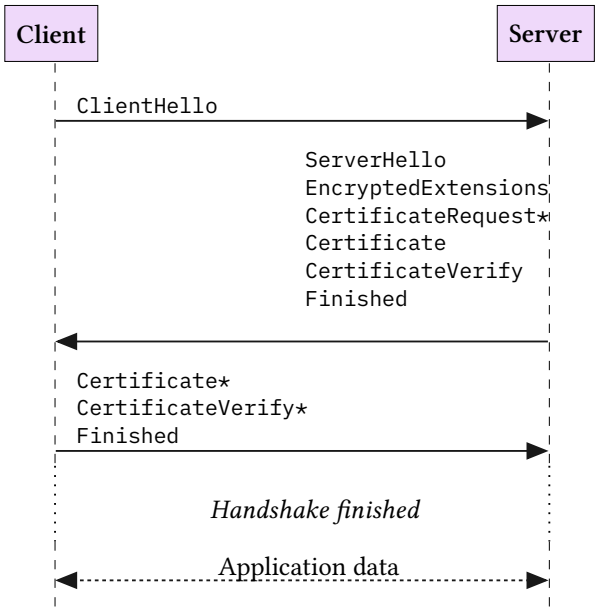



**Figure 4.1:** Typical TLS 1.3 handshake. Messages marked with * are optional depending on the authentication mode.

(which is based on TLS 1.3, as discussed in this chapter). However, some publications explored post-quantum variants of TLS 1.2 because, at the time, TLS 1.3 did not exist.

Our description and notation of TLS 1.3 follows the IETF specification [Res18]. TLS consists of two sub-protocols. In the *handshake protocol*, the peers authenticate themselves, negotiate cryptographic parameters and establish a shared secret. In the *record protocol*, the peers use the negotiated parameters and secrets to securely transmit sensitive data. We do not present the record protocol, since, from a quantum security analysis perspective, it is not the vulnerable part because it relies on symmetric cryptography—see Section 4.5 for security analysis of symmetric primitives in TLS.

### 4.1.1 Cryptographic Building Blocks

TLS employs the following cryptographic building blocks: a hash function (Definition 5), a signature (Definition 13), a message authentication code (MAC) (Figure 2.2), the Diffie-Hellman key exchange, and an authenticated encryption scheme (AEAD) (Definition 3).

### 4.1.2 Handshake Protocol

Below we describe the most common variant of the TLS 1.3 handshake, in which the peers do not have pre-shared keys (non-PSK), with a single round trip (1-RTT) and unilateral or mutual authentication using X.509 certificates (Section 3.1.3). The handshake structure is illustrated in Figure 4.1.

The client begins the handshake by sending the message `ClientHello`, which includes the cryptographic algorithms and parameters that the client supports:

- The supported TLS cipher suites, consisting of an AEAD and a hash function, are included in the `cipher_suites` field using code points. A *code point* is an identifier of a cryptographic algorithm used in TLS.
- The supported signature algorithms are listed in the `signature_algorithms` extension.
- The supported Diffie-Hellman groups for the (EC)DHE key exchange are listed in the `supported_groups` extension.
- The client generates a keypair for each DH group and includes these public keys in the `key_share` extension. A *key share* is a public key that is exchanged between participants and is used to compute a shared secret.
- Optionally, the client sends an additional list of signature algorithms in the `signature_algorithms_cert` extension, indicating supported signatures inside certificates.

The server computes the DH shared secret, which at that point is still unauthenticated; the shared secret is used to derive subsequent symmetric keys for the record protocol. The server responds with the following messages:

- `ServerHello` contains the server's choice of the parameters from `ClientHello`. The server generates its own keypair for the selected DH group and inserts the public key into the `key_share` extension.
- `EncryptedExtensions` contains extensions, which are encrypted using the derived keys from the shared secret. An example of such an extension is the `supported_groups` extension containing the server's supported DH groups.
- If the server requires that the client also authenticates itself, it sends the optional `CertificateRequest` message.
- `Certificate` contains the server's certificate chain. The server's certificate must contain a public key corresponding to one of the signature schemes listed in `signature_algorithms`. If the client sent the `signature_algorithms_cert` extension, the certificates in the chain need to be signed using one of the signature algorithms listed in the extension.
- `CertificateVerify` contains a signature over the hash of all handshake messages up to that point (which includes the public keys). The private key used to compute the signature corresponds to the public key in the server's certificate.
- `Finished` contains a MAC tag over the hash of all handshake messages up to that point. The MAC algorithm is HMAC using the hash function from the negotiated cipher suite.

Note that all *server* messages, except `ServerHello`, are encrypted with the AEAD using a derived key from the shared secret—the specifics of the encryption are handled by the record protocol.

**Remark**

At that stage of the protocol, the derived AEAD key(s) may not be secret due to a man-in-the-middle attack. If such an attack occurs, it will get discovered at the end of the protocol by verifying the MAC tags in the `Finished` messages. However, encrypting those messages is still beneficial compared to sending them in plaintext—as was the case in TLS 1.2 [RD08].

If the server sent the `CertificateRequest` message, the client responds with its `Certificate` and `CertificateVerify` messages. The final handshake message from the client is `Finished`, which includes a MAC tag over the entire handshake transcript.

After each side receives the `Finished` message and successfully verifies the tags and signatures, the handshake protocol concludes. The peers begin sending application data protected by the AEAD algorithm—the specifics are handled by the record protocol. TLS 1.3 derives the symmetric keys used by the record protocol with a key derivation function (Definition 7), which is referred to as the *key schedule* in the TLS context. The exact KDF is determined by the negotiated hash function and takes the shared DH secret as input.

### 4.1.3 Quantum Threat to TLS

We explain how a quantum adversary can break the security properties of TLS. Recall from Section 1.4.1 that quantum computers break existing (classical) public-key cryptography. TLS uses RSA, or ECDSA, as signatures and DH for key exchange—these algorithms do not provide any security in the presence of a quantum adversary. Below, we discuss the impact on confidentiality and authenticity in TLS. In Section 4.5, we discuss the quantum security of symmetric primitives in TLS.

#### 4.1.3.1 Confidentiality

A quantum adversary with access to the transcript of a TLS session can break the DH key exchange protocol and thus learn the peers' shared secret. With this information, the adversary can decrypt the data encrypted using the record protocol, which completely undermines the confidentiality of the channel.

It is important to note that future quantum attackers who have stored transcripts of past TLS sessions can break the confidentiality of those sessions retrospectively and obtain sensitive information that remains sensitive in the future. We touched upon these attacks in Section 1.4.3, and they illustrate the need for a timely transition to post-quantum key exchange in TLS.

#### 4.1.3.2 Authenticity

A quantum adversary can forge signatures in certificates that peers use to authenticate themselves in TLS sessions. In particular, a quantum adversary can impersonate the root certification authority of the underlying PKI and thus break the authenticity of any TLS communication channel established with certificates from that PKI. This is essentially one of the attacks on a PKI described in Chapter 3.

Unlike the confidentiality of today's TLS sessions, the authenticity of these channels will not be affected by future quantum adversaries because authentication is complete at the end of the

handshake phase. However, this does not mean that we can "sit back and relax", because for a variety of reasons it will take a long time to make the underlying PKIs fully quantum-secure.

## 4.2 Post-Quantum Authentication in TLS

We review and compare existing proposals for post-quantum authentication in TLS. We distinguish between purely post-quantum and hybrid post-quantum approaches: *purely post-quantum* solutions (Section 4.2.1) use only post-quantum cryptography, while *hybrid post-quantum* solutions (Section 4.2.2) combine classical and post-quantum cryptography.

Recall from Section 4.1 that authentication in TLS requires two cryptographic primitives: signatures and MACs. Since MACs are generally considered quantum-secure (see Section 4.5), only the digital signatures used in TLS need to be updated to make authentication in TLS quantum-secure. Signatures are used in TLS for two purposes: first, as "static" CA signatures of the certificate chain that the server sends to the client, and second, as "fresh" server signatures of messages in the handshake protocol.

From a conceptual perspective, the use of new signature schemes, classical or post-quantum, in TLS is not complicated because the signature scheme is treated as a black box at the higher protocol level. For example, TLS can use reserved code points to negotiate and employ additional signature schemes that are not yet officially supported. The real challenge with post-quantum authentication in TLS is finding a cryptographic solution that provides a reasonable balance between post-quantum security and performance.

### 4.2.1 Purely Post-Quantum Authentication in TLS

We describe works that replace classical signature schemes in TLS with post-quantum ones. We classify these papers based on the underlying post-quantum hardness assumptions.

In Table 4.1, we summarize the sizes of the most relevant post-quantum signature schemes and their impact on the computational performance of TLS. To this end, we simulated TLS with these signatures in a realistic and consistent environment to obtain meaningful and comparable benchmarks (see Section 4.6 for details).

#### 4.2.1.1 Hash-based Signatures

Numerous works [Bür+20; PSS21; SM16; SKD20a; KS19; WvdG20; SKD20b; DG22; Pau+22; Tas+22; SSW20] proposed the use of hash-based signatures in TLS and provided some benchmarks. Specifically, three hash-based signatures were studied: XMSS [Hue+18], Picnic[1] [Zav+20], and SLH-DSA [NIS24c].[2]

Recall from Section 2.4.2.3 that XMSS is a *stateful* signature scheme that has been standardized [Coo+20] before the NIST standardization concluded. The main advantage of XMSS over the other signature schemes below is its performance, both in terms of computation and size. However,

[1] Picnic follows the MPC-in-the-head paradigm to implement digital signatures with symmetric cryptographic primitives. We have categorized Picnic as hash-based because that is its inner core.

[2] LMS [MCF19] is another hash-based signature standardized by NIST, but we do not include it in our analysis because its use has not been studied in the context of post-quantum TLS.

its statefulness limits its applicability in practice. In fact, since stateful signature schemes require private keys to be updated each time they are used, XMSS could well be used in the real world for creating certificates, as tested in [SSW20; Pau+22], but hardly for signing messages in individual TLS sessions. Note that the limitations of XMSS apply to any stateful signature scheme in this context.

Picnic is a *stateless* signature scheme whose performance in the TLS authentication phase has been tested in [PST20; WvdG20]. These results show that Picnic's performance in terms of speed, key and signature sizes is similar to other hash-based signatures such as SLH-DSA (see below). However, during the NIST competition, it was discovered that Picnic does not meet the originally claimed security level because the underlying block cipher is flawed [Moo+20; Liu+22; LIM21], and therefore NIST discarded Picnic.

SLH-DSA, née SPHINCS+ [Hül+22], is the only *stateless* hash-based signature scheme standardized by NIST [Ala+22]. While its key sizes are small, its signature size is relatively large which limits its applicability in TLS. Specifically, while the public key size is only 32 bytes, the signature size of SLH-DSA at security level 1 is 7 856 or 17 088 bytes [Ala+22]. Additionally, the operations of SLH-DSA are orders of magnitude slower than, e.g., those of RSA or ML-DSA [SKD20a]. As a result, the TLS handshake with SLH-DSA is significantly slower than with other signature schemes: [SKD20b; SKD20a] report that the handshake with SLH-DSA takes 3–4× the time of the handshake with classical (RSA/ECDSA) or post-quantum lattice-based (ML-DSA/FN-DSA) signatures, and according to [DG22] the handshake with SLH-DSA takes 2.5–15× the time of the handshake with ML-DSA. SLH-DSA was evaluated on embedded devices in [Bür+20; Tas+22]; [Bür+20] reports that a handshake with ML-KEM and SLH-DSA takes 12–20× the time of the classical handshake with ECDHE and ECDSA; [Tas+22] reports that the handshake with SLH-DSA takes 714× the time of the handshake with ML-DSA. In our measurements, the TLS 1.3 handshake with SLH-DSA, at security level 1, took 1.58× the time of the classical handshake with Ed25519. We suspect that the wide range of reported SLH-DSA performance is due to a combination of different setups and the fact that it has 12 parameter sets (with varying performance) at *each* security level (1, 3 and 5).

#### 4.2.1.2 Lattice-based Signatures

Various works [De +20; SM16; PST20; SKD20b; DG22; SKD20a; KS19; WvdG20; MS22] studied the use of lattice-based signature schemes in TLS and tested their performance. Specifically, three signatures were studied: ML-DSA [NIS24a], FN-DSA [Pre+22] and qTesla [Bin+19a].

ML-DSA, née Dilithium [Lyu+22], has been standardized by NIST in 2024 [NIS24a]. FN-DSA, née Falcon [Pre+22], is going to be standardized by NIST in the near future [Ala+22]. Recall from Section 2.4.2.1 that the security of ML-DSA is based on the MLWE problem and that of FN-DSA on the NTRU problem. The computational efficiency of these schemes is in the same order of magnitude as classical algorithms, such as ECDSA, but their sizes are much larger. In fact, FN-DSA's keys and signatures are about 16× the size of those of ECDSA, and ML-DSA's are about 39× the size of those of ECDSA (e.g., Table 1 in [SKD20b]). At the computational level, FN-DSA signature verification time is 0.3–0.5× ML-DSA's time, but its signing time is about 10× ML-DSA's time [Pau+22; DG22]. On the other hand, FN-DSA requires double-precision floating-point arithmetic, which is not always supported; in such cases, these operations must be emulated, which slows down signing by a factor of 10 according to [Por19]. Floating-point arithmetic

**Table 4.1:** Purely Post-Quantum Signature TLS Performance

| Algorithm (sec. level) | Public key [bytes] | Signature [bytes] | TLS slowdown coefficient Our data | TLS slowdown coefficient External sources |
|---|---|---|---|---|
| FN-DSA (1) | 897 | 666 | 0.98 | 0.75–1.04 |
| ML-DSA (2) | 1 312 | 2 420 | 1.02 | 0.95–1.03 |
| SLH-DSA (1) | 32 | 7 856 | 1.58 | 3–13.8* |

Data sizes taken from [Ala+22] (for SLH-DSA we take the small parameter choice, as defined in [Hül+22]). External sources: [SKD20b; DG22; KS19] (FN-DSA, ML-DSA and SLH-DSA), [SKD20a] (SLH-DSA). "TLS slowdown coefficient" is defined as $\frac{t_{pq}}{t_c}$, where $t_{pq}$ is the TLS handshake time with the post-quantum scheme and $t_c$ is the time with a comparable classical scheme (RSA/ECDSA). We provide ranges for the external sources since their setups are not completely consistent. *We attribute the wide range to SLH-DSA having multiple security level 1 parameter sets which differ in performance. Setup for our data (see Section 4.6 for more details): comparison with Ed25519, certificate chain length 2, and key exchange x25519; RTT 7ms, bandwidth 1.08Gb/s; SLH-DSA parameter set: SHA2, simple, small.

also introduces side-channel vulnerabilities making FN-DSA insecure on some systems [Wes22; HW23]. As [KS19; Pau+22] have shown, ML-DSA and FN-DSA offer similar performance in TLS, except in cases where FN-DSA's slower signing time creates a bottleneck, e.g., when a server (doing the signing) has to serve multiple clients. Our measurements show that FN-DSA's advantage over ML-DSA grows with increasing security level: a TLS 1.3 handshake with ML-DSA takes 1.04× the time of the handshake with FN-DSA at security levels 1 and 2, while the handshake with ML-DSA takes 1.1× the time of the handshake with FN-DSA at security level 5.

The security of qTesla [Bin+19a] is based on RLWE (Section 2.4.2.1). qTesla was submitted to the NIST competition but did not advance from the second to the third round because its security and performance balance was not convincing [Moo+20] and its closely related competitor, ML-DSA, is based on a weaker hardness assumption, MLWE, and offers better performance. In fact, [PST20; WvdG20] have shown that qTesla typically performs worse than ML-DSA in TLS handshake scenarios.

#### 4.2.1.3 Multivariate-based Signatures

The use of multivariate signature schemes in the TLS handshake, such as GeMSS [Cas+20], Rainbow [Din+20] and MQDSS [Sam+19], has been studied in some works [SKD20b; DG22; SSW20]. These schemes offer comparable or even better performance than hash-based or lattice-based ones, but all of them have been broken subsequently [Beu22; TPD21; KZ20]. Since those schemes are broken, we do not provide details on how the TLS handshake performs with these schemes in this paper, but only refer to those works that studied their deployment.

### 4.2.2 Hybrid Post-Quantum Authentication

Hybrid post-quantum authentication uses a classical and a post-quantum signature scheme in combination, thus combining classical and post-quantum security. We have identified two approaches: the *concatenation* approach (Section 4.2.2.1) and the *separation* approach (Section 4.2.2.2).

#### 4.2.2.1 Concatenation

This approach uses the parallel composite signatures presented in Definition 24 in combination with a composite hybrid post-quantum PKI (Section 3.3.2).

Since the TLS standard does not support the use of multiple signatures in parallel, the Open Quantum Safe (OQS) project [SM16; CPS19] extended the OpenSSL library as follows: classical and post-quantum keys and signatures are concatenated and regarded as regular public keys and signatures, respectively, of a single signature scheme. This hybrid signature scheme is then used in TLS as usual. The approach is implemented for TLS 1.3 and currently supports the combination of classical algorithms RSA/ECDSA with ML-DSA/FN-DSA/SLH-DSA.

Two works [WvdG20; MS22] tested the OQS implementation in several configurations. From their results, we can conclude that hybrid post-quantum authentication takes about 1.2–2× the time of purely post-quantum authentication. The specific factor depends, of course, on the performance ratio between the classical and post-quantum signatures used. For example, according to [WvdG20], a purely post-quantum TLS handshake using ML-DSA (purely post-quantum authentication) and ML-KEM [NIS24b], née Kyber [Sch+22b], (purely post-quantum key exchange, Section 4.3.1) takes 18 ms, while a hybrid TLS handshake using RSA + ML-DSA (hybrid post-quantum authentication) and ECDHE + ML-KEM (hybrid post-quantum key exchange, Section 4.3.2) takes 30 ms. Another result, using FrodoKEM instead of ML-KEM and Picnic instead of ML-DSA, shows that the purely post-quantum handshake takes 64 ms and the hybrid handshake takes 75 ms. The main reason the factor is about 1.7 in the first example but only 1.2 in the second is that the post-quantum primitives in the first example are more efficient than those in the second. We conducted an experiment to independently study the performance of solutions for hybrid post-quantum authentication.

Our benchmarks (Tables 4.3 and 4.4 in Section 4.6) confirm prior results. In particular, for hybrids with security level $> 2$, classical signatures become a bottleneck because the performance gap between ML-DSA and FN-DSA at different security levels is not as significant as for ECDSA with P-256 and P-521.

#### 4.2.2.2 Separation

As an alternative to the concatenation approach presented in Section 4.2.2.1, the authors of the draft RFC [She+25] propose transporting the classical and post-quantum signatures separately. Specifically, the authors propose that clients indicate their preference for this authentication method in `ClientHello`, and in return, the server sends them 1) a modified `Certificate` TLS message containing separate classical and post-quantum certificates, and 2) a new `DualCertificateVerify` TLS message—containing two (separate) signatures over the transcript, a classical one and a post-quantum one—in place of the standard `CertificateVerify` TLS message.

Note that this method follows a similar design strategy as the separation approach used in key exchange presented in Section 4.3.2.2. To the best of our knowledge, this approach has not been implemented yet or benchmarked.

### 4.2.3 Merkle Tree Certificates

The approaches presented in Sections 4.2.1 and 4.2.2 implicitly assume the existence of post-quantum certificates and consequently a post-quantum PKI. Recall from Chapter 3 that post-quantum certificates essentially only differ from classical certificates in what public keys and what signatures they contain; otherwise, the PKI paradigm remains unchanged. The draft RFC [Ben+26] proposing *Merkle Tree Certificates* (MTC) introduces a more drastic change to the PKI ecosystem, specifically to the Web PKI, i.e., the PKI used on the web by regular users. In this section, we briefly introduce this novel concept, which is planned to be used to deploy post-quantum authentication in TLS [Goo26b].

A *Merkle tree certificate* is effectively a standard X.509 public-key certificate except that it does not contain a standard signature over its contents made by the CA, but instead contains an "inclusion proof." The inclusion proof proves that the certificate is part of a specific Merkle tree. In a traditional Merkle tree fashion, the proof consists of a subset of nodes in the tree so that the tree root may be recomputed. The tree root is signed by a CA, which is referred to as the *Merkle Tree Certification Authority* (MTCA).

The benefit of MTC-based Web PKI compared to the traditional X.509 Web PKI is that it optimizes the TLS handshake size. The average TLS handshake authentication "costs" two public keys and five signatures: the public keys of the server and the CA; one signature over the handshake; two signatures over the server and CA certificates; and two signatures over certificate transparency timestamps. Note that we have not counted revocation-related costs because revocation checking is being superseded by short-lived certificates; see Section 3.1.4.3. The MTC-based TLS handshake requires only one public key, one signature and one inclusion proof: the server's public key; the signature over the handshake; and the inclusion proof.

This optimization is possible because of the paradigm changes introduced by the MTC standard. The (MT)CA issues certificates in batches: the "to-be-signed" certificates in a batch are used to build a Merkle tree. The tree root is then signed by the CA and also co-signed by third parties in the spirit of certificate transparency, to prevent rogue CAs. The final certificate is augmented with the inclusion proof, which may or may not include the signature of the tree hash; if the signature is present, the certificate is referred to as "standalone," and if not, then it is referred to as "landmark."

The signed Merkle tree roots are assumed to be periodically distributed out of band to clients, which are saved locally by the client until they expire. The verification public keys are also assumed to be pre-shared, analogous to pre-sharing root certificates. In this paradigm, each CA effectively runs its own certificate transparency log, which is also used to validate certificates that are audited by other Web PKI participants. Note that our description is heavily simplified and the actual standard is more complex.

Overall, MTCs are clearly a superior approach in terms of performance in the context of Web PKI. However, the whole paradigm shift introduces significant costs to existing CAs, which may lead to further centralization in the Web PKI ecosystem. Also, there is nothing preventing the use of MTCs outside the Web PKI ecosystem; however, for other PKIs that do not require certificate transparency, the performance benefit may not justify the complexity costs.

## 4.3 Post-Quantum Key Exchange in TLS

We describe and compare existing proposals for post-quantum key exchange in TLS. Again, as with post-quantum authentication (Section 4.2), we distinguish between purely post-quantum and hybrid post-quantum approaches. Recall that *purely post-quantum* solutions (Section 4.3.1) use only post-quantum cryptography, while *hybrid post-quantum* ones (Section 4.3.2) combine classical and post-quantum cryptography.

There is only one general purely post-quantum key exchange approach, while there are several approaches for hybrid post-quantum key exchange.

### 4.3.1 Purely Post-Quantum Key Exchange in TLS

All works in the literature that studied replacing the classical DH key exchange with post-quantum alternatives are based on KEMs. This approach was studied for TLS 1.2 in [Bos+15; Bos+16; SM16; CPS19; De +20; Bür+20; PSS21] and for TLS 1.3 in [SM16; CPS19; SKD20a; WvdG20; Why+17; Pau+22; SSW20; DG22; Ber+22; Tas+22; MS22].

The idea of using KEMs for ephemeral key exchange in TLS works as follows (Figure 4.2b):

1) The client creates an ephemeral KEM keypair and sends the public key to the server.
2) The server runs the encapsulation mechanism with the public key of the client as the input and obtains the ciphertext and the shared secret. The server sends the ciphertext to the client.
3) The client decapsulates the ciphertext with its private key and gets the shared secret.

Comparing this process with the original DH flow (Figure 4.2a) shows that it is possible to use KEMs in TLS instead of DH key exchange without changing the overall protocol flow.[3] Thus, the main challenge for purely post-quantum key exchange in TLS is to find a cryptographic solution with a reasonable balance between security and performance.

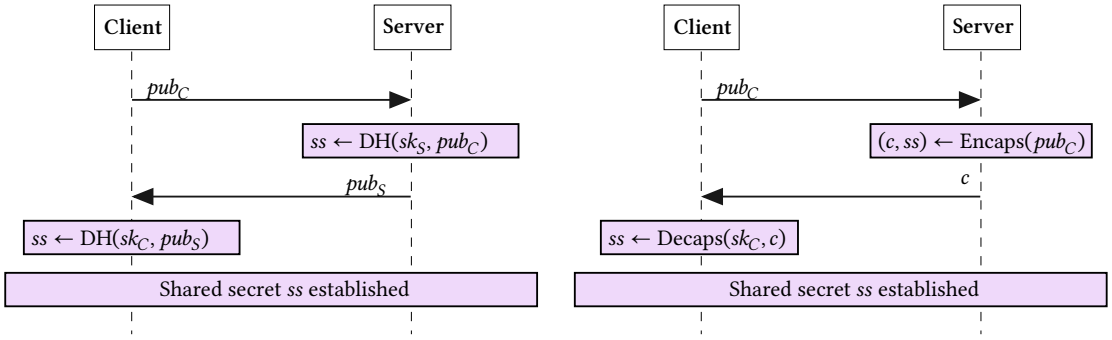
In Table 4.2, we summarize the sizes of the most relevant post-quantum KEMs and their impact on the computational performance of TLS. As with the purely post-quantum authentication, we simulated TLS with these KEMs in a realistic and consistent environment to obtain meaningful and comparable benchmarks (see Section 4.6 for details).

As in Section 4.2.1, we classify existing work on purely post-quantum key exchange based on the underlying hardness assumptions of the key exchange schemes.

#### 4.3.1.1 Lattice-based KEMs

We identified three different groups of lattice-based KEMs (Section 2.4.2.1) which we characterize by their underlying hardness assumptions: variants of Learning-with-Errors (LWE) [Bos+16; Bos+15; Pöp+19; Sch+22b], Learning-with-Rounding (LWR) [DAn+20], and variants of NTRU [Ber+20; Che+20].

[3] This approach is not entirely new. RSA-based key exchange, which can be considered a KEM, is supported by TLS 1.2 [RD08].




(a) DH Key Exchange (b) KEM Key Exchange

**Figure 4.2:** Key Exchange Diagrams

**Table 4.2:** Purely Post-Quantum KEM TLS Performance

| **Algorithm** (sec. level) | **Public key** [bytes] | **Ciphertext** [bytes] | **TLS slowdown coefficient** Our data | External sources |
|---|---|---|---|---|
| Saber (1) | 672 | 736 | 1.07 | 0.89 |
| ML-KEM (1) | 800 | 768 | 1.06 | 0.86–1.02 |
| NTRU (1) | 930 | 930 | 1.13 | 1.57–1.64* |
| NTRU Prime (2) | 1 158 | 1 039 | 1.53 | 0.95 |
| BIKE (1) | 1 540 | 1 572 | 2.55 | N/A |
| HQC (1) | 2 249 | 4 481 | 1.23 | N/A |
| FrodoKEM (1) | 9 616 | 9 720 | 2.53 | 1.14–1.29* |

Data sizes taken from [Ala+22] (for NTRU Prime we take the streamlined parameter choice at security level 2, as defined in [Ber+20], because this parameter was used in the external source [Ber+22]). External sources: [DG22] (Saber, ML-KEM), [SM23b] (ML-KEM), [Bos+16] (NTRU, FrodoKEM), [Ber+22] (NTRU Prime). "TLS slowdown coefficient" is defined as $\frac{t_{pq}}{t_c}$, where $t_{pq}$ is the TLS handshake time with the post-quantum scheme and $t_c$ is the time with ECDHE. We provide ranges for the external sources since their setups are not completely consistent. "N/A" is used where we did not find external sources from which we could compute the TLS slowdown coefficient. * indicates that tested KEMs were only IND-CPA secure. Setup for our data (see Section 4.6 for more details): comparison with x25519, certificate chain length 2, and signature ECDSA (P-256); RTT 7ms, bandwidth 1.08Gb/s; FrodoKEM: AES, NTRU Prime: Streamlined.

Among the LWE KEMs, we distinguish between the following three categories, depending on which variant of LWE their security is based on. The security of Frodo [Bos+16] reduces to the "conservative" plain LWE assumption, that of ML-KEM [NIS24b], née Kyber [Sch+22b], to the MLWE assumption (Definition 16), and that of NewHope [Pöp+19] (and its predecessor BCNS [Bos+15]) to the RLWE assumption. All of these KEMs were submitted to the NIST standardization process.[4] NIST selected ML-KEM as a winner, while FrodoKEM and NewHope did not advance to subsequent rounds because ML-KEM's performance is significantly better than FrodoKEM's (see below), and because ML-KEM offers a similar performance to NewHope but requires a weaker hardness assumption for this (see [Ala+22; Moo+20] for details about NIST's selection).

The TLS performance of these LWE-based KEMs was extensively studied in the literature (see, e.g., [PSS21; Bür+20; SKD20a; WvdG20; Pau+22; DG22; SSW20; Tas+22; MS22]) and their characteristics can be summarized as follows.

Overall, the performance of TLS with ML-KEM is similar to that of TLS with classical key exchange protocols such as ECDHE, and in some configurations even better. According to [SKD20a], the TLS 1.3 handshake with ML-KEM and ML-DSA, at security level 1, takes roughly the same time as a classical handshake with ECDHE and RSA while at security level 3 it takes 2× the time (compared to the same classical scheme). Another work [DG22] finds that ML-KEM, at NIST security level 1, has almost identical performance to ECDHE with Curve25519 in TLS 1.3, and is even faster at security levels 3 (ECDHE with Curve448) and 5 (ECDHE with curve P-521) by a factor of 2 and 7 respectively. NewHope's performance is similar to that of ML-KEM [SKD20a], but it requires a stronger hardness assumption (RLWE vs. MLWE) and therefore ML-KEM can be considered superior to NewHope. The TLS handshake with FrodoKEM takes 2.7× the time a handshake with ML-KEM or NewHope takes according to [WvdG20] and its public key and ciphertext sizes are about 10× the sizes of ML-KEM or NewHope [Ala+22]. The original FrodoKEM paper [Bos+16] reports that the TLS 1.2 handshake with FrodoKEM takes 1.14–1.29× the time of a classical handshake with ECDHE, but we note that this paper tested only the IND-CPA version (Figure 2.5). We measured the latest FrodoKEM IND-CCA version, and we observed that the TLS 1.3 handshake with FrodoKEM at security level 1 takes 2.53× the time of the classical handshake with ECDHE (Curve25519). While these numbers show that ML-KEM's performance is significantly better than FrodoKEM's, they demonstrate that the more conservative FrodoKEM can still be a practical choice for use in TLS. Indeed, FrodoKEM is still recommended by some authorities (e.g., BSI [BSI26]) and is currently being standardized by ISO [Fro25].

The security of Saber [DAn+20] is based on the Module Learning With Rounding (MLWR) assumption, which can be seen as a modification of the MLWE assumption (Definition 15). The use of Saber in TLS 1.3 was studied in [WvdG20; DG22; MS22], all of which report that the key exchange phase in TLS with Saber is at least as efficient as with ML-KEM, both in terms of time and size. We also measured Saber's performance with the same outcome (Section 4.6). Saber was also submitted to the NIST competition but, unlike ML-KEM, Saber did not advance to the 4th round of the NIST competition because its hardness assumption MLWR had not yet been researched as much as ML-KEM's MLWE assumption [Ala+22].

[4] The submitted versions of Frodo, called FrodoKEM [Nae+20], and of NewHope [Pöp+19] differ from their original specifications [Alk+16; Bos+16] in that they provide not only IND-CPA but even IND-CCA security (Figure 2.5), as required by NIST.

There are two KEMs whose security reduces to the NTRU problem [HPS98]. The first KEM is based on a modification of NTRU, called NTRU Prime [Ber+20]. Its hardness assumption is intended to be more conservative than the original NTRU problem and MLWE/MLWR, with only slightly larger key shares. In [Ber+22], the NTRU Prime KEM was integrated into TLS 1.3, essentially as described above, but with some performance optimizations which ensured that the resulting TLS 1.3 handshake with NTRU Prime takes 0.95× the time of the classical handshake with ECDHE (Curve25519). Our measurements show that the TLS 1.3 handshake with NTRU Prime's LPR variant at security level 1 takes 1.11× the time of the handshake with ECDHE (Curve25519) while the Streamlined variant at security level 1 takes 1.45× the time.

The security of the second KEM reduces to the classic NTRU problem and is therefore commonly referred to as NTRU [Che+20]. The key shares of NTRU and NTRU Prime have essentially the same size. According to [SKD20a], the TLS 1.3 handshake with NTRU takes similar time as with ML-KEM (at security level 3) and takes about 2× the time of a classical ECDHE handshake; however, this result is inconsistent with the claim in the same paper that NTRU's algorithms take up to 100× the time of ML-KEM algorithms while having similar sizes. The performance of NTRU and ML-KEM in TLS has also been studied and compared in [WvdG20] which reports that the handshake with NTRU takes about 3.25× the time of the handshake with ML-KEM while being comparable to FrodoKEM. According to [MS22] the TLS 1.3 handshake with NTRU takes 1.05–1.07× the time of a handshake with ML-KEM or Saber at multiple security levels which corresponds to our measurements presented in Section 4.6.

#### 4.3.1.2 Code-based KEMs

HQC [Agu+22] and BIKE [Ara+22] are the only code-based KEMs tested in TLS [WvdG20; Tas+22; DG22].[5] HQC and BIKE both advanced into the fourth round of NIST's standardization, and in 2025 NIST selected HQC for standardization [Ala+25a].

BIKE's public key and ciphertext sizes are about 0.5× the sizes of HQC but its computations (key generation, encapsulation, decapsulation) take approximately 4.3× the time of HQC's computations [SM23a].[6] According to [Tas+22], the TLS 1.3 handshake with BIKE takes 4.3× the time of the handshake with ML-KEM, and the handshake with HQC takes about 1.8× the time of the handshake with ML-KEM, whose performance is similar to that of ECDHE. Our measurements show that the TLS 1.3 handshake with HQC at security level 1 takes 1.45× the time of the classical handshake with ECDHE (Curve25519) while the handshake with BIKE at security level 1 takes 2.55× the time of the classical handshake.

#### 4.3.1.3 Isogeny-based KEMs

SIKE [Jao+22] is the only isogeny-based KEM that was tested in TLS, in fact in [WvdG20; DG22; SSW20; Tas+22]. The advantage of SIKE over code-based or lattice-based schemes is its key and ciphertext sizes, which are at most 0.5× the sizes of lattice-based KEMs, such as ML-KEM [Sch+22b]

[5] A reader may wonder whether Classic McEliece [Alb+22] has also been integrated into TLS. We have not found such a work, and we suspect that this is because TLS does not support public keys larger than $2^{16} - 1$ bytes [Res18], but the smallest public key size for Classic McEliece is 4× this limit (see Table 2.1).

[6] Computed as sum of cycles for each operation at security level 1, x86 architecture, code type - performance mode, 2023-06-24.

or Saber [DAn+20]. On the downside, a TLS 1.3 handshake with SIKE takes around 15× the time of the handshake with ML-KEM or Saber according to [WvdG20], and even 31× according to [DG22]. The main problem with SIKE, however, is that it has been shown to be insecure [CD23].

### 4.3.2 Hybrid Post-Quantum Key Exchange in TLS

We identified three different approaches to hybrid key exchange in TLS. The first two run one classical and one post-quantum key exchange algorithm in parallel, and they differ in how they integrate the additional post-quantum key share into the overall protocol flow (Sections 4.3.2.1 and 4.3.2.2). The third approach differs from the other two in that it uses one classical and $n \geq 1$ post-quantum algorithms in parallel (Section 4.3.2.3). All of these approaches are essentially variants of the hybrid post-quantum composite KEM (see Definition 27).

#### 4.3.2.1 Concatenation

The main idea is to concatenate the classical and post-quantum key material (public keys/ciphertexts) and treat them as single elements in TLS. This is conceptually similar to the hybrid authentication as described in Section 4.2.2.

More specifically, in the negotiation phase, one classical and one post-quantum key exchange protocol are negotiated. Then, in the key exchange phase, the classical and post-quantum key exchange algorithms are run in parallel on a concatenated input, the classical and post-quantum parts of which are first decoupled and then processed individually by the respective algorithms. The outputs of the two algorithms are concatenated to form the shared secret, which is then processed as in the usual TLS protocol. However, that one of the key shares must be of known constant length to be able to decouple them; this requirement can be avoided, for example, by defining an encoding of the key shares.

The concatenation approach was first proposed for TLS 1.2 in [Bos+15] and later in [Bos+16]. The authors of [Bos+15] instantiated this approach with ECDHE and their RLWE-based key exchange algorithm BCNS (Section 4.3.1). Their evaluation results show that the TLS handshake using this hybrid key exchange takes about 1.7× the time of the handshake using ECDHE and about 1.4× the time of the handshake using the purely post-quantum key exchange with BCNS. The results of [Bos+16] show that the handshake using a hybrid key exchange (ECDHE with NewHope [Alk+16]) takes about 1.2× the time of the handshake with ECDHE and about 1.4× the time of the handshake using the purely post-quantum key exchange NewHope.

Later, the concatenation approach was also proposed for the latest version 1.3 of TLS in [SFG25; KK18]. The OpenSSL fork from the Open Quantum Safe (OQS) project implements the concatenation approach [SM16; CPS19] for hybrid key exchange in TLS 1.3. The efficiency of this implementation in various settings was analyzed in [PST20; WvdG20; SSW20]; we summarize their main insights next. First, [PST20] reports that the performance of a hybrid key exchange with ECDHE and ML-KEM in TLS 1.3 is essentially identical to a classical key exchange with ECDHE under *ideal network conditions*. In environments with a high packet loss, the additional bandwidth requirements of post-quantum algorithms make the hybrid key exchange protocol significantly less efficient than the classical one. Another observation made in [PST20] is that SIKE [Jao+22], with key shares 0.4× the size of ML-KEM, performs significantly worse than ML-KEM under ideal network conditions due to its much slower computations, but in environments with high packet

loss, SIKE outperforms ML-KEM, since SIKE's slow computations get negligible compared to ML-KEM's higher bandwidth requirements, which cause a higher packet retransmission rate.

We conducted an experiment to study the performance of solutions for hybrid post-quantum key exchange; see Table 4.6 and Figure 4.6 in Section 4.6 for our detailed results. In particular, we observed that a hybrid post-quantum handshake with a combination of ECDHE (Curve25519) and ML-KEM at security level 1 took 1.25× the time of a classical handshake with ECDHE (Curve25519). Interestingly, the more secure combination of ECDHE (Curve25519) with ML-KEM at security level 3 took 1.28× the time of a classical handshake with ECDHE (Curve25519) and is therefore similarly fast; this combination is currently used in Google Chrome (see below).

Google conducted two hybrid post-quantum TLS experiments [Lan16; Lan18] that also implemented the concatenation approach. In the first experiment [Lan16], the authors tested a hybrid key exchange in TLS 1.2 with NewHope [Alk+16] and ECDHE, and they concluded that the tested hybrid post-quantum solution can be considered practical. The second experiment [Lan18] was conducted in collaboration with Cloudflare [KV19]. It tested two hybrid key exchange configurations in TLS 1.3 with NTRU [Che+20] and ECDHE, and with SIKE [Jao+22] and ECDHE. The overall result is consistent with the results of [PST20] (see above), as the performance of the pair NTRU/ECDHE is similar to that of the classical ECDHE, while the performance of the pair SIKE/ECDHE is worse than NTRU/ECDHE under ideal network conditions, but better under poorer network conditions such as in mobile communications.

In 2022, Google announced that it had implemented this hybrid approach (using ECDHE with NTRU) in its modified TLS protocol called ALTS [Goo22]. Since August 2023, Google Chrome 116+ [Goo23b] supports post-quantum hybrid key exchange by a combination of ECDHE (Curve25519) with ML-KEM-768; see [WS23] for implementation details.

A recent development in this context is KEM combiners, which derive a hybrid secret from composite secrets (see Section 2.4.5.2). For example, X-Wing [Bar+24] is a unified KEM combiner for x25519 and ML-KEM-768. The deployment of these tools to simplify the standardization and migration process has recently been discussed on the IETF mailing list [Wes24].

#### 4.3.2.2 Separation

An alternative solution to concatenating key shares by using existing TLS data structures (Section 4.3.2.1) is to modify messages or to create extensions in order to transport additional post-quantum key shares in the hybrid key exchange. In this way, the classical and post-quantum key shares are separated more clearly in the TLS data structures.

The authors of [CC21] apply this idea to TLS 1.2 by extending `ClientHello` with a list of supported post-quantum KEMs, ordered by preference. In order to transport both the classical and the post-quantum keys, they modify `ServerKeyExchange` and `ClientKeyExchange` so that the post-quantum key share is embedded in a newly defined field, while the classical key share is embedded as in the TLS 1.2 standard. As in the previous approach (Section 4.3.2.1), the final shared secret is the concatenation of the resulting classical and post-quantum secrets.

The same approach has been proposed for TLS 1.3 in [SS17] but with some technical differences. First, the authors define new *group* code points for the post-quantum key exchange algorithms, so that the hybrid pair of key exchange algorithms can be negotiated as in standard TLS 1.3. Second, the authors define a new extension `additional_key_share` of `ClientHello` and `ServerHello`. While the classic key shares are transported as in the TLS 1.3 standard, the post-quantum ones

are sent using the new extension. Finally, the authors modified the key schedule (i.e., the key derivation algorithm) in TLS 1.3 to take as input the classical and the post-quantum shares.

AWS conducted an experiment using this approach in TLS 1.2 [HC19]. Specifically, they tested ECDHE combined with BIKE [Ara+22] and SIKE [Jao+22]. Their results show that BIKE and SIKE provide different tradeoffs: BIKE consumes more bandwidth while SIKE has slower computations.

#### 4.3.2.3 Multiple hybrid

The idea of this approach, which we call the *multiple hybrid*, is to use more than two key exchange algorithms in parallel to further reduce the attack surface.

In [SWZ16], this approach is applied to TLS 1.2. First, the authors extend `ClientHello` and `ServerHello` to negotiate the combination of multiple (post-quantum) key exchange algorithms and to transport all public keys. Second, they modify `ServerKeyExchange` and `ClientKeyExchange` to transport the additional ciphertexts in new dedicated fields. The classical key exchange phase remains unchanged. Finally, the shared secret is the concatenation of the classical secret and the additional (post-quantum) secrets.

A similar approach was proposed in [Why+17] for TLS 1.3. The proposal modifies the standard `key_share` extension to accommodate hybrid key sharing, and also defines a new TLS extension `hybrid_extension` for `ClientHello`, which serves as a mapping between code points in `supported_groups` and the set of key exchange algorithms used in the hybrid key exchange. For example, if the client wants to use a hybrid key exchange with ECDHE, ML-KEM and NTRU, the client selects an unallocated code point in `supported_groups` and includes the description of the code point into `hybrid_extension`. The public keys are transported in the `key_share` extension, except that the `KeyShareEntry` corresponding to the hybrid code point contains a list of three `KeyShareEntry` values representing the public keys. The authors of [Why+17] note that their modification is interoperable with standard TLS 1.3, since the modified `KeyShareEntry` is treated identically to a `KeyShareEntry` of an unsupported algorithm in TLS 1.3.

We have not found any publications that analyze the performance of this approach. While, for one classical and one post-quantum algorithm, we assume that the performance is the same as in the concatenation approach, which has been tested extensively (Section 4.3.2.1), it is not obvious how this approach scales with more than one post-quantum key exchange algorithm in realistic settings.

## 4.4 Post-Quantum Authenticated Key Exchange in TLS

We describe and discuss proposals that simultaneously guarantee authenticity and confidentiality of the TLS handshake, unlike the proposals in the previous two sections, which provide exactly one of these two properties.

### 4.4.1 KEMTLS

KEMTLS [SSW20] uses post-quantum KEMs to provide post-quantum authenticated key exchange (AKE) in TLS 1.3 without signing messages.[7] The main idea of KEMTLS is based on the obser-

[7] KEMTLS still uses digital signatures for certificates.

vation that signing messages between the client and server is sufficient but not necessary for authentication. In fact, the handshake can be more efficient using a KEM instead of signatures. This is done by using a KEM to prove to the client that the server knows a secret key corresponding to its public "verification" key.

While KEMTLS is conceptually similar to the "TLS 1.3 candidate" OPTLS [KW16], which uses DH for AKE without signatures, KEMTLS is the first work that realizes this idea with post-quantum KEMs.

In the usual TLS 1.3 setup with unilateral authentication, the server has a static signature keypair and a certificate binding the signature public key to the server. In the corresponding KEMTLS setup, the server has a static KEM keypair and a certificate binding the static KEM public key to the server.

#### 4.4.1.1 Description

We now describe the KEMTLS handshake protocol. Since KEMTLS is a modification of the standard TLS 1.3 protocol (Section 4.1), we only present the modifications.

1) Client → Server: The client sends `ClientHello` with a selection of KEM algorithms and the corresponding ephemeral KEM public keys.

2) Server → Client: The server selects a KEM and runs encapsulation using the client's public key. The resulting KEM ciphertext is the server's key share in `ServerHello` and the shared secret is fed into the key schedule. The server's `Certificate` message contains a certificate with its static KEM public key. `CertificateVerify` is never sent. `Finished` is delayed and sent in the next message flow.

3) Client → Server: The client decapsulates the KEM ciphertext from `ServerHello` and the resulting shared secret is fed into the key schedule. The client runs encapsulation using the server's static public key. The client sends a new KEMTLS message, `ClientKemCiphertext`, containing the resulting KEM ciphertext. The resulting (second) shared secret is also fed into the key schedule. The client sends `Finished`.

4) Server → Client: The server decapsulates the KEM ciphertext from `ClientKemCiphertext` and inputs the shared secret into the key schedule. The server sends its `Finished` message. Note that this message flow is extra compared to TLS 1.3.[8]

After the client receives the server's `Finished` message, the handshake is concluded. Note that compared to TLS 1.3, the `Finished` messages are computed using HMAC keys which are derived from the initial ephemeral key exchange and the static key exchange.

The modifications can be summarized as follows: the ephemeral key exchange uses KEMs instead of DH, the client sends `ClientKemCiphertext` instead of the server sending `CertificateVerify`, the server delays its `Finished` message and the TLS key schedule includes an extra round.

As mentioned above, the main concept of KEMTLS is to use KEMs instead of signatures for AKE, whereby KEMTLS trades the cost of creating and verifying one signature for the cost of encapsulating and decapsulating one message (plus a practically negligible key scheduling round).

[8] The client can still send its application data together with the `Finished` message.

#### 4.4.1.2 Performance

The main advantage of KEMTLS is its smaller communication size, due to the fact that post-quantum KEMs typically have smaller bandwidth requirements than the post-quantum signatures used in the protocols presented in Section 4.2.

For example, [SSW20] reports that a KEMTLS handshake with ML-KEM as KEM and ML-DSA as the signature scheme for certificates is about 0.85× the size of the TLS 1.3 handshake using post-quantum authentication with ML-DSA (as in Section 4.2.1) and post-quantum key exchange with ML-KEM (as in Section 4.3.1). In another example, [SSW20] states that the size of the KEMTLS handshake is about 0.71× the size of the corresponding post-quantum TLS handshake when SIKE [Jao+22] is used as the KEM and FN-DSA and XMSS are used to create certificates.

Overall, the KEMTLS handshake can therefore be faster in certain settings, as reported in [Cel+21; SSW20]. While the performance of KEMTLS is essentially the same as the separate post-quantum approach for the ML-KEM/ML-DSA pair, the KEMTLS handshake takes 0.98× the time of the corresponding TLS 1.3 handshake when used with the NTRU/FN-DSA pair. On the other hand, when KEMTLS is used with a computationally expensive KEM such as SIKE, it can take up to 1.36× the time of the corresponding separate post-quantum approach. The performance of KEMTLS was also studied on embedded devices in [GW22], which reports that "KEMTLS can reduce handshake time by up to 38%, can lower peak memory consumption and can save traffic volume compared to TLS 1.3."

#### 4.4.1.3 Extensions

While the original KEMTLS paper [SSW20] focuses on unilateral authentication, it also describes in its appendix how to achieve mutual authentication by adding an extra round trip.

In [SSW21], a modification of KEMTLS with *pre-distributed keys* is proposed, called KEMTLS-PDK. Pre-distributed keys are similar to pre-shared keys in TLS 1.3, except that pre-shared keys are symmetric keys, while pre-distributed keys are public keys. This method significantly reduces handshake size and can perform marginally better than purely post-quantum TLS (Section 4.2.1 and 4.3.1) when certificates are cached. Furthermore, it allows using post-quantum KEMs with large public keys in KEMTLS, such as Classic McEliece [Alb+22] (for authentication) combined with ML-KEM (for ephemeral key exchange), thus providing hybrid confidentiality: even if ML-KEM was broken, confidentiality would be preserved as long as Classic McEliece is safe.

While the original KEMTLS paper [SSW20] only mentions the possibility of using hybrid KEMs to achieve hybrid AKE, this option was implemented in [Gir+22], which demonstrated that the overhead of using hybrids is small (about 8%) and that hybrid KEMTLS is therefore a realistic option.

#### 4.4.1.4 Security

We note that, as pointed out in [SSW20], the KEMs in KEMTLS need to be IND-CCA secure. In addition to its original pen-and-paper proof, the KEMTLS(-PDK) protocol was formally verified using Tamarin [Cel+22] and SAPIC+ [Che+22].

#### 4.4.1.5 Related Work

A recent paper [Cam+24] proposes an alternative instantiation of OPTLS [KW16] using CSIDH [Cas+18] to provide post-quantum authenticated key exchange. Additionally, the paper proposes new parameters for CSIDH taking into account papers which called CSIDH's security into question, e.g. [Pei20]. Although conceptually similar to KEMTLS, the performance of this proposal is much worse than KEMTLS: in particular, the handshake of this protocol takes *at least* 117× the time of KEMTLS (using ML-KEM).

### 4.4.2 Mutually Authenticated Key Exchange

While KEMTLS was primarily designed for unilateral authentication in TLS, where the server authenticates itself to the client, other works proposed post-quantum solutions for mutually authenticated key exchange in TLS, where the server and the client authenticate themselves to each other. We identified three approaches in this group: authenticated key exchange (AKE), password-authenticated key exchange (PAKE), and identity-based encryption (IBE). The security of all presented proposals reduces to the RLWE problem.

#### 4.4.2.1 Authenticated Key Exchange

The authors of [Gao+17] proposed a modification of the lattice-based AKE of [Zha+15] for use in TLS 1.2. They define a cipher suite for this primitive, but they do not specify exactly how this cipher suite is integrated into TLS 1.2; for example, they do not define which TLS messages transport the public keys. In short, [Gao+17] focuses on the cryptographic aspects rather than on the protocol itself.

Since the underlying AKE protocol [Zha+15] is designed for mutual authentication, employing it in TLS requires static keys for both parties, which need to either be pre-shared or part of a certificate. The distribution of these static keys is not discussed in [Gao+17].

The authors of [Gao+17] compare the performance of their AKE in TLS 1.2 with two classical cipher suites (FFDH with RSA and ECDH with RSA) and with the RLWE key exchange protocol [Bos+15] (Section 4.3.1) in combination with RSA and ECDSA for authentication. The authors report that their AKE with a parameter set supposedly providing 80-bit security is faster than the 1024-bit FFDH with 2048-bit RSA, and their AKE with a 256-bit parameter set is slower than ECDH with a 256-bit parameter set and 2048-bit RSA. They also compared their AKE with [Bos+15] (combined with RSA/ECDSA) and observed that the TLS handshake with their AKE takes 1.12–1.18× the time. We note, however, that these comparisons should be taken with a grain of salt, as the AKE provides mutual authentication while the others provide unilateral authentication.

#### 4.4.2.2 Password-Authenticated Key Exchange

Another paper [Gao+18] proposed employing the PAKE protocol of [Din+17] in TLS 1.2. Unlike standard AKE protocols, such as the one above, which uses pre-shared keys, PAKE protocols rely on pre-shared passwords for higher efficiency in exchange for weaker security.

The only performance data provided in [Gao+18] shows that the handshake with their PAKE takes on average about 4.9 ms, when both the server and the client are on the same machine,

which is an unrealistic setting. The authors also state that the transcript size of the PAKE key exchange is about 2–3× the size of classical (EC)DHE/RSA key exchange.

#### 4.4.2.3 Identity-based Encryption

The authors of [BC20] took a completely different approach. They proposed using an identity-based encryption (IBE) scheme together with a KEM for a mutually authenticated key exchange in TLS 1.3. In IBE schemes, the individual public key of each user can be publicly derived from a master public key and the user's identity. The master public key is generated by a trusted authority, which also distributes the individual private keys to the users. The main idea of [BC20] is to have the CA of the PKI create a master public key and then distribute the individual private keys to the clients and servers. The authors claim that this avoids the need for certificates. The authors of [BC20] implemented their approach in TLS 1.3 using a RLWE IBE scheme in combination with NewHope [Alk+16] as the KEM (Section 4.3). They stated that their approach was about 4.5× more efficient in terms of size and time than alternative certificate-based solutions, without specifying which solutions they were referring to.

## 4.5 Symmetric Primitives in Post-Quantum TLS

This section discusses the quantum security of the symmetric ciphers and hash functions used in TLS 1.3. Existing proposals for post-quantum TLS, presented in Sections 4.2 and 4.3, focus on the public-key cryptography in TLS. Consequently, there has been no explicit evaluation of the quantum security of the symmetric ciphers and hash functions used in TLS 1.3. This section aims to fill this gap.

### 4.5.1 Symmetric Ciphers

TLS 1.3 [Res18] supports three symmetric ciphers: AES-128 (128-bit keys), AES-256 (256-bit keys), and ChaCha20 (256-bit keys).

#### 4.5.1.1 Quantum Security of Symmetric Ciphers

Recall from Section 1.4.2 that Grover's quantum search algorithm [Gro96] provides a quadratic speedup over a classical exhaustive key search attack against symmetric ciphers such as AES, i.e., if a classical key search requires $2^{128}$ classical operations, a quantum key search requires $\sqrt{2^{128}} = 2^{64}$ *quantum* operations.

Although cryptographic algorithms that can be broken in $2^{64}$ operations are usually considered insecure [Sma+18; Bar20; BSI26], notable works [NIS16; MST21; Wes22; BSI26] still consider AES-128 to be quantum-secure because it is difficult to parallelize Grover's algorithm [Zal99] and thus exploit its full theoretical power.

On the other hand, ETSI recommends the use of symmetric ciphers with 256-bit keys, since these primitives "will withstand quantum computer attacks until way after 2050. For these primitives, it is not necessary to look for alternatives with the same urgency as the search for quantum-safe asymmetric primitives." [ETS17b]. The NSA also recommends the use of AES-256 [NSA22].

#### 4.5.1.2 Symmetric Cipher Recommendations

While symmetric ciphers with 128-bit keys are not as easily broken by Grover's algorithm as it might first appear, the conservative recommendation is to use 256-bit keys.

This means that all ciphers supported in TLS 1.3 can be considered quantum-secure, but it makes sense to use a symmetric cipher together with public-key cryptographic primitives (KEMs, signatures) with similar security levels. Specifically, we recommend using AES-128 with post-quantum KEMs or signature schemes with NIST security level 1, and AES-256 or ChaCha20 for those primitives with NIST security level $\geq 2$.

### 4.5.2 Hash Functions

TLS 1.3 [Res18] supports two different hash functions: SHA-256 and SHA-384.

#### 4.5.2.1 Quantum Security of SHA-256 and SHA-384

Recall from Section 1.4.2 that quantum security of hash functions is similar to the security of symmetric ciphers, since all known quantum attacks are based on Grover's algorithm [Gro96].

There are two types of attacks on hash functions that are improved by Grover's algorithm. First, while finding the pre-image of a hash function (with output size of $n$ bits) using brute force requires $2^n$ operations on a classical computer, Grover's quantum algorithm can reduce this complexity to $2^{\frac{n}{2}}$. Second, while finding a collision requires on average $2^{\frac{n}{2}}$ classical operations, the BHT quantum algorithm [BHT98] can reduce the complexity to $2^{\frac{n}{3}}$; however, the practical efficiency of the BHT algorithm is questionable since, according to [Ber09], the BHT algorithm has "fundamentally worse price-performance ratio" than classical collision search algorithms.

#### 4.5.2.2 Hash Function Recommendations

Both hash functions, SHA-256 and SHA-384, can be considered quantum-secure. Indeed, they correspond to the NIST security levels 2 and 4, respectively, [NIS16].

## 4.6 Post-Quantum TLS Performance Experiment

We conducted our own post-quantum TLS 1.3 experiment to measure the performance of purely and hybrid post-quantum authentication (Sections 4.2.1 and 4.2.2) and key exchange (Sections 4.3.1 and 4.3.2.1). While our purely post-quantum results were already presented in Tables 4.1 and 4.2, we provide more extensive results, which include hybrid post-quantum key exchange and authentication, and details about our methodology in this section.

### 4.6.1 Methodology

Let us describe our methodology—which hardware was used and how we measured TLS performance.

#### 4.6.1.1 Hardware

We used two `e2-highcpu-16` machines hosted on Google Cloud, one for the server and one for the clients. These machines have a 16-core x86 CPU (AMD/Intel based on availability [Goo23a]) with 16 GB of RAM. The RTT was 7ms and the bandwidth was 1.08Gb/s.

#### 4.6.1.2 Software

The OS was Ubuntu 22.04.3 LTS. We used the modified OpenSSL libraries from the Open Quantum Safe project [SM16]. Specifically, we used the *oqsprovider* [Ope23b] in combination with OpenSSL 3.2 alpha 2 [Ope23d].[9] For benchmarking purely and hybrid post-quantum signatures, as well as hybrid KEMs, we used the oqsprovider 0.5.2 (compiled with liboqs 0.9.0 [Ope23a]). For benchmarking purely post-quantum KEMs we had to use an older version (0.4.0) of the oqsprovider (compiled with liboqs 0.7.2) because the newer version supports only algorithms that were still in the NIST PQC competition. Liboqs was compiled with default (auto) optimization target.

See [Oup24b] for the benchmarking scripts; more details are provided in the README and comments.

#### 4.6.1.3 Measurement and Instantiation

We measured the TLS performance using the command `openssl s_time`, which outputs the number of connections per second. Each server process was established using the `openssl s_server` command. We ran 32 of these processes in parallel (32 servers and 32 clients) with 10 repetitions and `s_time` duration of 60s. When benchmarking signatures, the certificate chain length was 2 and the key exchange algorithm was x25519. When benchmarking key exchange, the certificate chain length was 2 and the signature algorithm was ECDSA with (P-256).

### 4.6.2 Results

The TLS slowdown coefficient is defined as $\frac{f_c}{f_{pq}}$, where $f_{pq}$ is the mean of connections per second for the post-quantum scheme and $f_c$ is the mean of connections per second for x25519 (for KEMs) or Ed25519 (for signatures). The specific parameter set is noted in parentheses, such as the security level and possibly other details.

The results of our evaluation of purely post-quantum signatures in TLS 1.3 are presented in Table 4.3. These results are visualized in Figure 4.3. The results of our evaluation of purely post-quantum KEMs in TLS 1.3 are presented in Table 4.5. These results are visualized in Figure 4.5. Note that we excluded from our results two NTRU level 5 parameter sets (NTRU-HPS-4096-1229, NTRU-HRSS-1373) because liboqs provides only their non-optimized implementation (compared to the rest).

The results of our evaluation of hybrid post-quantum signatures in TLS 1.3 are presented in Table 4.4. These results are visualized in Figure 4.4. The results of our evaluation of hybrid post-quantum KEMs in TLS 1.3 are presented in Table 4.6. These results are visualized in Figure 4.6.

[9] We used the alpha version because it allows to benchmark post-quantum signatures in TLS 1.3 with the oqsprovider. Otherwise, it is necessary to use the deprecated OpenSSL 1.1.1 fork that is available at [Ope23c].

**Table 4.3:** Purely Post-Quantum Signature TLS Performance

| **Algorithm** (security level and parameters) | **Connections/s** [mean ± st. dev.] | **TLS Slowdown Coefficient** |
|---|---|---|
| FN-DSA (L1) | 892.63 ± 20.38 | 0.98 |
| Ed25519 (control) | 875.83 ± 17.64 | 1.00 |
| ML-DSA (L2) | 854.54 ± 18.01 | 1.02 |
| ECDSA (P-256) | 839.76 ± 16.99 | 1.04 |
| FN-DSA (L5) | 809.83 ± 14.26 | 1.08 |
| ML-DSA (L3) | 782.14 ± 17.19 | 1.12 |
| ML-DSA (L5) | 734.33 ± 15.11 | 1.19 |
| SLH-DSA (L1, s, SHA-2) | 553.04 ± 47.68 | 1.58 |
| SLH-DSA (L1, f, SHA-2) | 390.84 ± 5.8 | 2.24 |
| SLH-DSA (L5, s, SHA-2) | 390.29 ± 31.09 | 2.27 |
| SLH-DSA (L5, f, SHA-2) | 296.22 ± 5.65 | 2.99 |

**Table 4.4:** Hybrid Post-Quantum Signature TLS performance

| **Algorithm** (security level and parameters) | **Connections/s** [mean ± st. dev.] | **TLS Slowdown Coefficient** |
|---|---|---|
| Ed25519 | 870.04 ± 23.45 | 1.00 |
| ECDSA (P-256) | 851.33 ± 17.85 | 1.02 |
| P-256 + FN-DSA (L1) | 669.76 ± 10.96 | 1.30 |
| P-256 + ML-DSA (L2) | 620.09 ± 10.42 | 1.40 |
| ECDSA (P-384) | 410.24 ± 5.73 | 2.12 |
| P-384 + ML-DSA (L3) | 334.10 ± 3.97 | 2.60 |
| ECDSA (P-521) | 246.24 ± 2.75 | 3.53 |
| P-521 + FN-DSA (L5) | 221.33 ± 3.30 | 3.93 |
| P-521 + ML-DSA (L5) | 214.19 ± 2.30 | 4.06 |

P-256/P-384/P-521 refers to ECDSA using corresponding NIST curves.

**Table 4.5**: Purely Post-Quantum KEM TLS Performance

| **Algorithm** (security level and parameters) | **Connections/s** [mean ± st. dev.] | **TLS Slowdown Coefficient** |
|---|---|---|
| x25519 (control) | 829.57 ± 15.48 | 1.00 |
| ML-KEM (L1) | 782.80 ± 14.03 | 1.06 |
| Saber (L1) | 776.79 ± 14.36 | 1.07 |
| ML-KEM (L3) | 760.13 ± 13.93 | 1.09 |
| NTRU Prime (L1, LPR) | 745.84 ± 15.26 | 1.11 |
| NTRU Prime (L2, LPR) | 746.60 ± 14.57 | 1.11 |
| Saber (L3) | 743.02 ± 14.89 | 1.12 |
| ML-KEM (L5) | 733.00 ± 15.24 | 1.13 |
| NTRU (L1, HPS) | 734.14 ± 13.45 | 1.13 |
| Saber (L5) | 733.30 ± 12.69 | 1.13 |
| NTRU Prime (L3, LPR) | 728.22 ± 14.31 | 1.14 |
| NTRU Prime (L5, LPR) | 720.67 ± 14.25 | 1.15 |
| NTRU (L3, HPS) | 700.24 ± 12.31 | 1.18 |
| NTRU (L3, HRSS) | 687.05 ± 13.80 | 1.21 |
| HQC (L1) | 676.69 ± 11.73 | 1.23 |
| NTRU (L5, HPS) | 661.54 ± 12.81 | 1.25 |
| x448 | 582.92 ± 9.10 | 1.42 |
| NTRU Prime (L1, S) | 572.24 ± 10.83 | 1.45 |
| HQC (L3) | 553.22 ± 9.20 | 1.50 |
| NTRU Prime (L2, S) | 543.28 ± 8.47 | 1.53 |
| NTRU Prime (L3, S) | 488.73 ± 6.80 | 1.70 |
| HQC (L5) | 420.72 ± 7.95 | 1.97 |
| NTRU Prime (L5, S) | 347.78 ± 4.39 | 2.39 |
| FrodoKEM (L1, AES) | 328.00 ± 3.59 | 2.53 |
| BIKE (L1) | 325.66 ± 3.42 | 2.55 |
| FrodoKEM (L3, AES) | 222.30 ± 2.53 | 3.73 |
| FrodoKEM (L5, AES) | 141.68 ± 1.59 | 5.86 |
| BIKE (L3) | 133.63 ± 1.01 | 6.21 |
| BIKE (L5)* | - | 15.26 |

* BIKE security level 5 ciphersuite is not present in the older versions of liboqs [Ope23a]. Therefore, we only present the slowdown coefficient which was computed from our other experiment which used the latest version (0.9.0).

**Table 4.6:** Hybrid Post-Quantum KEM TLS Performance

| **Algorithm** (security level and parameters) | **Connections/s** [mean ± st. dev.] | **TLS Slowdown Coefficient** |
|---|---|---|
| x25519 | 836.08 ± 13.27 | 1.00 |
| P-256 | 735.71 ± 10.91 | 1.14 |
| x25519 + ML-KEM (L1) | 670.77 ± 9.18 | 1.25 |
| x25519 + ML-KEM (L3) | 653.46 ± 10.19 | 1.28 |
| x25519 + HQC (L1) | 647.19 ± 10.27 | 1.29 |
| x448 | 571.88 ± 6.66 | 1.46 |
| x448 + ML-KEM (L3) | 423.81 ± 4.61 | 1.97 |
| P-256 + ML-KEM (L1) | 417.59 ± 4.59 | 2.00 |
| P-256 + ML-KEM (L3) | 411.93 ± 4.23 | 2.03 |
| P-256 + HQC (L1) | 392.19 ± 4.80 | 2.13 |
| x448 + HQC (L3) | 356.83 ± 3.23 | 2.34 |
| x25519 + FrodoKEM (L1) | 305.60 ± 2.68 | 2.74 |
| x25519 + BIKE (L1) | 300.34 ± 2.77 | 2.78 |
| P-256 + FrodoKEM (L1) | 233.12 ± 2.65 | 3.59 |
| P-256 + BIKE (L1) | 224.11 ± 2.38 | 3.73 |
| x448 + FrodoKEM (L3) | 178.15 ± 1.83 | 4.69 |
| P-384 | 175.47 ± 1.72 | 4.76 |
| P-384 + ML-KEM (L3) | 143.59 ± 1.51 | 5.82 |
| P-384 + HQC (L3) | 130.47 ± 0.89 | 6.41 |
| x448 + BIKE (L3) | 101.95 ± 0.70 | 8.20 |
| P-384 + FrodoKEM (L3) | 100.40 ± 0.98 | 8.33 |
| P-521 | 77.89 ± 0.44 | 10.73 |
| P-521 + ML-KEM (L5) | 72.55 ± 0.67 | 11.52 |
| P-521 + HQC (L5) | 69.41 ± 0.56 | 12.05 |
| P-384 + BIKE (L3) | 66.30 ± 0.34 | 12.61 |
| P-521 + FrodoKEM (L5) | 49.02 ± 0.57 | 17.06 |
| P-521 + BIKE (L5) | 30.27 ± 0.22 | 27.62 |

P-256/P-384/P-521 refers to ECDH using corresponding NIST curves.

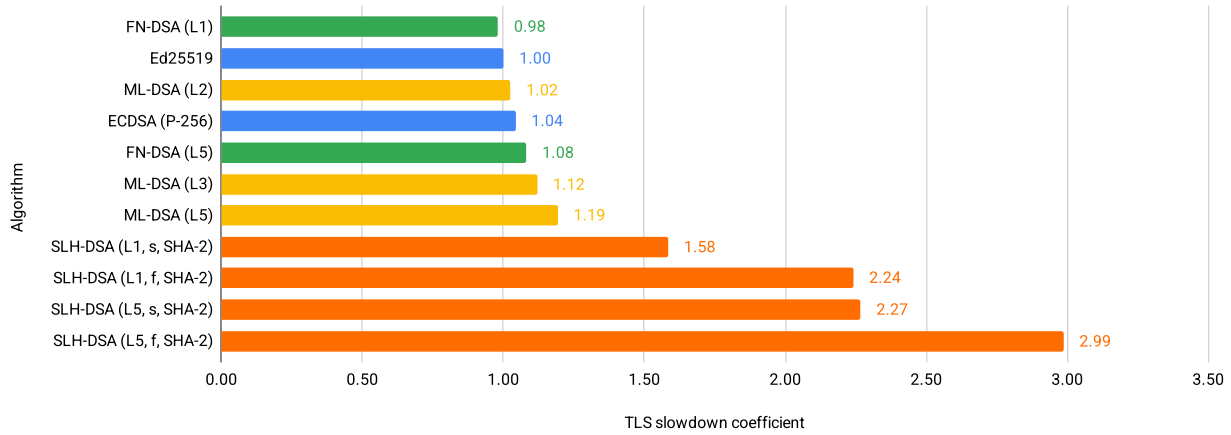



**Figure 4.3:** Purely Post-Quantum Signature TLS Slowdown Coefficient

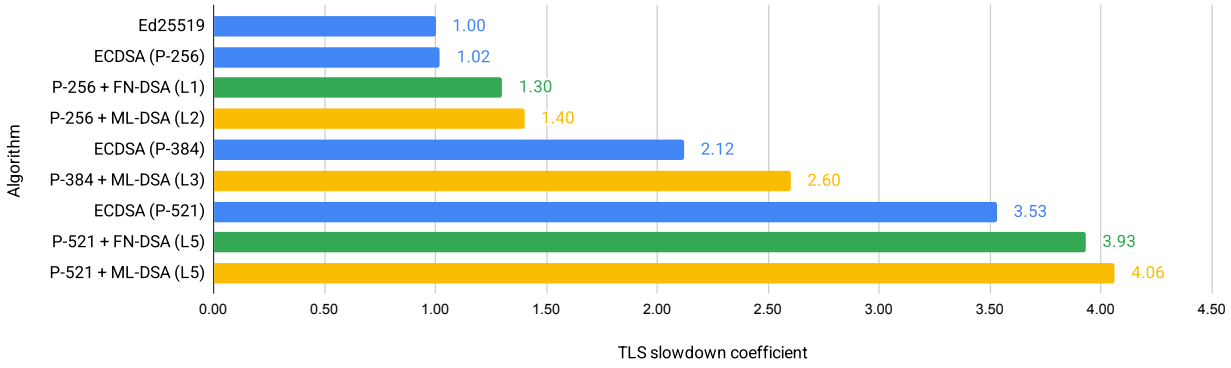



**Figure 4.4:** Hybrid Post-Quantum Signature TLS Slowdown Coefficient

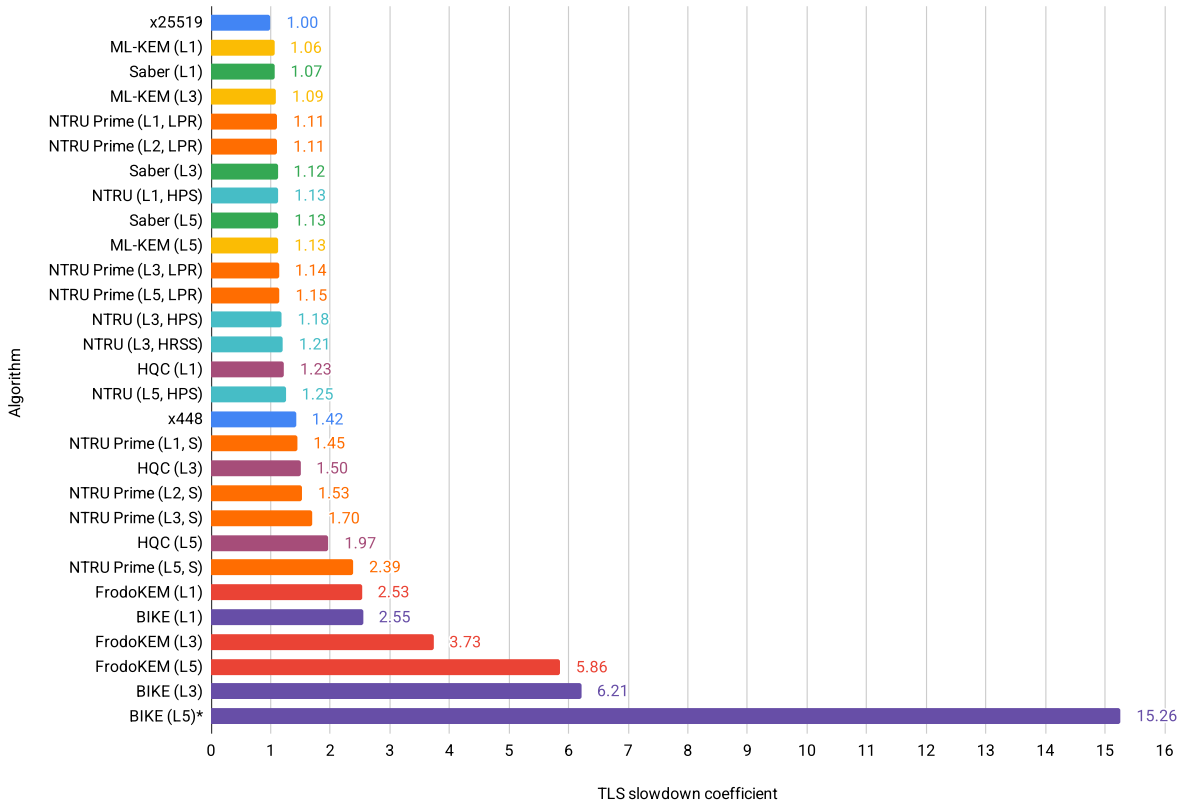



**Figure 4.5**: Purely Post-Quantum KEM TLS Slowdown Coefficient

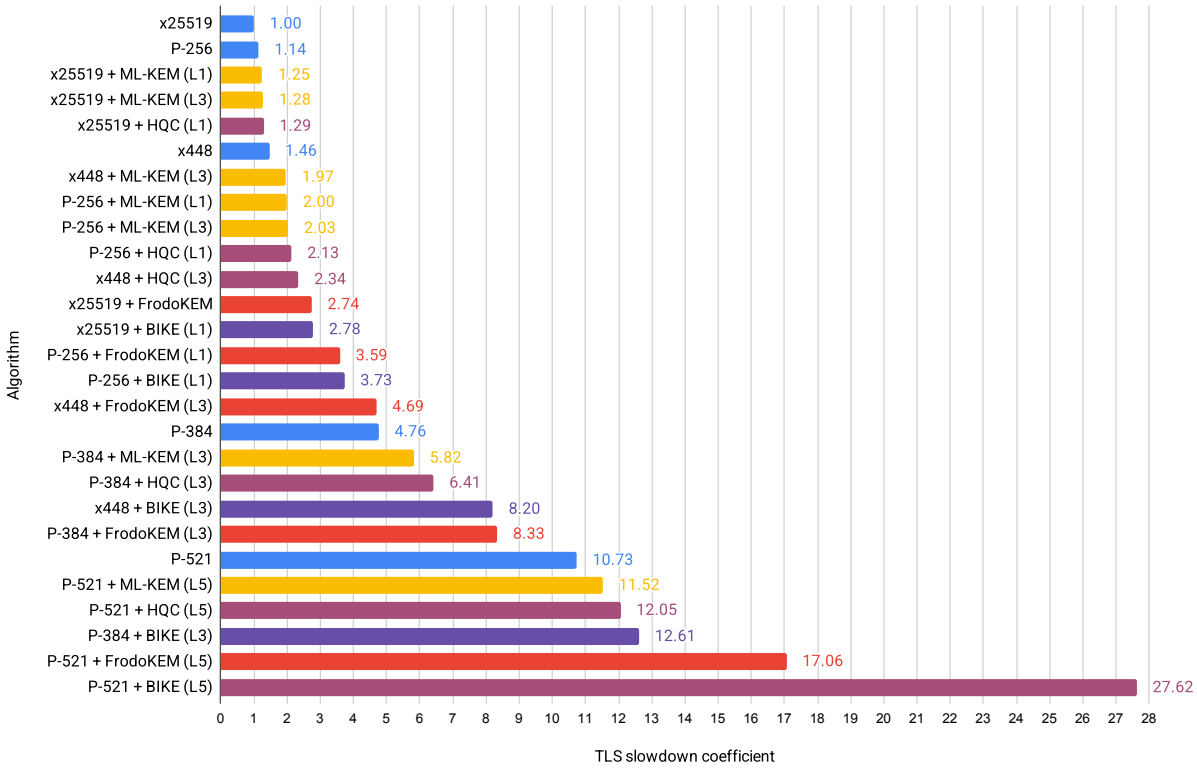



**Figure 4.6**: Hybrid Post-Quantum KEM TLS Slowdown Coefficient

## 4.7 Conclusion

This chapter provided a comprehensive survey and performance evaluation of post-quantum TLS. We began by introducing the basics of TLS 1.3 and by analyzing TLS security with respect to quantum adversaries. We categorized post-quantum TLS proposals into three categories based on which part of TLS they focus on: authentication, key exchange, and authenticated key exchange.

Our analysis and subsequent benchmarks show that migration of TLS key exchange to post-quantum security is highly practical today and necessary to prevent "store now, decrypt later" attacks. Specifically, the concatenation hybrid post-quantum key exchange instantiated with ECDHE and ML-KEM is the most promising option; it provides hybrid post-quantum security with a negligible overhead in most scenarios. In fact, this approach has already been in use for multiple years by Google and others. Eventually, purely post-quantum key exchange will be deployed, a process made straightforward by the crypto-agility of TLS.

Migrating TLS authentication to post-quantum security is generally a greater challenge due to its dependency on a post-quantum PKI. However, assuming the existence of a post-quantum PKI, the technical implementation is as straightforward as in the case of TLS key exchange. However, the performance impact of post-quantum authentication will be larger compared to post-quantum key exchange. This is because several signatures and public keys are needed for a typical TLS authentication, whereas a (non-hybrid) key exchange requires only one public key and one ciphertext.

This motivated the invention of alternative approaches, such as the use of Merkle Tree Certificates for authentication, or the redesign of the TLS handshake to use KEMs instead of signatures, dubbed KEMTLS. These approaches are certainly interesting, but they require significantly more work to implement compared to the straightforward drop-in replacements. It remains to be seen whether they will be adopted at large.

# Chapter 5

# Post-Quantum XML and SAML


5.1 **XML Security** . . . 128
5.1.1 XML Signatures . . . 128
5.1.2 XML Encryption . . . 131
5.1.3 Quantum Threat Analysis . . . 135
5.2 **Post-Quantum XML Security** . . . 135
5.2.1 Post-Quantum XML Signatures . . . 135
5.2.2 Post-Quantum XML Encryption . . . 139
5.3 **SAML Framework** . . . 143
5.3.1 SAML Single Sign-On . . . 143
5.3.2 SAML SSO Security . . . 145
5.3.3 Quantum Threat Analysis . . . 146
5.4 **Post-Quantum SAML SSO** . . . 147
5.4.1 Purely Post-Quantum SAML SSO . . . 147
5.4.2 Hybrid Post-Quantum SAML SSO . . . 148
5.5 **Conclusion** . . . 153


Every one of us has multiple user accounts tied to various services, such as emails, banking, and shopping. The inevitable consequence of having multiple accounts is that it introduces the need to remember many credentials—assuming the user follows best practices and does not reuse passwords. Tools such as password managers aid with this problem, but this also comes with drawbacks.

Single Sign-On (SSO) addresses this problem. SSO is a method of authentication where the user only needs to know one set of credentials to access multiple independent services. One of the most prominent SSO standards, particularly in the enterprise, is the Security Assertion Markup Language (SAML) framework. One of the building blocks of SAML is Extensible Markup Language (XML). XML is a markup language used for data serialization. It is one of the most popular markup

languages and has a large ecosystem built around it. The XML ecosystem includes standardized security mechanisms, such as XML Encryption and XML Signatures, which are used in XML-based protocols such as SAML SSO.

We introduce XML-based signatures and encryption in Section 5.1 and discuss their security with respect to quantum adversaries. Subsequently, in Section 5.2, we describe post-quantum solutions for XML signatures and XML encryption. After that, Section 5.3 presents the SAML SSO protocol and provides an analysis of its quantum security. We conclude this chapter with Section 5.4, in which we describe how to achieve post-quantum security in SAML SSO with the help of the XML building blocks introduced in Section 5.2.

**Remark**

The content of this chapter is based on our published work titled "Post-quantum XML and SAML Single Sign-On" [MO24].

## 5.1 XML Security

In this section, we present the XML Signature [W3C13c] and XML Encryption [W3C13a] standards. Although the XML signature and encryption standards provide multiple methods, we focus here on those methods that are relevant for SAML (Section 5.3).

### 5.1.1 XML Signatures

Suppose you want to sign some data. Fundamentally, the data to be signed are a binary file, a set of bytes. The same applies for the resulting signature. In many use cases, there is a need for the resulting signature to be tied together with the signed data. For example, we produce a document and sign it with the intention of using the signature as a proof of authenticity. It is impractical to send these two files, the document and its signature, separately. The XML signature standard [W3C13c] provides a standardized way to combine these files into a single "envelope" in the form of an XML file.

**Remark**

In Chapter 6 we discuss in detail other methods of unifying signed data—one of those methods, XAdES, is an extension of XML signatures presented here.

An XML signature can be applied to any type of data, it can be an XML document or a binary file. In either case, the resulting signature is represented as an XML element `Signature`.

The XML signature standard [W3C13c] defines three types of XML signatures:

- *Enveloping*: The signed data[1] is an XML element `Object`, which is a child of `Signature`.
- *Enveloped*: The signed data is an XML document. `Signature` is a descendant of this document.

[1] We are slightly misusing the term "signed data," since the input to the cryptographic signing operation is only the element `SignedInfo`, which contains a hash of the "document to be signed."

- *Detached*: The signed data is external to `Signature`, i.e., `Signature` is a separate XML document.

From now on, we focus on *enveloped* XML signatures as they are the only supported type in SAML.

#### 5.1.1.1 XML Signature Structure

```xml
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:AuthnRequest ID="_87d28f1e-72d7-45ea-9df1-977ad466cf47" (other attributes...)>
   <samlp:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity" xmlns:samlp="urn:oasis:names:tc:SAML:2.0:
        assertion">https://desktop.luxtrust.lu</samlp:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
         <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
         <ds:Reference URI="#_87d28f1e-72d7-45ea-9df1-977ad466cf47">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
            <ds:DigestValue><!-- base64-encoded digest --></ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue><!-- base64-encoded signature --></ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate><!-- base64-encoded certificate --></ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
   </ds:Signature>
   <saml2p:Extensions>
      <!-- custom extension data -->
   </saml2p:Extensions>
</saml2p:AuthnRequest>
```

**Listing 5.1**: XML Signature Example

Listing 5.1 provides an example of an enveloped XML signature. Specifically, this is a SAML message (Section 5.3), but we use it here in the context of XML signatures, so we ignore SAML specificities. The document root is the element `AuthnRequest` with multiple attributes. The signature-relevant attribute is the `ID` attribute. It is used to reference the XML element (the signed data).

`AuthnRequest` has three child elements: `Issuer`, `Extensions` and `Signature`.

`Issuer` and `Extensions` are SAML-related elements and are not relevant to the XML signing process.

`Signature` contains three child elements:

- `SignedInfo` is the actual input into the signature algorithm. It contains the elements:
  - `CanonicalizationMethod` indicates the method for canonicalization of the `SignedInfo` element. Canonicalization processes the XML element into a standard format so that it is unambiguous. This is important for signature verification.

    In Listing 5.1, the canonicalization method is `http://www.w3.org/2001/10/xml-exc-c14n#` (Exclusive XML Canonicalization), this method is recommended in the SAML specification.

- `SignatureMethod` indicates the underlying signature algorithm used.
  In Listing 5.1, the signature algorithm is `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256` which refers to the PKCS#1 v1.5 RSA signature with the hash function SHA-256.
- `Reference` is used to reference XML elements that are protected with the signature.
  In Listing 5.1, the element has the attribute `URI` which points to the element with the `ID="_87d..."` which is the root element `AuthnRequest`.
  This element contains three children:
  * `Transforms` contains two child `Transform` elements. They define how to process the referenced element, in our case `AuthnRequest`, before its hash (digest) is calculated.[2]
    In Listing 5.1, the first `Transform` algorithm is `http://www.w3.org/2000/09/xmldsig#enveloped-signature` which indicates that the `Signature` element needs to be removed from the XML document before calculating the digest. The second algorithm specifies the canonicalization method to be applied to the element; the same one is used to process `SignedInfo`.
  * `DigestMethod` indicates the hash algorithm used to calculate the digest of the referenced element after applying the transforms.
    In Listing 5.1, the digest algorithm is SHA-256.
  * `DigestValue` contains the Base64-encoded output from the digest algorithm.

- `SignatureValue` contains the Base64-encoded output from the signature algorithm.
- `KeyInfo` contains the information about the public key that can be used to verify the signature.

  In Listing 5.1, the element contains an element `X509Data` containing another element `X509Certificate`. The `X509Certificate` contains a Base64-encoded X.509 certificate with the public key.

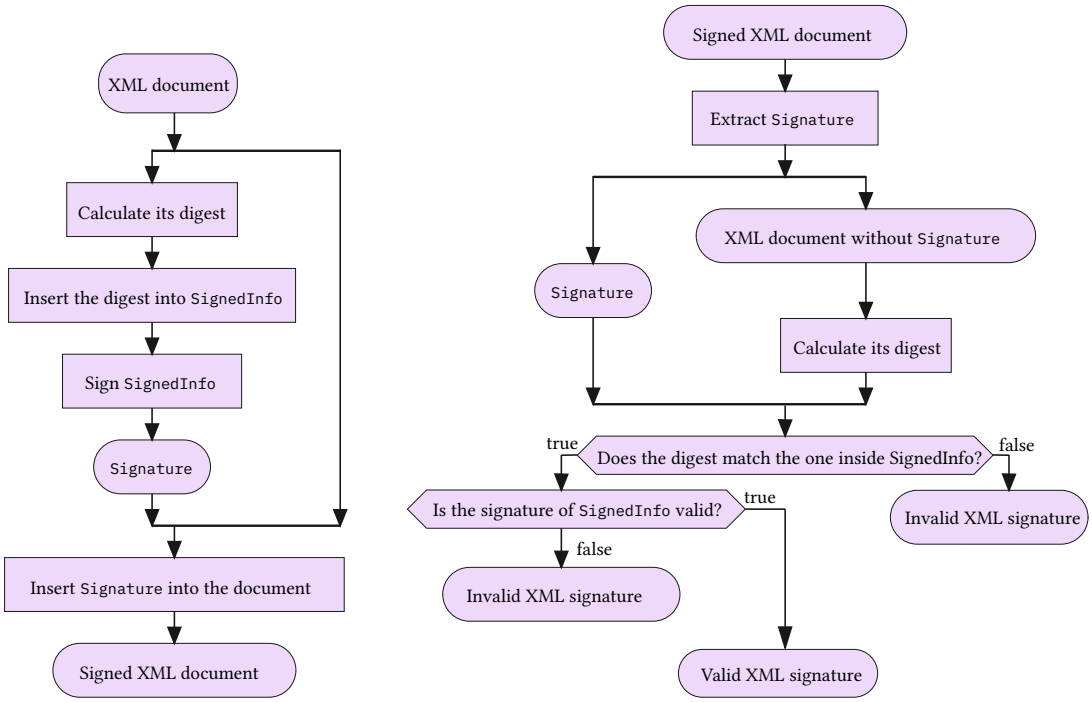
  The `X509Data` may contain multiple `X509Certificate` elements forming a certificate chain.

#### 5.1.1.2 Signing Procedure

An enveloped signature of an XML document is created as follows:

1) Prepare a `Signature` element with predefined algorithms for canonicalization, hashing, and signing.
2) Calculate the digest of the XML document and prepare a `Reference` element with the resulting digest. Since this is an enveloped signature, the `Reference` must contain the *enveloped signature transform* which removes the `Signature` element from the digest computation.
3) Sign `SignedInfo` using the signature algorithm. Put the result into `SignatureValue`.
4) Prepare `KeyInfo` containing the verification public key.

[2] The order of the transformations corresponds to the order of the `Transform` elements.




(a) XML Signature Creation (b) XML Signature Verification

**Figure 5.1:** XML Signature Procedures

5) Paste the resulting `Signature` into the original XML document. The result is a signed XML document.

Figure 5.1a provides a simplified diagram of the signing procedure.

#### 5.1.1.3 Verification Procedure

Verification works as follows:

1) Check if `SignatureValue` is a signature of `SignedInfo` using the public key specified in `KeyInfo` and the algorithm specified in `SignedInfo`. If invalid, abort.

2) Check if the digest of the document matches the one in `Reference`. If it matches, the XML signature is valid.

Figure 5.1b provides a simplified diagram of the verification procedure. While the order of the verification steps can be reversed (verify the reference digest(s) first, then verify the signature), it is not recommended as it allows for Denial of Service (DoS) attacks [W3C13b].

### 5.1.2 XML Encryption

In practice, it is necessary to provide metadata, such as the used encryption algorithm, with ciphertexts, since a ciphertext is a set of random bytes. The XML Encryption standard [W3C13a] defines how to tie together the ciphertext with the relevant metadata to form an XML document.

Although the standard defines various ways to represent the data, we only focus on the variant that is used by SAML (Section 5.3).

The XML element that contains the encrypted data (and the metadata about the encryption) is `EncryptedData`. It is the encryption equivalent of the `Signature` element. Listing 5.2 provides an example XML document with encryption. In this example, only a part of the XML document contains encryption-related data.

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:Response ID="LT_56c27b2a-4d6b-4bd5-a005-08421d2a2d63" InResponseTo="_87d28f1e-72d7-45ea-9df1-977ad466cf47" (other attributes...)>
   <samlp:Issuer xmlns:samlp="urn:oasis:names:tc:SAML:2.0:assertion" Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">https://luxtrust.lu/SAML</samlp:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
         <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
         <ds:Reference URI="#LT_56c27b2a-4d6b-4bd5-a005-08421d2a2d63">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
            <ds:DigestValue><!-- base64-encoded digest --></ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue><!-- base64-encoded signature --></ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate><!-- base64-encoded certificate --></ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
   </ds:Signature>
   <saml2p:Status xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol">
      <saml2p:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success" />
   </saml2p:Status>
   <saml2:EncryptedAssertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
      <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="_67fcc95f0a9c8cc85382696da871e7c3" Type="http://www.w3.org/2001/04/xmlenc#Element">
         <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" />
         <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <xenc:EncryptedKey Id="_8b1889201540681e7809a50186b5bf39" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
               <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" />
               </xenc:EncryptionMethod>
               <ds:KeyInfo>
                  <ds:X509Data>
                     <ds:X509Certificate><!-- base64-encoded certificate --></ds:X509Certificate>
                  </ds:X509Data>
               </ds:KeyInfo>
               <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                  <xenc:CipherValue><!-- base64-encoded encrypted key --></xenc:CipherValue>
               </xenc:CipherData>
            </xenc:EncryptedKey>
         </ds:KeyInfo>
         <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            <xenc:CipherValue><!-- base64-encoded encrypted data --></xenc:CipherValue>
         </xenc:CipherData>
      </xenc:EncryptedData>
   </saml2:EncryptedAssertion>
</saml2p:Response>
```

**Listing 5.2:** XML Encryption Example

The root element is `Response`. It has four child elements: `Issuer`, `Signature`, `Status` and `EncryptedAssertion`. The `Issuer` and `Status` elements are specific to the SAML protocol (Section 5.3) and are not relevant for this example. `Signature` indicates that this response is signed using XML signatures (Section 5.1.1)—this is independent of the XML encryption. The XML element relevant to XML encryption is `EncryptedAssertion`. Specifically, its content, `EncryptedData`, is relevant here, since `EncryptedAssertion` is a SAML-specific element.

`EncryptedData` has three children:

- `EncryptionMethod` indicates the encryption algorithm used.

  In Listing 5.2, the encryption algorithm is `http://www.w3.org/2001/04/xmlenc#aes256-cbc` which refers to the AES-256 in CBC mode with padding defined in [W3C13a].

- `CipherData` has one child element `CipherValue` which contains the Base64-encoded encrypted data using the algorithm from `EncryptionMethod`.

- `KeyInfo` serves the same purpose as in the signature process, i.e., it contains information about the encryption key.

  It has one child, `EncryptedKey`. This element contains the encrypted encryption key (the encrypted AES-256 key). It has three children:

  - `EncryptionMethod` indicates the key encryption algorithm.

    In Listing 5.2, the key encryption algorithm is `http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p` which refers to RSA encryption with OAEP padding using the mask generating function SHA-1.

  - `CipherData` has one child element `CipherValue` which contains the Base64-encoded encrypted key data using the algorithm from `EncryptionMethod`.

  - `KeyInfo` contains the information about the key encryption key.

    In Listing 5.2, the element contains an element `X509Data` containing another element `X509Certificate`. The `X509Certificate` contains a Base64-encoded X509 certificate with the RSA public key.

#### 5.1.2.1 Encryption Procedure

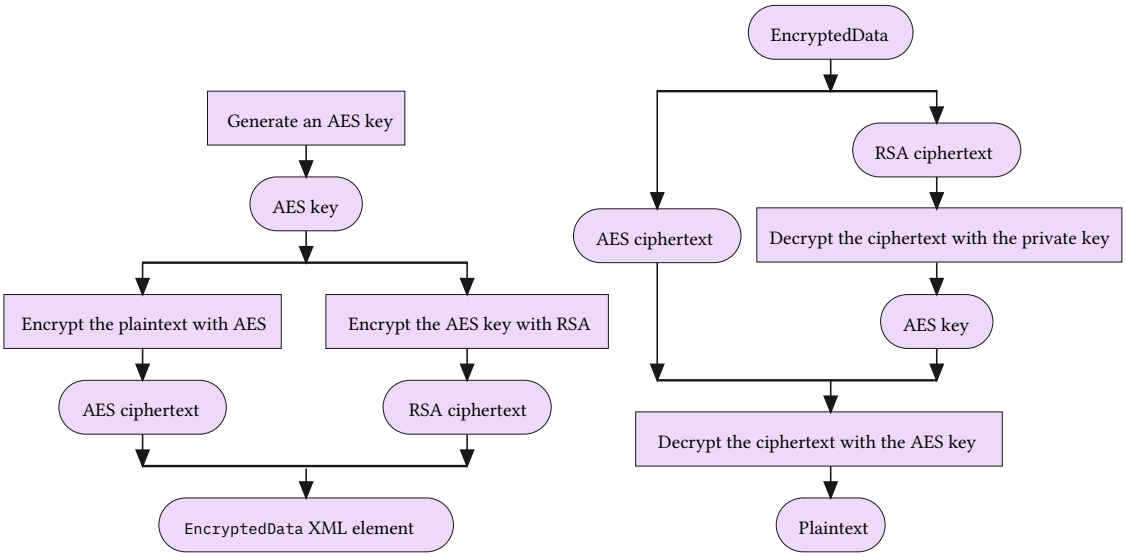
While the XML encryption standard can be implemented with different cryptographic primitives, we describe the case where RSA is the key encryption algorithm and AES-256 is the data encryption scheme:

1) Generate an AES-256 key.

2) Encrypt the plaintext with AES-256 using the generated key.

3) Encrypt the AES-256 key with RSA using the recipient's RSA public key.

4) Create the XML element `EncryptedData`. The AES ciphertext is placed inside its `CipherData` child while the RSA-encrypted AES key is placed inside `KeyInfo`.




(a) XML Encryption (b) XML Decryption

**Figure 5.2:** XML Encryption Procedures

See Figure 5.2a for a simplified diagram of the encryption procedure. The encryption result is represented by the `EncryptedData` element. It contains ciphertexts and their metadata. The metadata typically includes the encryption algorithm, in the form of the `EncryptionMethod` element. It also includes information about the encryption key, in the form of the `KeyInfo` element, which consists of the `EncryptedKey` element containing the RSA ciphertext and metadata about the encryption algorithm and public key.

**Remark**

Formally, the XML encryption procedure is an instance of *hybrid encryption* (Definition 12)—the combination of public-key encryption with symmetric-key encryption. Note that this is unrelated to the post-quantum term "hybrid" defined in Definition 22.

#### 5.1.2.2 Decryption Procedure

The decryption is the encryption process in reverse: the AES key is decrypted using the RSA private key, and then it is used to decrypt the actual data. See Figure 5.2b for a diagram of the decryption procedure.

#### 5.1.2.3 Authentication

Although the XML encryption standard [W3C13a] does not explicitly focus on ciphertext authentication, the ciphertext can be authenticated by using an authenticated encryption (AE) scheme (Definition 3), such as AES-GCM [Dwo07]. However, the encrypted symmetric-key ciphertext is not authenticated as it is encrypted (typically) using a public-key encryption scheme. For this reason, XML encryption is usually combined with XML signatures.

### 5.1.3 Quantum Threat Analysis

Compared to, e.g., TLS (Chapter 4), the threat analysis for the XML encryption and signing is fairly straightforward.

- XML signatures: If any classical signature algorithm is used, then a quantum attacker can forge any signature. This can be achieved by computing the private key corresponding to the public key. For RSA or DSA, it takes only one call to the quantum oracle to calculate the corresponding private key.
- XML encryption: If any classical public-key encryption algorithm is used, then a quantum attacker can decrypt the ciphertext. If only symmetric ciphers are used (assuming a pre-shared key scenario), at least 256-bit keys shall be used.

#### 5.1.3.1 Classical Security of XML

Throughout the years, there have been attacks [JSS12; JS11; Kup+15] on XML signatures and encryption. These attacks are primarily caused by the complexity of the XML standards since they are hard to implement correctly and sometimes are under-defined. Some of the attacks exploit use cases in which the encryption is not authenticated which leads to standard padding oracle attacks. Other attacks exploit implementation bugs in XML security libraries.

## 5.2 Post-Quantum XML Security

The security of SAML, which we discuss in Section 5.3, relies on XML signatures and encryption as building blocks. Recall from Section 5.1.3 that XML signatures and encryption need to be upgraded to obtain post-quantum security.

This section presents multiple approaches to post-quantum XML encryption and signatures. In Section 5.2.1, we present post-quantum XML signatures, and in Section 5.2.2, we present post-quantum XML encryption. Similar to other solutions in this work, we propose two variants of post-quantum solutions: *purely post-quantum* (Definition 23) and *hybrid post-quantum* (Definition 22).

### 5.2.1 Post-Quantum XML Signatures

We discuss how to achieve post-quantum security for XML signatures. We focus on *purely post-quantum* security in Section 5.2.1.1, and then we discuss *hybrid post-quantum* solutions in Section 5.2.1.2.

#### 5.2.1.1 Purely Post-Quantum XML Signatures

Recall from Section 5.1.1 that the XML signature standard [W3C13c] is cryptographically agile—it works with high-level primitives, such as signatures and hash functions, rather than specific algorithms, such as RSA or SHA-2. Consequently, we "only" need to choose post-quantum instances of these primitives to achieve purely post-quantum XML signatures.

Listing 5.3 is an example XML document `AuthnRequest` signed by ML-DSA, a post-quantum signature, and digested with SHA3-256, a hash function with 256-bit output, which is secure against quantum adversaries (Section 1.4.2).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" ID="_30de6016-b33f-44ab-9190-1026ad8c84c8" (other
      attributes...)>
   <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://desktop.luxtrust.lu</saml2:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2023/02/xmldsig-pqc#dilithium"/>
         <ds:Reference URI="#_30de6016-b33f-44ab-9190-1026ad8c84c8">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2007/05/xmldsig-more#sha3-256"/>
            <ds:DigestValue><!-- base64-encoded SHA3-256 digest --></ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue><!-- base64-encoded Dilithium-2 signature --></ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate><!-- base64-encoded Dilithium-2 certificate --></ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
   </ds:Signature>
   <saml2p:Extensions>
      <!-- custom extension data -->
   </saml2p:Extensions>
</saml2p:AuthnRequest>
```

**Listing 5.3:** Post-Quantum XML Signature Example

The purely post-quantum XML signature structure is identical to the standard (classical) XML signature, except that post-quantum identifiers are used in `SignatureMethod`.

**Purely Post-Quantum XML Signature Performance.** We conducted an experiment in which we signed XML documents with different purely post-quantum XML signatures and measured their performance.

*Experimental Setup.* For all experiments in this chapter, we use the following setup. Our implementation [Oup24c] consists of forks of the following libraries: OpenSAML [Shi25], Apache Santuario [Apa25], and BouncyCastle [Leg25b]. The machine used for measuring was an Oracle Cloud VM.Standard.A1.Flex (Arm64) running Ubuntu 22.04.3 LTS. OpenSAML was used to create, sign and verify SAML messages. It relies on the XML security library Apache Santuario and BouncyCastle for cryptographic primitives.

To make our experiment realistic, we chose XML documents in a real-world framework, namely the SAML messages `AuthnRequest` and `Response`, which vary in size. We simulated a typical SAML SSO exchange (Section 5.3), which includes signing and verifying `AuthnRequest` and `Response`, 10000 times, and computed the average time to sign and verify messages, as well as the total time of the process. We also computed the average size of the `Signature` element inside `AuthnRequest` and `Response`, as well as the average size of the complete XML document.

*Results.* The results are presented in Table 5.1. For each metric, we compare the relative change against the state-of-the-art classical signature algorithm ECDSA with curve P-256.

Using post-quantum XML signatures increases the document size significantly (at least 2×), but for larger XML documents the relative size increase becomes smaller since `Signature`'s size

**Table 5.1:** Purely Post-Quantum XML Signature Performance

| Algorithm | `Signature` Size | Document Size | Signing Time | Verification Time | Total Time |
|---|---|---|---|---|---|
| ECDSA (P-256) | 1563 | 2513 | 0.28 ± 0.05 | 0.51 ± 0.06 | 1.58 ± 0.12 |
| ML-DSA (2) | 9816 (×6.28) | 10766 (×4.28) | 0.85 ± 0.23 (×3.04) | 0.37 ± 0.06 (×0.73) | 2.46 ± 0.13 (×1.56) |
| FN-DSA (1) | 4303 (×2.75) | 5253 (×2.09) | 1.35 ± 0.18 (×4.82) | 0.25 ± 0.04 (×0.49) | 3.18 ± 0.26 (×2.01) |
| ML-DSA (5) | 17703 (×11.33) | 18653 (×7.42) | 1.13 ± 0.14 (×4.04) | 0.84 ± 0.07 (×1.65) | 3.94 ± 0.15 (×2.49) |
| FN-DSA (5) | 7229 (×4.63) | 8179 (×3.25) | 2.71 ± 0.43 (×9.68) | 0.41 ± 0.09 (×0.80) | 6.23 ± 0.76 (×3.94) |
| ECDSA (P-521) | 1843 (×1.18) | 2793 (×1.11) | 0.93 ± 0.07 (×3.32) | 3.29 ± 0.15 (×6.45) | 8.45 ± 0.26 (×5.35) |
| RSA (3072 bits) | 2898 (×1.85) | 3848 (×1.53) | 10.60 ± 0.22 (×37.86) | 0.58 ± 0.14 (×1.14) | 22.36 ± 0.46 (×14.15) |
| SLH-DSA (1) | 23220 (×14.86) | 24170 (×9.62) | 1023.13 ± 6.71 (×3654.04) | 2.07 ± 0.52 (×4.06) | 2050.40 ± 12.46 (×1297.72) |
| RSA (15360 bits) | 9258 (×5.92) | 10208 (×4.06) | 1193.92 ± 6.58 (×4264) | 11.03 ± 0.43 (×21.63) | 2409.9 ± 12.40 (×1525.25) |
| SLH-DSA (5) | 84456 (×54.03) | 85406 (×33.99) | 1581.27 ± 9.96 (×5647.39) | 4.45 ± 0.67 (×8.73) | 3171.45 ± 18.37 (×2007.25) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. SHA2-512 was used as a reference hashing function. ECDSA with SHA2-256 for P-256, SHA2-512 for P-521, RSA PKCS# v1.5 with SHA2-512. Included certificate chain length 1. Assertions not encrypted.

remains constant.

Signing with post-quantum schemes is slower than signing with ECDSA (P-256), but the average signing times for ML-DSA, at both security levels 2 and 5, and FN-DSA, at security level 1, are better compared to RSA and ECDSA (P-521). Notably, SLH-DSA's signing times are more than 100× slower than the others, except for 15360-bit RSA.[3]

FN-DSA's verification time at security level 1 is more than twice as fast as ECDSA (P-256). ML-DSA's (security level 2) and FN-DSA's (security level 5) times are also about 30% faster than ECDSA (P-256). SLH-DSA verification is slower than ECDSA (P-256), but only by a factor of 4 (as opposed to a factor of 3654 for signing).

Among post-quantum signatures, ML-DSA (security level 2) comes out on top when considering the total time (2× signing + 2× verifying), being only 56% slower than ECDSA (P-256). At all security levels, ML-DSA and FN-DSA are faster than ECDSA (P-521) and RSA. Due to the slow signing time, SLH-DSA is the slowest post-quantum signature in total.

Note that in unstable or low-bandwidth networks, the larger signature sizes of ML-DSA and the other post-quantum algorithms can make these solutions even slower compared to the lighter classical ones.

#### 5.2.1.2 Hybrid Post-Quantum XML Signatures

This section discusses approaches to hybrid post-quantum XML signatures. Recall from Section 5.1.1 that we focus on *enveloped* XML signatures where the `<Signature>` element is a child of the XML document which is signed. The focus on enveloped signatures is motivated by their use in the SAML SSO protocol discussed in Section 5.3.

We want our hybrid post-quantum XML signature to have the following desirable properties:

- *Backward Compatibility*: The hybrid post-quantum XML signature should also be verifiable by a party that only supports classical signature schemes. This property is important for a smooth transition phase to post-quantum cryptography.

[3] We included 15360-bit RSA as a representative of security level 5 RSA. However, such large parameters are not used in practice to the best of our knowledge.

- *Non-separability*: Recall from Section 2.4.5.1 that a hybrid signature scheme is non-separable if an adversary cannot strip the post-quantum component of the signature and obtain a valid classical signature (or vice versa).

The combination of these two features provides the two main advantages of the hybrid post-quantum approach: compliance with current regulations that only support classical schemes, and mitigation of the risk that the post-quantum part turns out to be less secure than intended.

Hybrid post-quantum XML signatures can be realized in essentially two ways. The first approach, *Composite Hybrid*, uses a hybrid post-quantum signature scheme (Section 2.4.5.1) instead of a classical signature scheme and proceeds according to the standard [W3C13c]. The problem with this first approach is that it is not backward compatible. The second approach, *Separate Hybrid*, aims to solve this.

**Composite Hybrid.** This approach is fairly straightforward. Recall from Section 5.1.1 that the XML signature standard [W3C13c] is cryptographically agile. Consequently, we get post-quantum hybrid XML signatures by using a hybrid signature scheme (Section 2.4.5.1) instead of a classical one. The particular hybrid post-quantum signature must be properly identified using the URI identifier in the `<SignatureMethod>` element.

However, the approach comes with a disadvantage. It is not backward compatible: the resulting signature cannot be verified by a classical verifier since the hybrid signature scheme is not supported by the classical-only verifier.

Recall from Section 2.4.5.1 that hybrid signature schemes can be non-separable, e.g., the prefixed parallel composite scheme (Definition 24).

**Separate Hybrid.** We now present the separate hybrid approach, which uses two separate `Signature` elements, one to represent the classical signature and the other to represent the post-quantum signature. Unlike the composite hybrid approach (Section 5.2.1.2), the separate hybrid approach provides backward compatibility.

The most important design decision for implementing the separate hybrid approach is to decide which message to sign. The naive way is to sign the same XML document with both signatures independently. The resulting `Signature` elements are then placed inside the XML document as usual for enveloped signatures. Although the XML signature standard [W3C13c] does not prohibit placing multiple `Signature` elements within an XML document, there are several problems with this naive approach:

- Although we require that the entire document signature is valid if and only if the classical `Signature` *and* the post-quantum `Signature` are valid, it cannot be guaranteed that real-world implementations will check this conjunction, only the disjunction.
- If the standard XML signature verification procedure (Figure 5.1b) is used, then (at least) one of the XML signatures will fail verification. This is because the enveloped signature transform, only removes its parent `Signature` from the reference digest calculation, not the other `Signature`.
- It does not provide the desirable feature of non-separability.

Our solution to achieve backward compatibility and non-separability[4] is to sign sequentially. In this case, the XML document $d$ contains two `Signature` elements $S_A, S_B$: $S_A$ is a signature over the original document $d$, and $S_B$ is a signature over $(S_A, d)$. However, if we use the standard verification procedure, only $S_B$ verifies successfully. $S_A$ does not, because it is a signature over $d$ only. Although it contains the enveloped signature transformation, it does not remove the second $S_B$ from the hash calculation. To avoid this problem, we need to add an extra transformation to $S_A$ that removes the other signature as well.

There is no generic solution for how this transform should be defined. This is because each XML-signed document has a different structure. We present a transform tailored to the SAML SSO protocol messages in Section 5.4.

**Hybrid Post-Quantum XML Signature Performance.** Analogous to purely post-quantum XML signatures (Section 5.2.1.1), we conducted an experiment to benchmark the performance of hybrid post-quantum XML signatures. We have tested both presented hybrid variants: the composite hybrid and the separate hybrid. Specifically, for the composite hybrid signatures we used the IETF draft composite signatures [Oun+25a].

*Experimental Setup.* The experimental setup is identical to the purely post-quantum experiment from Section 5.2.1.1.

*Results.* We present the results of our experiment in Tables 5.2 and 5.3. While composite hybrid post-quantum XML signatures are larger than purely post-quantum XML signatures, and also slower, the performance drop is not as drastic as going from classical to post-quantum signatures (see Section 5.2.1.1). For example, the document signed with the ECDSA (P-256) + ML-DSA (security level 2) composite signature is only 3–4% larger than the document signed with ML-DSA alone, because the classical signature is already small. On the other hand, the total time is about 1.4× the total time with ML-DSA alone, because ML-DSA already has fast operations. We see similar results for composite signatures with FN-DSA.

For each combination, the separate hybrid approach produces larger signed XML documents compared to the composite hybrid because it includes the extra `Signature` element with metadata. For ECDSA (P-256) + ML-DSA (security level 2), the increase in size (compared to the composite hybrid) is about 17%, and for ECDSA (P-256) + FN-DSA (security level 1) it is about 32%.

Performance is also affected because of the overhead of creating and verifying the extra `Signature` element. The separate hybrid approach with ECDSA (P-256) + ML-DSA (security level 2) is about 61% slower than the equivalent composite hybrid. For example, at security level 5, overhead becomes smaller for ECDSA + FN-DSA to about 37%.

### 5.2.2 Post-Quantum XML Encryption

In this section, we design and test purely post-quantum (Section 5.2.2.1) and hybrid post-quantum (Section 5.2.2.2) solutions for XML encryption. As mentioned in Section 5.1.3, symmetric encryption schemes can be made quantum-safe by doubling their key sizes (see Section 1.4.2 for

[4] Specifically, we achieve 2-non-separability since the second component, the classical signature, is not separable.

**Table 5.2:** Composite Hybrid Post-Quantum XML Signature Performance

| Algorithm | `Signature` Size | Document Size | Signing Time | Verification Time | Total Time |
|---|---|---|---|---|---|
| ECDSA (P-256) | 1563 | 2513 | 0.28 ± 0.05 | 0.51 ± 0.06 | 1.58 ± 0.12 |
| ECDSA (P-256) + ML-DSA (2) | 10173 (× 6.51) | 11123 (×4.43) | 1.09 ± 0.45 (×3.89) | 0.67 ± 0.12 (×1.31) | 3.52 ± 0.40 (×2.23) |
| ECDSA (P-256) + FN-DSA (1) | 4692 (× 3.00) | 5642 (×2.25) | 1.49 ± 0.17 (×5.32) | 0.52 ± 0.04 (×1.02) | 4.01 ± 0.25 (×2.54) |
| ECDSA (P-521) + ML-DSA (5) | 18344 (×11.74) | 19294 (×7.68) | 2.20 ± 0.47 (×7.86) | 2.74 ± 0.09 (×5.37) | 9.88 ± 0.33 (×6.25) |
| ECDSA (P-521) + FN-DSA (5) | 7907 (× 5.06) | 8857 (×3.52) | 3.46 ± 0.24 (×12.36) | 2.32 ± 0.11 (×4.53) | 11.53 ± 0.38 (×7.30) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. SHA2-512 was used as a reference hashing function. ECDSA with SHA2-256 for P-256, SHA2-512 for P-521. Included certificate chain length 1. Assertions not encrypted.

**Table 5.3:** Separate Hybrid Post-Quantum XML Signature Performance

| Algorithm | `Signature` Size | Document Size | Signing Time | Verification Time | Total Time |
|---|---|---|---|---|---|
| ECDSA (P-256) | 1563 | 2513 | 0.28 ± 0.05 | 0.51 ± 0.06 | 1.58 ± 0.12 |
| ECDSA (P-256) + ML-DSA (2) | 11755 (× 7.52) | 12978 (×5.16) | 1.66 ± 0.73 (×5.93) | 1.18 ± 0.16 (×2.31) | 5.67 ± 0.74 (×3.59) |
| ECDSA (P-256) + FN-DSA (1) | 6241 (× 3.99) | 7464 (×2.97) | 1.87 ± 0.23 (×6.68) | 0.97 ± 0.14 (×1.90) | 5.68 ± 0.41 (×3.59) |
| ECDSA (P-521) + FN-DSA (5) | 9459 (× 6.05) | 10682 (×4.25) | 3.95 ± 0.30 (×14.11) | 3.96 ± 0.19 (×7.76) | 15.83 ± 0.55 (×10.02) |
| ECDSA (P-521) + ML-DSA (5) | 19922 (×12.75) | 21145 (×8.41) | 4.10 ± 0.57 (×14.64) | 4.57 ± 0.24 (×8.96) | 17.35 ± 0.83 (×10.98) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. SHA2-512 was used as a reference hashing function. ECDSA with SHA2-256 for P-256, SHA2-512 for P-521. Included certificate chain length 1. Assertions not encrypted.

more information), while *public-key* encryption schemes need to be enhanced or replaced with post-quantum solutions; the latter will therefore be the focus of this section.

#### 5.2.2.1 Purely Post-Quantum XML Encryption

There are essentially two ways to make the classical PKE scheme in XML purely post-quantum: replace the classical PKE scheme with a post-quantum PKE scheme, or replace it with a post-quantum key encapsulation mechanism (KEM).

**Straightforward Solution.** Similar to purely post-quantum XML signatures (Section 5.2.1.1), this solution is fairly straightforward. We replace the classical PKE scheme with a post-quantum PKE scheme. Although NIST did not standardize public-key encryption schemes (PKEs), but rather key encapsulation mechanisms (KEMs), recall from Definition 12 that we can build a post-quantum PKE from a post-quantum KEM and a secure DEM. This KEM+DEM-based PKE is then used in the XML encryption procedures in the same manner as a classical PKE.

**Efficient Solution.** Recall from Section 5.1.2 that XML-based encryption is effectively a hybrid encryption scheme—a combination of a PKE with a symmetric cipher. In the straightforward solution we use a post-quantum PKE which internally is also a hybrid encryption scheme (KEM+DEM). Consequently, there is a redundancy—an extra symmetric cipher. It is conceptually cleaner to use a KEM directly in XML encryption, instead of using it to build a PKE. However, the current XML encryption standard [W3C13a] does not support the notion of a KEM, which is a fairly novel primitive. Therefore, although this solution is more "clean," we do not investigate and implement it further.

**Table 5.4:** Purely Post-Quantum XML Public-Key Encryption Performance

| Algorithm | Enc. `Assertion` Size | Document Size | Encryption Time | Decryption Time | Total Time |
|---|---|---|---|---|---|
| RSA (3072 bits) | 4 388 | 5 001 | 0.55 ± 0.21 | 10.46 ± 0.22 | 11.01 ± 0.35 |
| ML-KEM (1) | 5 231 (×1.19) | 5 832 (×1.17) | 0.25 ± 0.05 (×0.45) | 0.23 ± 0.05 (×0.02) | 0.48 ± 0.09 (×0.04) |
| ML-KEM (5) | 7 417 (×1.69) | 8 018 (×1.60) | 0.30 ± 0.04 (×0.56) | 0.30 ± 0.04 (×0.03) | 0.60 ± 0.06 (×0.05) |
| Cl. McEliece (1) | 355 929 (×81.11) | 356 530 (×71.29) | 2.51 ± 0.37 (×4.59) | 12.92 ± 0.57 (×1.23) | 15.43 ± 0.68 (×1.40) |
| BIKE (1) | 7 387 (×1.68) | 7 988 (×1.60) | 0.67 ± 0.05 (×1.22) | 21.50 ± 0.33 (×2.06) | 22.17 ± 0.34 (×2.01) |
| Cl. McEliece (5) | 1 415 005 (×322.47) | 1 415 606 (×283.06) | 9.60 ± 0.84 (×17.54) | 63.57 ± 0.84 (×6.08) | 73.17 ± 1.23 (×6.65) |
| BIKE (5) | 17 380 (×3.96) | 17 981 (×3.60) | 5.02 ± 0.09 (×9.19) | 136.35 ± 1.51 (×13.03) | 141.38 ± 1.54 (×12.84) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. RSA-OAEP with MGF1-SHA1 [JK03].

**Purely Post-Quantum XML Encryption Performance** This section presents results of our purely post-quantum XML encryption experiment.

*Experimental Setup.* The experimental setup is similar to the XML signature experiments in Section 5.2.1 except that we encrypt and decrypt the `Assertion` element within the `Response` SAML message (see Section 5.3). The result of the encryption process is the `EncryptedAssertion` element. The certificate containing the public key used for encryption is signed using RSA for both classical and post-quantum PKEs. We evaluate the *straightforward solution* from Section 5.2.2.1.

*Results.* We present the results of our experiment in Table 5.4.

ML-KEM is significantly faster (about 20×) than classical 3072-bit RSA-OAEP public-key encryption. The ML-KEM encryption still produces larger XML ciphertext than RSA because the internal KEM ciphertext and the KEM public key are larger (the public key is inside the `KeyInfo` element).

BIKE at security level 1 takes about twice as long as RSA, and the size of its XML ciphertext is similar to the size of ML-KEM's XML ciphertext at security level 1. BIKE at security level 5 is the slowest option we tested, taking about 13× as long as RSA; its XML ciphertext is about 2.3× the size of ML-KEM's XML ciphertext at security level 5.

The total size of Classic McEliece XML ciphertexts is several orders of magnitude larger than the others. However, it has the smallest (non-XML) ciphertexts of all post-quantum KEMs and also of RSA (see Table 2.1). The reason for this difference is that XML ciphertexts also contain the public keys, and Classic McEliece has large public keys. Therefore, if we removed the public key from the `KeyInfo` element, Classic McEliece would be the most bandwidth-efficient XML public-key encryption algorithm. In terms of speed, McEliece is about 40% slower than RSA at security level 1.

#### 5.2.2.2 Hybrid Post-Quantum XML Encryption

Our main idea for implementing hybrid post-quantum XML encryption is to use two nested layers of encryption, one classical and one post-quantum. This way, even if one of the layers is broken, the encrypted message will remain secret if the other layer remains secure. While this solution is the simplest to implement in our libraries, at the end of this section we explain more complicated approaches that can be more efficient in some applications.

**Table 5.5:** Hybrid Post-Quantum XML Public-Key Encryption Performance

| Algorithm | Enc. `Assertion` Size | Document Size | Encryption Time | Decryption Time | Total Time |
|---|---|---|---|---|---|
| RSA (3072 bits) | 4 388 | 5 001 | 0.55 ± 0.21 | 10.46 ± 0.22 | 11.01 ± 0.35 |
| + ML-KEM (1) | 10 255 (×2.34) | 10 856 (×2.17) | 0.83 ± 0.28 (×1.52) | 10.74 ± 0.67 (×1.03) | 11.57 ± 0.79 (×1.05) |
| + ML-KEM (5) | 12 441 (×2.84) | 13 042 (×2.61) | 0.79 ± 0.08 (×1.44) | 10.68 ± 0.28 (×1.02) | 11.47 ± 0.29 (×1.04) |
| + Cl. McEliece (1) | 360 953 (×82.26) | 361 554 (×72.30) | 3.28 ± 0.17 (×6.00) | 23.37 ± 0.19 (×2.23) | 26.65 ± 0.28 (×2.42) |
| + BIKE (1) | 12 411 (×2.83) | 13 012 (×2.60) | 1.14 ± 0.24 (×2.09) | 31.89 ± 0.64 (×3.05) | 33.04 ± 0.69 (×3.00) |
| + Cl. McEliece (5) | 1 420 029 (×323.62) | 1 420 630 (×284.07) | 11.02 ± 0.24 (×20.14) | 73.41 ± 0.86 (×7.02) | 84.43 ± 0.95 (×7.67) |
| + BIKE (5) | 22 404 (×5.11) | 23 005 (×4.60) | 5.46 ± 0.11 (×9.99) | 147.50 ± 1.54 (×14.10) | 152.96 ± 1.58 (×13.89) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. RSA-OAEP with MGF1-SHA1 [JK03].

To implement hybrid post-quantum encryption, we first encrypt the message using the classical public-key encryption scheme, and then encrypt the resulting ciphertext using the purely post-quantum public-key encryption scheme (Section 5.2.2.1).

Although not specifically intended for the post-quantum setting, such layered encryption is called *super-encryption* in the XML encryption standard [W3C13a]. We slightly improve the existing super-encryption technique as follows. In general, the decrypting party does not directly see whether the `EncryptedData` element is an instance of layered encryption or regular single-layer encryption. Therefore, we add metadata to the outer-layer `EncryptedData` to indicate that it is an outer layer. More specifically, to conform to the existing standard [W3C13a], we use the `MimeType` attribute of `EncryptedData` with a value such as `enc/multilayer`.

**Hybrid Post-Quantum XML Encryption Performance** This section presents the results of our hybrid post-quantum XML encryption experiment.

*Experimental Setup.* To evaluate the performance of our hybrid post-quantum XML public-key encryption, we used the same experimental setup as in the purely post-quantum case (Section 5.2.2.1). We combined the classical XML encryption algorithm 3072-bit RSA-OAEP [JK03] with ML-KEM, BIKE and Classic McEliece at security levels 1 and 5.

*Results.* We present the results of our experiment in Table 5.5.

We see that using hybrid post-quantum encryption with RSA and ML-KEM results in only a 4–5% slowdown compared to using plain RSA. The resulting document is more than twice as large.

Using RSA with BIKE at security level 1 is about 3× slower than using plain RSA, and the document size is about the same as using RSA with ML-KEM at security level 5. RSA with BIKE at level 5 is the slowest combination tested, taking almost 14× as long as plain RSA.

RSA with Classic McEliece at security level 1 takes about 2.4× the time of plain RSA, but due to the larger public keys, the resulting XML document is about 72× larger compared to using only RSA.

## 5.3 SAML Framework

In this section, we describe the basics of Security Assertion Markup Language (SAML) [OAS12]. SAML is an XML-based framework for exchanging security information, referred to as *assertions*, between various parties.

SAML messages are typically exchanged between two entities called the *Relying Party* (RP) and the *Asserting Party* (AP). In some scenarios, such as the Single Sign-On (SSO) protocol, there are additional participants called *users*.

SAML protocols are request-response protocols: the RP sends a SAML request indicating which assertions the RP wants to obtain from the AP, and the AP sends back a SAML response with the requested assertions, or some error message. For example, an assertion can contain a confirmation that a specific user, indicated in the request, is authorized to access some resource.

The SAML protocol that is the focus of our work is the Single Sign-On (SSO) protocol (Section 5.3.1).

### 5.3.1 SAML Single Sign-On

Single Sign-On (SSO) is an authentication method allowing users to log in to multiple services with one set of credentials. One of the most prominent realizations of SSO is the SAML SSO protocol. The protocol has three participants: the *Identity Provider* (IdP), the *Service Provider* (SP) and the *user*. In the SAML terminology mentioned above, IdP acts as the asserting party while SP is the relying party.

There are multiple variants of the SSO protocol based on the so-called *bindings*. Bindings specify how the protocol messages are transported between the participants. In our work, we focus on the HTTP POST binding. Our results are essentially binding-agnostic; therefore, they can be easily adapted to any protocol variant.

The protocol assumes that the SP and IdP have a pre-existing relationship—they have exchanged configuration details, such as trusted certificates and user identifiers. In our simplified description, we assume that SP and IdP trust each other's certificates.

The protocol utilizes two SAML messages: `AuthnRequest` (the request) and `Response` (the response). It can be summarized into these logical steps with references to the diagram Figure 5.3:

(1) User requests to log in at the SP using SSO.

(2-4) The SP creates the `AuthnRequest` and transports it to the IdP using the defined binding.

(5-6) The IdP processes the `AuthnRequest` and authenticates the user.

(7-9) In case the user successfully authenticates, the IdP creates the `Response` and sends it back to the SP.

(10-11) The SP processes the `Response` and issues the user a cookie that proves that the user is logged in.[5]

[5] This is just one of the many ways the SP can keep the user logged in. Authentication-state management is beyond the scope of the protocol.

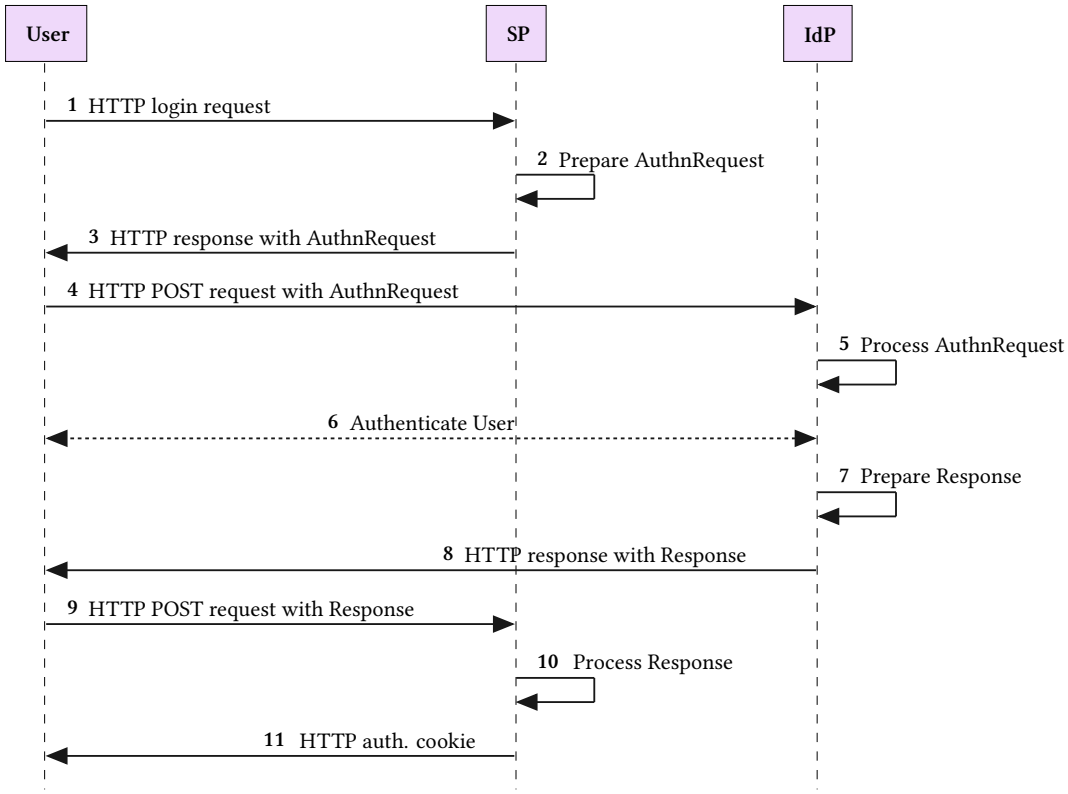



**Figure 5.3:** SAML SSO Protocol

Let us describe in detail the protocol messages. An example `AuthnRequest` message is shown in Listing 5.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:AuthnRequest
   AssertionConsumerServiceURL="https://serviceprovider.com/samlEndpoint"
   Destination="https://identityprovider.com/samlEndpoint"
   ID="SP_c27869a7-c83f-43ad-8a77-77ac2606a90e"
   ForceAuthn="true"
   IssueInstant="2023-02-24T14:49:08.830Z" (other attributes...)>
   <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://serviceprovider.com</saml2:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"><!-- enveloped signature --></ds:Signature>
   <saml2:Subject xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
      <saml2:NameID>john@example.com</saml2:NameID>
   </saml2:Subject>
</saml2p:AuthnRequest>
```

**Listing 5.4:** SAML AuthnRequest Example

`AuthnRequest` can have many attributes, but these are the essential ones:

- `Destination` indicates the endpoint which should process this request.
- `AssertionConsumerServiceURL` indicates the `Destination` attribute for the subsequent `Response` to this request.
- `ID` is a unique identifier of the SAML message.
- `IssueInstant` is the timestamp when this SAML message has been created.

- `ForceAuthn` states if the IdP is obligated to freshly establish the user's identity, i.e., making the user log in using his credentials (opposed to reusing a previous session information, e.g., from a cookie).

`AuthnRequest` can also have various child elements, but these three are relevant to our work:

- `Issuer` indicates the issuer of the SAML message.
- `Signature` contains the XML signature (Section 5.1.1) of the SAML message.
- `Subject` contains the information about the user that is requested to be authenticated.[6]

The SP specifies which type of information it wants to obtain from IdP, such as the user's email. The details depend on the policy between the SP and the IdP.

Listing 5.5 shows an example `Response` to the previous `AuthnRequest`.

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:Response
   Destination="https://serviceprovider.com/samlEndpoint"
   ID="IDP_d18ba520-3e82-4a82-9788-9293a063c97b"
   InResponseTo="SP_c27869a7-c83f-43ad-8a77-77ac2606a90e"
   IssueInstant="2023-02-24T14:49:46.014Z" (other attributes...)>
   <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://identityprovider.com</saml2:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"><!-- enveloped signature --></ds:Signature>
   <saml2p:Status>
      <saml2p:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
   </saml2p:Status>
   <saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0">
      <saml2:Issuer>https://identityprovider.com</saml2:Issuer>
      <saml2:Subject><!-- data about the authenticated subject --></saml2:Subject>
   </saml2:Assertion>
</saml2p:Response>
```

**Listing 5.5:** SAML Response Example

Similar to the `AuthnRequest`, the `Response` has attributes `ID`, `Destination` and `IssueInstant`. As a response, this message has the attribute `InResponseTo` indicating the `ID` of the corresponding `AuthnRequest`. It also has the children `Issuer` and `Signature` serving the same purpose as for the request.

`Response` has response-specific child elements `Status` and `Assertion`. `Status` indicates the result of the authentication. In Listing 5.5, the authentication was successful.

`Assertion` contains the information about the authenticated user. SAML supports an encrypted version of the `Assertion`, the `EncryptedAssertion`. The encryption follows the XML encryption standard presented in Section 5.1.2.

### 5.3.2 SAML SSO Security

The security mechanisms in SAML SSO fall into two categories: those that depend on the binding and those that do not.

[6] This element is not required for the SSO to work because the user authenticates with his credentials at the IdP and the SP might not know anything about the user beforehand.

#### 5.3.2.1 Binding-Dependent Mechanisms

These mechanisms vary depending on how the binding is implemented. For example, in the HTTP POST binding, HTTP connections are secured using TLS (Chapter 4), while all SAML messages pass through the user, who can read and modify them. In contrast, the HTTP Artifact binding does not route SAML messages through the user.

The main binding-specific mechanism is TLS, which establishes secure channels between two peers. Although TLS is not required by the SAML standard, we assume throughout this paper that all HTTP connections are secured with TLS. In the scenario shown in Figure 5.3, there are two TLS channels: user–SP and user–IdP.

#### 5.3.2.2 Binding-Independent Mechanisms

While TLS prevents external parties from reading or modifying messages, it is also necessary to prevent attacks from the peers themselves, such as the user. This is where binding-independent mechanisms come into play. These mechanisms are twofold: XML signatures (Section 5.1.1) provide integrity and authenticity of SAML messages, and XML encryption (Section 5.1.2) is used to protect some sensitive data in the SAML SSO protocol.

Unless otherwise noted, we assume that SAML messages are signed by their creators (even though this is not explicitly required by the standard). We also assume that IdP responses are partially encrypted—the `Assertion` element within `Response` is encrypted, and the result of the XML encryption is placed inside `EncryptedAssertion`. Additionally, we assume that the `Signature` element is a child of `AuthnRequest`, or `Response`.

Recall that there is a setup phase where the SP and IdP agree on configuration details, in particular to establish trust. This can be done in a number of ways. They can exchange long-term signing public keys, or they can use a PKI and establish trust anchor(s). In our work, we assume that the SP and the IdP share a common trust anchor, i.e., they use certificates signed by a CA that is trusted by both parties.

Recall from Section 5.1 that XML signatures and XML encryption have the `KeyInfo` element, which contains information about the signing/encryption key. We assume that `KeyInfo` always contains an X.509 public-key certificate (Section 3.1) signed by the trusted CA or, in the case of XML public-key encryption, an encrypted symmetric key.

In summary, XML signatures are the main security mechanism used in the SSO protocol, since the purpose of the protocol is to prove to the SP that the user is who they claim to be, and the `Response` signed by the IdP provides that proof.

### 5.3.3 Quantum Threat Analysis

Based on Section 5.3.2, we identify three security mechanisms which make SAML SSO vulnerable to quantum adversaries: XML signatures, XML public-key encryption and TLS. We discuss the impact on the SSO protocol if a quantum adversary breaks any of these security mechanisms.

An adversary attacking the SSO protocol can have multiple goals, for example:

(1) It wants to learn the content of the SAML messages.

(2) It wants to impersonate a user to the SP.

(3) It wants to impersonate the SP to the IdP (or vice versa).

(4) It wants to impersonate a user to the IdP.

With the current SAML SSO protocol in use, a quantum adversary can achieve goals (1), (2) and (3), while (4) is dependent on the authentication method used by the IdP, which is outside the scope of the SAML SSO protocol:

1) The adversary can record the traffic and break the confidentiality of TLS and/or XML public-key encryption. This attack is the "store now, decrypt later" attack (Section 1.4.3).

2) The adversary receives the `AuthnRequest` from the SP and then forges a `Response` from the IdP claiming to be the targeted user. This requires forging an XML signature where the signing private key can be pre-computed, e.g., from the IdP's public-key certificate.

3) The adversary creates an `AuthnRequest` with SP's identifiers and forges the SP's signature and sends it to the IdP. Similarly, the adversary can create a `Response` and forge the IdP's signature.

In this way, a quantum attacker can, for example, access sensitive private data that the service provider is supposed to make available only to the designated user.

We can see that upgrading just one security mechanism to be post-quantum is not enough to prevent the attacks: while post-quantum TLS prevents non-participants from learning the contents of messages, it does not prevent impersonation, and while upgrading to post-quantum XML signatures prevents impersonation, it does not provide confidentiality.

## 5.4 Post-Quantum SAML SSO

In this section, we describe how to integrate the post-quantum cryptographic building blocks introduced in the previous sections to make the SAML Single Sign-On (SSO) protocol (Section 5.3) quantum-safe. We present two post-quantum variants: a purely post-quantum one (Section 5.4.1) and a hybrid post-quantum one (Section 5.4.2). As with our post-quantum XML solutions, we implemented these constructions and conducted performance experiments.

**Post-Quantum PKI and TLS in SAML.** Recall from Section 5.3 that SAML SSO relies on a PKI.[7] In our solutions, we assume a post-quantum PKI, which we describe in Chapter 3.

Another essential component of a secure SAML SSO deployment is TLS. Recall from Section 5.3 that if we want to protect SAML messages against eavesdroppers, they need to be transported using a secure channel. The most common channel is the TLS protocol. A post-quantum SAML SSO deployment therefore also needs post-quantum TLS. Post-quantum TLS solutions are discussed in Chapter 4.

### 5.4.1 Purely Post-Quantum SAML SSO

We obtain a purely post-quantum SAML SSO by replacing the classical XML signatures and encryption with purely post-quantum XML signatures and encryption (Sections 5.2.1.1 and 5.2.2.1).

[7] Technically, SAML SSO can operate without certificates, but such configuration is impractical for most use cases.

**Table 5.6**: Purely Post-Quantum SAML SSO Performance

| Signature Alg. | PKE Alg. | Bandwidth | SP Time | IdP Time | Total Time |
|---|---|---|---|---|---|
| ECDSA (P-256) | RSA (3072 bits) | 9 481 | 11.38 ± 0.66 | 1.65 ± 0.38 | 13.03 ± 0.84 |
| ML-DSA (2) | ML-KEM (1) | 33 733 (×3.56) | 1.38 ± 0.10 (×0.12) | 1.80 ± 0.42 (×1.09) | 3.18 ± 0.43 (×0.24) |
| FN-DSA (1) | ML-KEM (1) | 17 924 (×1.89) | 1.89 ± 0.20 (×0.17) | 2.07 ± 0.20 (×1.25) | 3.95 ± 0.30 (×0.30) |
| ML-DSA (2) | BIKE (1) | 36 890 (×3.89) | 22.60 ± 0.52 (×1.99) | 2.30 ± 0.43 (×1.39) | 24.90 ± 0.69 (×1.91) |
| FN-DSA (1) | BIKE (1) | 21 065 (×2.22) | 23.20 ± 0.63 (×2.04) | 2.56 ± 0.19 (×1.55) | 25.77 ± 0.68 (×1.98) |
| ML-DSA (2) | Cl. McEliece (1) | 736 098 (×77.64) | 22.05 ± 0.29 (×1.94) | 12.27 ± 0.48 (×7.44) | 34.32 ± 0.60 (×2.63) |
| FN-DSA (1) | Cl. McEliece (1) | 720 265 (×75.97) | 22.61 ± 0.26 (×1.99) | 13.40 ± 0.19 (×8.12) | 36.02 ± 0.37 (×2.76) |
| SLH-DSA (1) | ML-KEM (1) | 75 203 (×7.93) | 1027.41 ± 6.50 (×90.28) | 1028.46 ± 6.58 (×623.31) | 2055.87 ± 12.17 (×157.78) |
| SLH-DSA (1) | BIKE (1) | 78 360 (×8.26) | 1048.58 ± 6.64 (×92.14) | 1028.85 ± 6.56 (×623.55) | 2077.43 ± 12.27 (×159.43) |
| SLH-DSA (1) | Cl. McEliece (1) | 777 560 (×82.01) | 1048.33 ± 6.84 (×92.12) | 1039.09 ± 6.68 (×629.75) | 2087.42 ± 12.58 (×160.20) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. SHA2-512 was used as a reference hashing function. ECDSA with SHA2-256, SLH-DSA parameters: SHA256, 128s/256s, simple. RSA-OAEP with MGF1-SHA1 [JK03].

#### 5.4.1.1 Purely Post-Quantum SAML SSO Performance

We evaluated the performance of purely post-quantum SAML SSO instantiated with various algorithms.

**Experimental Setup.** The basic setup is analogous to our post-quantum XML experiments presented in Section 5.2. The SP creates an `AuthnRequest` with a KEM certificate placed inside `Extensions` and signs the `AuthnRequest` using the purely post-quantum XML signature. The IdP verifies the signature of the `AuthnRequest` and crafts a corresponding `Response` with an `EncryptedAssertion` encrypted using the public key from the KEM certificate found in the corresponding `AuthnRequest`. Finally, the IdP signs the `Response`. The SP verifies the signature of the `Response` and decrypts the `EncryptedAssertion` using its corresponding KEM private key. Signing and encryption certificates are verified.

**Results.** In Table 5.6 we present the total bandwidth of the protocol, i.e., the sum of the sizes of `AuthnRequest` and `Response` messages, the processing time on the SP's side and the IdP's side, and the total processing time. Due to the large number of possible algorithm combinations, we only provide results for the lowest-security-level parameters. We use the classical combination of ECDSA signature with RSA PKE as the baseline.

We see that certain combinations of post-quantum algorithms provide better performance compared to the baseline: ML-DSA with ML-KEM is four times faster and FN-DSA with ML-KEM is more than three times faster. Unsurprisingly, all post-quantum algorithm combinations result in higher bandwidth compared to the baseline. The most bandwidth-efficient post-quantum combinations are FN-DSA with ML-KEM or BIKE; however, these combinations still consume about 2× the baseline's bandwidth. The worst-performing combination is the security-conservative pair SLH-DSA with Classic McEliece, with 82× the baseline bandwidth and 160× the baseline total time.

### 5.4.2 Hybrid Post-Quantum SAML SSO

We present two approaches to implementing hybrid post-quantum SAML SSO: the first relies on composite hybrid post-quantum XML signatures (Section 5.2.1.2) and is *not* backward compatible,

and the second uses separate hybrid post-quantum XML signatures (Section 5.2.1.2) and is backward compatible.

**Backward Compatibility in SAML.** We say that a post-quantum solution for SAML SSO is backward-compatible if the SP/IdP can construct `AuthnRequest`/`Response` such that its authenticity and integrity achieve hybrid post-quantum security (Definition 22), while it can still be processed by classical-only verifiers.

#### 5.4.2.1 Non-Backward-Compatible Hybrid

The simplest way to implement hybrid post-quantum SAML SSO is to use composite hybrid post-quantum XML signatures (Section 5.2.1.2) to sign SAML messages and hybrid post-quantum XML public-key encryption (Section 5.2.2.2) to encrypt them. This approach does not make any changes to the standard SAML message structure since the message contains one `Signature` element containing the composite hybrid post-quantum signature (Definition 24).

Hybrid post-quantum XML encryption can also be implemented without modifying the SAML layer as only the underlying XML encryption library needs to be modified to process the layered encryption—decrypt twice if the `EncryptedData` has the `MimeType` attribute `enc/multilayer`.

Since neither composite hybrid XML signatures nor hybrid post-quantum XML encryption are backward compatible, this hybrid solution is also *not* backward compatible. For example, an IdP cannot issue hybrid post-quantum SAML responses that are verifiable by classical-only SPs that have not migrated their systems.

#### 5.4.2.2 Backward-Compatible Hybrid

We present our solution to achieve backward compatibility in hybrid post-quantum SAML SSO. We needed to find a reasonable trade-off between requirements that are in part mutually exclusive. The reason is that hybrid post-quantum XML public-key encryption is incompatible with backward compatibility—classical-only parties cannot process the nested hybrid post-quantum XML ciphertexts.

Consequently, we need to compromise on post-quantum secrecy of SAML assertions—that is what is encrypted in the SAML SSO protocol—and use the classical XML encryption for this purpose. Note that we still use hybrid post-quantum TLS to secure the network channels from non-protocol participants.

To achieve backward compatible hybrid post-quantum authentication in SAML, we use the separate hybrid XML signatures presented in Section 5.2.1.2. Recall that the separate hybrid post-quantum XML signature is composed of two `Signature` elements, where the post-quantum `Signature` contains an additional transformation that removes the classical `Signature` during the verification process.

Specifically, a SAML SSO `AuthnRequest` is signed as follows using the separate hybrid post-quantum XML signature (analogous for `Response`):

1) The `AuthnRequest` is signed by the (purely) post-quantum signature. The resulting `Signature` is placed inside the `Extensions` SAML-defined element, which is a child of `AuthnRequest`. The `Signature` element contains the additional `Transform` to remove the first `Signature`

that is a child of the root element from the digest calculation. The transform is an XPath filter [W3C02] in the following form:[8]

```
<ds:Transform Algorithm="http://www.w3.org/2002/06/xmldsig-filter2">
   <dsig-xpath:XPath Filter="subtract">/saml2p:*/ds:Signature[1]</dsig-xpath:XPath>
</ds:Transform>
```

2) The `AuthnRequest` (with the post-quantum signature inside `Extensions`) is signed using the classical algorithm and the resulting classical `Signature` is placed as a child of `AuthnRequest`. This (classical) `Signature` element contains only the standard transforms: the canonicalization and the enveloped signature transforms.

Listing 5.6 is an example of a backward-compatible hybrid post-quantum SAML SSO `AuthnRequest` message.

```
<!-- namespace attributes omitted -->
<saml2p:AuthnRequest ID="SP_50f55bc7-4bea-4f0b-a9b1-84fc881212a7" <!-- other attributes -->>
   <saml2:Issuer>https://sp.example.com</saml2:Issuer>
   <ds:Signature>
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256"/>
         <ds:Reference URI="#SP_50f55bc7-4bea-4f0b-a9b1-84fc881212a7">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
            <ds:DigestValue><!-- base64-encoded digest --></ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue><!-- base64-encoded signature --></ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate><!-- base64-encoded ECDSA certificate --></ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
   </ds:Signature>
   <saml2p:Extensions>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate><!-- base64-encoded encryption certificate --></ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Signature>
         <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2023/02/xmldsig-pqc#dilithium"/>
            <ds:Reference URI="#SP_50f55bc7-4bea-4f0b-a9b1-84fc881212a7">
               <ds:Transforms>
                  <ds:Transform Algorithm="http://www.w3.org/2002/06/xmldsig-filter2">
                     <dsig-xpath:XPath Filter="subtract">/saml2p:*/ds:Signature[1]</dsig-xpath:XPath>
                  </ds:Transform>
                  <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                  <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
               </ds:Transforms>
               <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
               <ds:DigestValue><!-- base64-encoded digest --></ds:DigestValue>
            </ds:Reference>
         </ds:SignedInfo>
         <ds:SignatureValue><!-- base64-encoded signature --></ds:SignatureValue>
```

[8] Recall from Section 5.3 that we assume the classical `Signature` is a child of `AuthnRequest` or `Response`. If the SAML implementation places the `Signature` at a different location, the transform needs to be modified accordingly.

```
        <ds:KeyInfo>
           <ds:X509Data>
              <ds:X509Certificate><!-- base64-encoded Dilithium certificate --></ds:X509Certificate>
           </ds:X509Data>
        </ds:KeyInfo>
     </ds:Signature>
  </saml2p:Extensions>
  <saml2:Subject>
     <saml2:NameID>subject@example.com</saml2:NameID>
  </saml2:Subject>
</saml2p:AuthnRequest>
```

**Listing 5.6:** Example of a backward-compatible hybrid post-quantum `AuthnRequest`.

A classical-only SP/IdP can verify the resulting SAML message as follows. It verifies the classical signature as usual, since it is compliant with the SAML standard, and ignores the post-quantum `Signature` inside `Extensions`, as the standard does not impose any restrictions on the content inside `Extensions`.

A hybrid post-quantum verifier first performs classical verification and aborts if the classical signature is invalid. Then the verifier runs the verification procedure for the `Signature` inside the `Extensions`. This second verification procedure must be modified to process the additional `Transform`, since standard-compliant SAML verifiers should only accept the standard transforms (and not the XPath Filter).[9] The verifier accepts the hybrid post-quantum SAML message if and only if both signatures are valid.

**Non-separability.** Recall from Section 5.2.1.2 that the separate hybrid post-quantum XML signature is 2-non-separable, which guarantees that an attacker cannot extract the classical `Signature` and use it as a standalone signature of `AuthnRequest`.

The attacker can extract the post-quantum `Signature` and try to use it as a standalone post-quantum XML signature of the `AuthnRequest`. However, this signature can be detected as "extracted from a hybrid XML signature" because it is a signature over the non-standard extra `Transform` element. Therefore, a standard-compliant non-hybrid verifier will reject this separated XML signature.

#### 5.4.2.3 Hybrid Post-Quantum SAML SSO Performance

As with other solutions presented in this chapter, we conducted an experiment to measure performance of the two proposed hybrid post-quantum SAML SSO variants.

**Experimental Setup.** To compare the performance between purely and non-backward-compatible hybrid post-quantum SAML SSO, we performed essentially the same experiment as in Section 5.4.1. We sign a SAML message using a composite hybrid signature and encrypt the assertion using the hybrid XML encryption (Section 5.2.2.2). The classical encryption (RSA) certificate is signed using the classical component of the composite signature (ECDSA), while the post-quantum KEM certificate is signed with the post-quantum component (ML-DSA, or FN-DSA). We chose this configuration because it allows a fair comparison with the backward-compatible solution.

[9] The SAML standard limits the possible transforms as arbitrary transforms pose a security threat as illustrated by [Som+12].

**Table 5.7:** Non-Backward-Compatible Hybrid Post-Quantum SAML SSO Performance

| Signature Alg. | PKE Alg. | Bandwidth | SP Time | IdP Time | Total Time |
|---|---|---|---|---|---|
| ECDSA (P-256) | RSA (3072 bits) | 9 481 | 11.38 ± 0.66 | 1.65 ± 0.38 | 13.03 ± 0.84 |
| + ML-DSA (2) | + ML-KEM (1) | 39 888 (× 4.21) | 12.26 ± 0.56 (×1.08) | 3.05 ± 0.63 (×1.85) | 15.32 ± 0.97 (×1.18) |
| + FN-DSA (1) | + ML-KEM (1) | 24 114 (× 2.54) | 12.79 ± 0.25 (×1.12) | 3.23 ± 0.24 (×1.96) | 16.02 ± 0.38 (×1.23) |
| + ML-DSA (2) | + BIKE (1) | 43 053 (× 4.54) | 33.58 ± 0.79 (×2.95) | 3.43 ± 0.43 (×2.08) | 37.00 ± 0.92 (×2.84) |
| + FN-DSA (1) | + BIKE (1) | 27 279 (× 2.88) | 34.13 ± 0.65 (×3.00) | 3.74 ± 0.25 (×2.27) | 37.87 ± 0.73 (×2.91) |
| + ML-DSA (2) | + Cl. McEliece (1) | 742 253 (× 78.29) | 33.24 ± 0.60 (×2.92) | 13.37 ± 0.50 (×8.10) | 46.60 ± 0.85 (×3.58) |
| + FN-DSA (1) | + Cl. McEliece (1) | 726 505 (× 76.63) | 33.78 ± 0.61 (×2.97) | 14.70 ± 0.48 (×8.91) | 48.48 ± 0.84 (×3.72) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. SHA2-512 was used as a reference hashing function. ECDSA with SHA2-256, RSA-OAEP with MGF1-SHA1 [JK03].

**Table 5.8:** Backward-Compatible Hybrid Post-Quantum SAML SSO Performance

| Signature Alg. | PKE Alg. | Bandwidth | SP Time | IdP Time | Total Time |
|---|---|---|---|---|---|
| ECDSA (P-256) | RSA (3072 bits) | 9 481 | 11.38 ± 0.66 | 1.65 ± 0.38 | 13.03 ± 0.84 |
| + ML-DSA (2) | + ML-KEM (1) | 43 559 (× 4.59) | 13.68 ± 0.50 (×1.20) | 4.21 ± 0.66 (×2.55) | 17.89 ± 1.04 (×1.37) |
| + FN-DSA (1) | + ML-KEM (1) | 27 718 (× 2.92) | 13.74 ± 0.34 (×1.21) | 4.18 ± 0.31 (×2.53) | 17.92 ± 0.49 (×1.38) |
| + ML-DSA (2) | + BIKE (1) | 46 716 (× 4.93) | 34.83 ± 0.76 (×3.06) | 4.56 ± 0.46 (×2.76) | 39.39 ± 0.93 (×3.02) |
| + FN-DSA (1) | + BIKE (1) | 30 891 (× 3.26) | 35.23 ± 0.68 (×3.10) | 4.77 ± 0.30 (×2.89) | 40.01 ± 0.80 (×3.07) |
| + ML-DSA (2) | + Cl. McEliece (1) | 745 932 (× 78.68) | 40.75 ± 0.45 (×3.58) | 20.93 ± 0.60 (×12.68) | 61.68 ± 0.86 (×4.73) |
| + FN-DSA (1) | + Cl. McEliece (1) | 730 099 (× 77.01) | 41.02 ± 0.69 (×3.60) | 21.86 ± 0.54 (×13.25) | 62.88 ± 0.99 (×4.83) |

Sizes are in bytes, time is in ms, value after ± is the standard deviation. (n) refers to the NIST security level. SHA2-512 was used as a reference hashing function. ECDSA with SHA2-256, RSA-OAEP with MGF1-SHA1 [JK03].

For the backward-compatible separate hybrid solution (Section 5.4.2.2), we use the separate hybrid XML signatures (Section 5.2.1.2) and the hybrid XML public-key encryption (Section 5.2.2.2) for a fair comparison with the non-backward-compatible solution. However, as mentioned before, using the hybrid XML PKE makes the protocol non-backward compatible.

**Results.** The results for the non-backward-compatible hybrid post-quantum SAML SSO are presented in Table 5.7. The results show that using hybrid post-quantum SAML SSO with ECDSA+ML-DSA or ECDSA+FN-DSA for signing, and RSA+ML-KEM for public-key encryption, is only about 20% slower overall. Bandwidth increases 4.2× for the ML-DSA variant and 2.54× for the FN-DSA variant. Using BIKE instead of ML-KEM results in about the same bandwidth, but the slowdown is almost 200%. Using Classic McEliece results in a bandwidth that is almost 80× the bandwidth of the non-hybrid classical solution and in a roughly 300% slowdown.

The results for the "backward-compatible" hybrid are presented in Table 5.8. Let us describe the performance differences between the backward-compatible versions and the non-backward-compatible versions that result from the different hybrid post-quantum XML signatures used. For example, we can see that the backward-compatible solution is 17% slower compared to the non-backward-compatible solution for the combination of ML-DSA with ML-KEM, and 12% slower for the combination of FN-DSA with ML-KEM. If BIKE is used instead of ML-KEM, the difference is 6%, and combinations with Classic McEliece are about 30% slower. The messages in the backward-compatible solution are also larger since they contain two `Signature` elements instead of one. As a result, for example, the combination of ML-DSA and ML-KEM uses 9% more bandwidth.

## 5.5 Conclusion

The topic of this chapter was the migration of the SAML Single Sign-On (SSO) protocol to post-quantum cryptography. Since SAML is an XML-based framework, the crucial element of SAML migration is the migration of XML encryption and XML signatures to post-quantum cryptography.

The first part of this chapter was dedicated to the study of post-quantum XML encryption and signatures. We have proposed both purely post-quantum and hybrid post-quantum approaches to XML encryption and signatures. We have shown that purely post-quantum solutions are straightforward in terms of implementation thanks to the crypto-agility of the standards. On the other hand, hybrid post-quantum approaches, which are desired because of their robust security, regulatory compliance, and potential backward compatibility, are not so straightforward. We have demonstrated that hybrid post-quantum XML signatures are possible in two variants, which differ in backward compatibility. In contrast, as backward compatibility of hybrid post-quantum (XML) encryption is inherently impossible, we have provided a straightforward but not backward-compatible method for hybrid post-quantum XML encryption.

The second part was dedicated to the SAML SSO protocol itself. We have shown that by migrating SAML's XML-based security components to post-quantum cryptography, achieving post-quantum SAML SSO is straightforward in both purely and hybrid post-quantum configurations. Notably, we designed a backward-compatible hybrid post-quantum SAML SSO solution that allows the IdP to issue hybrid post-quantum-authenticated responses while remaining compatible with classical-only verifiers.

To test the performance of our solutions, we have developed a proof-of-concept implementation of post-quantum XML signatures, encryption and SAML SSO using established libraries. Our results show that some purely post-quantum configurations are faster than existing classical setups—mainly due to ML-KEM's superior performance compared to RSA. Moreover, we observed that there is little performance impact from using backward-compatible hybrid post-quantum XML signatures compared to the non-backward-compatible variant. Consequently, there is little reason not to use the backward-compatible variant. On the other hand, as already stated, there cannot be a backward-compatible hybrid post-quantum SAML SSO setup. Ultimately, the implementor will have to decide between being backward compatible and having post-quantum-encrypted SAML assertions—the lack of which leaves the data vulnerable to "store now, decrypt later" attacks.

# Chapter 6

# Post-Quantum Advanced Electronic Signatures


**6.1 Advanced Electronic Signatures . . . 156**
6.1.1 AdES Concepts . . . 156
6.1.2 Comparison of AdES Signature Formats . . . 161
6.1.3 XML Advanced Electronic Signatures (XAdES) . . . 161
6.1.4 CMS Advanced Electronic Signatures (CAdES) . . . 164
6.1.5 PDF Advanced Electronic Signatures (PAdES) . . . 170
6.1.6 JSON Advanced Electronic Signatures (JAdES) . . . 175
6.1.7 Related Topics . . . 181
**6.2 Purely Post-Quantum Advanced Electronic Signatures . . . 182**
6.2.1 Example Purely Post-Quantum XAdES B-LTA . . . 182
6.2.2 Example Purely Post-Quantum JAdES B-LT . . . 183
**6.3 Hybrid Post-Quantum Advanced Electronic Signatures . . . 185**
6.3.1 Composite Hybrid Post-Quantum AdES . . . 185
6.3.2 Backward-Compatible Hybrid Post-Quantum AdES . . . 186
**6.4 Performance of Post-Quantum AdES . . . 191**
6.4.1 Methodology . . . 191
6.4.2 Results . . . 192
**6.5 Conclusion . . . 196**


The eIDAS regulation [Eur14] defines an *Advanced Electronic Signature* (AdES), which is a term for a digital signature compliant with the regulation. There are four standards for AdES: XML Advanced Electronic Signature (XAdES), CMS Advanced Electronic Signature (CAdES), PDF Advanced Electronic Signature (PAdES) and JSON Advanced Electronic Signature (JAdES).

Under the hood, AdES signatures use cryptographic signatures such as RSA or ECDSA, whose security depends on the hardness of the integer factorization or the discrete logarithm problem, respectively. As discussed in Section 1.4, quantum computers can solve these problems easily, i.e., in polynomial time. Consequently, quantum computers render these cryptographic signatures insecure.

This chapter begins with an overview of AdES signatures in Section 6.1, where we analyze their security and discuss the threat of quantum computers. Subsequently, we dedicate Section 6.2 to describing purely post-quantum AdES signatures. In Section 6.3, we discuss the notion of hybrid post-quantum AdES signatures. Finally, in Section 6.4, we present our performance results of various post-quantum AdES variants.

## 6.1 Advanced Electronic Signatures

We begin by introducing the general structure of an AdES signature in Section 6.1.1 and present its so-called signature levels. We analyze their security with respect to quantum adversaries. Following that, we dive into the actual signature formats and further analyze their security to encompass the formats' specificities. We present a brief comparison of the signature formats in Section 6.1.2. Then, we go over each signature format: XAdES in Section 6.1.3, CAdES in Section 6.1.4, PAdES in Section 6.1.5 and JAdES in Section 6.1.6. At the end, we briefly touch upon topics related to AdES, such as ERS and ASiC in Section 6.1.7.

### 6.1.1 AdES Concepts

This section presents Advanced Electronic Signatures on an abstract level. We follow the ETSI standard "Procedures for Creation and Validation of AdES Digital Signatures" [ETS16a] as it serves as a common reference for the AdES formats discussed in the subsequent sections.

Note that we do not describe the detailed procedures for creation and validation, as this is out of scope because these procedures do not depend on specific cryptographic algorithms, which could be considered vulnerable to quantum adversaries.

#### 6.1.1.1 AdES Signature Levels

Digital signatures that qualify as AdES differ from regular "plain" digital signatures in the inclusion of some additional data. There exist four "levels" of AdES signatures. These levels can be ordered based on the amount of included data. Also, it is possible to augment an existing signature into a higher level signature with the inclusion of more data.

The additional data included in an AdES signature is of two types:

- *Signed attributes*: These attributes are also signed, i.e., protected, during the signing process together with the signer's document,
- *Unsigned attributes*: These attributes are not protected by the signature, but they are still part of the AdES signature.

The signature levels are defined by the signed and unsigned attributes included in the signature. The levels are:

1) **Basic Signature** (Figure 6.1): This is the base level signature. Compared to a plain digital signature, the basic signature must contain the signing certificate as a signed attribute. The property of a basic signature is that it can be validated as long as the relevant certificates are valid (not expired or revoked).

2) **Signature with Time** (Figure 6.2): This level augments the basic level with the addition of a signature timestamp. The timestamp is an *unsigned* property serving as a proof that the signature existed at a given point in time. The included timestamp allows the signature to be validated even after, e.g., the expiration of the signing certificate.

3) **Signature with Long-Term Validation Material** (Figure 6.3): This level augments the signature with time with the addition of complete certificate and revocation data on the signature and the timestamp(s). The included data is unsigned. This allows the AdES signature to be verified even when data required for validation is no longer available, e.g., when the Certificate Revocation List (CRL) tied to the timestamp certificate is no longer available.

4) **Signature providing Long-Term Availability and Integrity of Validation Material** (Figure 6.4): This level augments the signature with long-term validation material with the addition of an *archival* timestamp. As opposed to the "regular" timestamp(s) from the Signature with Time level, the input into the archival timestamp is the whole AdES signature. This allows for the continuous augmentation of the AdES signature in cases where, for example, revocation data expires or cryptographic algorithms become weak.

Let us illustrate the archival timestamp mechanism on an example. Imagine that a signer has an existing AdES signature that is protected by a 2048-bit RSA. According to SOG-IS [SOG23], 2048-bit RSA should not be used after 2025, but the signer wants the AdES signature to remain valid and secure long after that. So, the signer can augment, e.g., in 2024, the signature with the addition of the archival timestamp, which will be secured by a 4096-bit RSA. A future validator will know that, in 2024, it was considered impossible to forge a 2048-bit RSA signature and the archival timestamp proves that the signature existed in 2024. Therefore, the signature will be considered valid. Eventually, when 4096-bit RSA becomes weak, the signer can create another archival timestamp over the signature with a new algorithm.

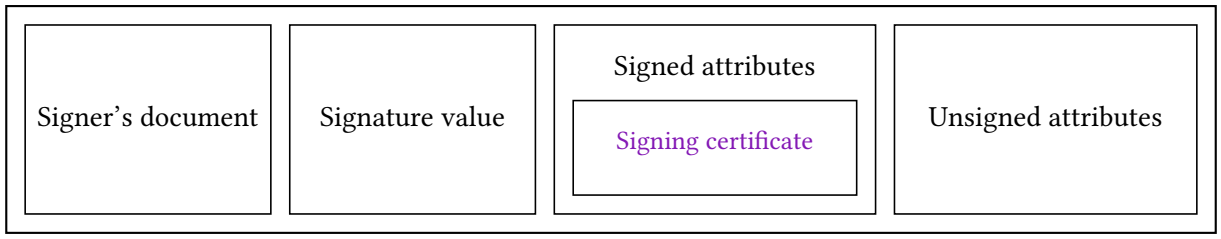



**Figure 6.1:** AdES Basic Signature Structure

We note that there may be additional requirements for each level depending on the format, e.g., a XAdES basic signature (XAdES-B-B level) must also contain the signing time and a description of the signed data. We discuss these requirements in the subsequent sections: XAdES (Section 6.1.3), CAdES (Section 6.1.4) and PAdES (Section 6.1.5).

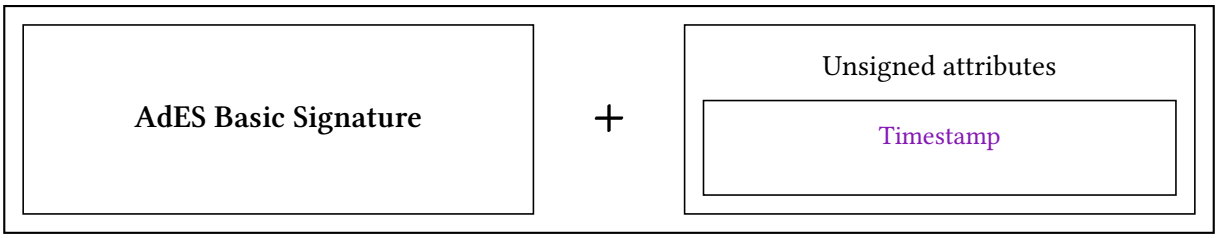



**Figure 6.2:** AdES Signature with Time Structure

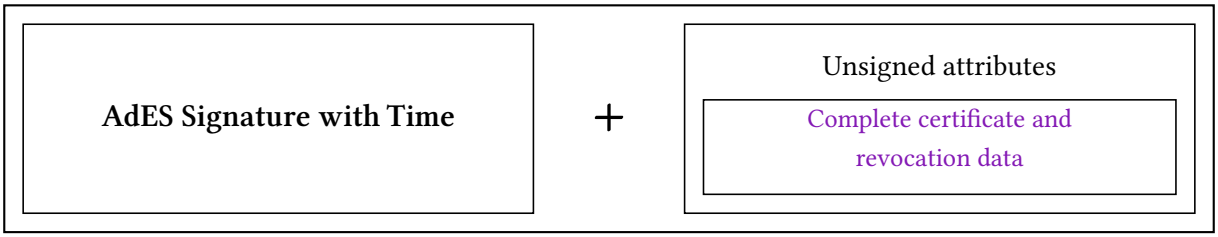



**Figure 6.3:** AdES Signature with Long-Term Validation Material Structure

In the format standards (XAdES etc.), different terms are used to describe the following signature levels:[1]

- *Basic Signature* is the **B-B** level.
- *Signature with Time* is the **B-T** level.
- *Signature with Long-Term Validation Material* is the **B-LT** level.
- *Signature providing Long-Term Availability and Integrity of Validation Material* is the **B-LTA** level.

#### 6.1.1.2 Security Properties

Let us look at the security mechanisms used within AdES signatures. Regarding cryptographic primitives, AdES signatures utilize signature schemes and hash functions.

Ascending into a higher level of abstraction, AdES signatures contain X.509 certificates (Section 3.1.3), timestamps (Section 3.1.6), Certificate Revocation Lists (CRLs) (Section 3.1.4.1) or Online Certificate Status Protocol (OCSP) responses (Section 3.1.4.2). These higher level concepts are components of a PKI which is discussed in Chapter 3.

Consequently, the security of AdES reduces to the security of the underlying signature schemes and hash functions. Note that there may also be multiple signature algorithms and hash functions used inside a single AdES signature. For example, in an AdES Signature providing Long-Term Availability and Integrity of Validation Material, there is:

- the signature computed over the signer's document and signed attributes,

[1] This is a bit imprecise since the B-∗ notation refers to *baseline* levels of signatures and for each format there exist also *extended* levels (E-∗), which can also be considered one of the above levels. However, the extended signature levels are out of scope of this work.

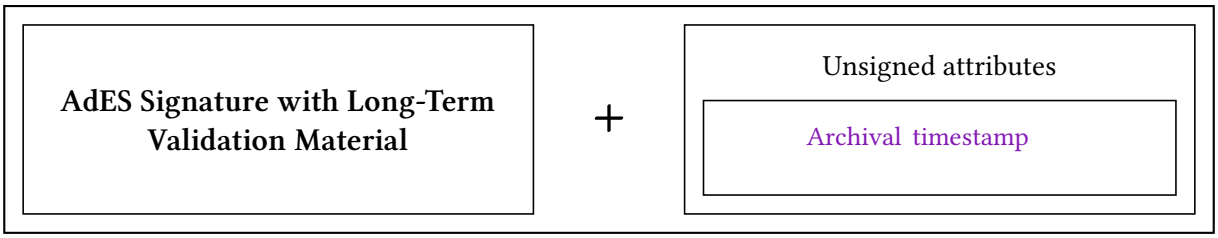



**Figure 6.4:** AdES Signature providing Long-Term Availability and Integrity of Validation Material Structure

- the signatures used to validate the signing certificate,
- the signature inside the non-archival timestamp,
- the signatures inside CRLs or OCSP responses,
- the signatures of certificates used to sign revocation status information (CRL or OCSP responses),
- the signature(s) inside the archival timestamp.

On top of that, there is a hash function corresponding to every signature. It may be that all signatures and hash functions are the same, but this is usually not the case: CA certificates are signed by more conservative signatures than end-user certificates (e.g., RSA with 4096-bit keys vs. 3072-bit keys, SHA2-512 vs. SHA2-256).

The specific signature schemes and hash functions that can be used in AdES are listed in [ETS23a].[2] Specifically, recommended hash functions are SHA2 and SHA3 with output lengths of 256, 384 or 512 bits. Recommended signature schemes are RSA (PKCS#1 v1.5 or PSS [JK03]) with at least 2048-bit keys, ECDSA [Nat23] and Schnorr-based ECSDSA with at least 256-bit elliptic curves.

#### 6.1.1.3 Impact of Quantum Computers.

In this section, we analyze the security threat that an adversary with a quantum computer poses to AdES.

As discussed in Section 1.4, quantum computers break most of the currently used cryptographic algorithms. Specifically, signature schemes RSA and ECDSA/ECSDSA, used in AdES, can be broken by a quantum computer. On the other hand, the recommended hash functions for AdES have output size $\geq 256$ bits, which makes them safe even against a quantum adversary.

Nonetheless, the security guarantees of AdES become void in the presence of a quantum adversary since the signature algorithms are insecure. Let us examine the potential steps of such an adversary trying to attack AdES signatures at various levels.

[2] The algorithms listed are only a recommendation which can be superseded by national requirements.

**AdES Basic Signature.** We assume the quantum adversary wants to forge an AdES Basic Signature. One option is to calculate the signer's private key from his public key (that is publicly available, e.g., in a certificate from previous signatures) and then create a new AdES signature.

The other, presumably more impactful, strategy is to compromise the trust anchor. The adversary calculates the trust anchor's private key (from the public key in the root certificate) and it can issue itself a new certificate with the target's identity and a new keypair.

The option of attacking the trust anchor has the benefit that it gives the adversary the ability to forge signatures on behalf of any subject part of that PKI. A potential disadvantage of this approach is that the signature algorithm used by the trust anchor has typically higher security level and the attack, i.e., calculating the private key, may take longer. However, Shor's algorithm scales polynomially; therefore, we consider such difference negligible in the grand scheme of things.

In total, the adversary needs to make one call to the quantum oracle to be able to forge an AdES Basic Signature.

**AdES Signature with Time.** Forging an AdES Signature with Time is slightly more complicated than forging a Basic Signature. That is because Signature with Time contains a timestamp computed over the signature. Therefore, the adversary additionally needs to forge the timestamp, which is just another signature.

In theory, the same trust anchor may be used for the document signing certificate and the timestamping certificate, i.e., the adversary would only need to compute the trust anchor's private key and sign his certificates for document signing and timestamping.[3] In practice however, the "timestamping PKI" is separate from the "signing PKI."

Overall, the adversary needs to forge two signatures. Similar to forging the basic AdES, the adversary can attack the trust anchors, or it can attack the end-user certificates. In both cases, the attack requires two calls to the quantum oracle.

**AdES Signature with Long-Term Validation Material.** Compared to AdES Signature with Time, the AdES Signature with Long-Term Validation Material contains revocation values corresponding to the document signing certificate (and others in its certificate chain) and contains certificates needed for the validation of the document signing certificate and the certificates used to sign the revocation status information.

In case the system that the adversary is attacking has a policy that allows only OCSP responses as the form of revocation values and the adversary chooses to create his own document signing certificate (by attacking the trust anchor), the adversary needs to forge the OCSP response as well. OCSP responses may be signed by the same private key as the signing certificate, or they may be part of some other PKI. In the latter case, the adversary also needs to attack the other PKI.

If CRL is used, the adversary does not need to forge anything as the CRL is not bound directly to his new forged certificate.

In total, compared to the AdES Signature with Time, forging an AdES Signature with Long-Term Validation Material requires at most $n \geq 0$ extra calls to the quantum oracle to forge $n$ OCSP responses (assuming the worst case of separate PKIs).

[3] By "document signing certificate" we mean the certificate whose public key corresponds to the private key used to compute the signature over the document and similarly for "timestamping certificate."

**AdES Signature providing Long-Term Availability and Integrity of Validation Material.** This AdES signature is augmented, compared to the AdES Signature with Long-Term Validation Material, with the archival timestamp. As this timestamp is computed over the whole signature, it is also necessary to forge it. Similar to previous attacks, the adversary can either compute the end-entity's private key and compute the specific timestamp or attack the trust anchor. We assume the worst-case scenario where each archival timestamp is part of a separate PKI.

In both cases, this increases the attack cost with $n \geq 1$ extra calls to the quantum oracle where $n$ is the number of archival timestamps applied.

**Conclusion.** In conclusion, we see that AdES signatures of all levels are insecure in the presence of a quantum adversary. We discuss how to secure them against quantum computers in Sections 6.2 and 6.3.

### 6.1.2 Comparison of AdES Signature Formats

This section aims to briefly compare AdES signature formats, as each format has its advantages and disadvantages.

**XAdES** builds upon the existing XML Signature standard [W3C03] whose primary use case is to sign XML documents. However, XAdES can also be used to sign arbitrary data. Such data can be either serialized, e.g., into Base64 and be part of the XML signature, or they can be *detached*[4] from the XML signature in a separate file. One of the main advantages of XAdES is that there exists a rich XML ecosystem, such as libraries and additional XML-related technologies such as XPath [W3C10] or XSLT [W3C17]. This also makes the format flexible, e.g., by allowing XPath expressions inside. However, this may introduce unforeseen vulnerabilities [Som14]. A disadvantage, compared to CAdES, is its slightly larger size due to the expressiveness of XML.

**CAdES** is based on the Cryptographic Message Syntax (CMS) which is an IETF standard [Hou09] for representing cryptographic data such as signatures or encrypted data. This format can sign any data. CAdES is a binary format and is therefore not human-readable, unlike XAdES. Typically, CAdES produces the smallest signatures as it is a binary format.

**PAdES** is a signature format tailored for signing PDF documents. Its advantages are that its validation and creation is supported by a wide variety of PDF readers and that it allows to put a visual representation of the signature inside the PDF file. The PAdES signature format cannot be used to sign anything other than PDF files. Because of the intricacies of the PDF standard, it is also complex to implement.

**JAdES** is similar to XAdES: it is human-readable, and it can be used to sign any document format. JSON is not as extendable and feature-rich compared to XML, which some may consider a downside, but in the security context this is actually a benefit—the attack surface is smaller. Since JAdES was introduced fairly recently, it is not broadly supported at the time of writing.

### 6.1.3 XML Advanced Electronic Signatures (XAdES)

This section introduces XAdES signatures and discusses their security, complementing the analyses from Sections 6.1.1.2 and 6.1.1.3.

[4] Such signatures are called "detached signatures" [W3C13c].

#### 6.1.3.1 XAdES Structure

XAdES [W3C03; ETS22] is an extension of the XML Signature standard [W3C13c], which is discussed in detail in Chapter 5. The aim of this section is to discuss how XAdES signatures differ from regular XML signatures.

First, we want to clarify which version of XAdES we consider. These versions do not differ that much in the cryptographic details, which is the focus of our work; however, the fact that there exist ETSI XAdES standards [ETS22; ETS10] and a W3C standard [W3C03] may cause some confusion. The W3C standard [W3C03] and the old ETSI standard [ETS10] were created in relation to the pre-eIDAS EU signature regulation.[5] eIDAS was created later and the implementing decision [Eur15] references the newer ETSI XAdES standard [ETS22]. Consequently, in our work we follow the newer standard [ETS22].

The main difference between "regular" XML signatures and XAdES signatures is that XAdES includes additional data, called *qualifying properties*, which can be of two categories:

- *Signed properties*: These properties are also signed during the signing process (together with the "input data").
- *Unsigned properties*: These properties are not signed, but they are included in the resulting XML signature element.

**Remark**

Recall from Section 6.1.1, that this difference corresponds to the difference between "regular" digital signatures and AdES signatures. The only difference is that XAdES refers to *attributes* as *properties*.

Qualifying properties are located inside the `Signature` element. Listing 6.1 shows its XML structure.

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <ds:SignedInfo>
      <ds:CanonicalizationMethod />
      <ds:SignatureMethod />
      <ds:Reference><!-- reference to the signed data --></ds:Reference>
      <ds:Reference><!-- reference to SignedProperties --></ds:Reference>
   </ds:SignedInfo>
   <ds:SignatureValue><!-- base64-encoded signature --></ds:SignatureValue>
   <ds:KeyInfo><!-- arbitrary key info --></ds:KeyInfo>
   <ds:Object>
      <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#">
         <xades:SignedProperties><!-- SignedProperties data --></xades:SignedProperties>
         <xades:UnsignedProperties><!-- UnsignedProperties data --></xades:UnsignedProperties>
      </xades:QualifyingProperties>
   </ds:Object>
</ds:Signature>
```

**Listing 6.1:** XAdES Structure

Recall from the XML signature standard [W3C13c] that `Object` is the only element, apart from `SignedInfo`, `SignatureValue` and `KeyInfo`, that is allowed to be a child of `Signature`. The standard does not impose any restrictions on the contents of `Object`; therefore, in XAdES it

[5] XAdES signatures conforming to the old W3C/ETSI standards are typically called *legacy XAdES signatures*.

is used to contain the element `QualifyingProperties`, which in turn contains two children: `SignedProperties` and `UnsignedProperties`.

**Remark**

Note that the W3C XAdES standard [W3C03] also allows to use an element `QualifyingPropertiesReference` that only contains a URI pointing to a `QualifyingProperties` element located outside of a `Signature` element. However, this is forbidden by the ETSI standard [ETS22, Section 6.3].

Typically, the `SignedInfo` element inside the main `Signature` contains two `Reference` elements; one referencing the signed data and the other referencing `SignedProperties`.

#### 6.1.3.2 Signature Levels

As XAdES is an instance of AdES, the standard [ETS22] defines four XAdES levels corresponding to AdES levels introduced in Section 6.1.1.1.

Each XAdES level is determined by the contents of `SignedProperties` and `UnsignedProperties`. Recall from Section 6.1.1.1, that the signature levels are **B-B**, **B-T**, **B-LT** and **B-LTA**. Next, we present how these levels are realized in XAdES and we discuss differences from AdES levels.

1) **B-B** (Figure 6.5): This is the base level of a (baseline) XAdES signature. It has the structure of an XML signature with the addition of some qualifying properties. Compared to an AdES Basic Signature, where only the signing certificate needs to be included, XAdES-B-B must also include among its *signed* properties the signing time and the MIME type of the signed data (document).

2) **B-T** (Figure 6.6): This level corresponds to AdES Signature with Time, i.e., a timestamp is included as an *unsigned* property. The input into the timestamp (`SignatureTimeStamp`) is the `SignatureValue` element.

3) **B-LT** (Figure 6.7): This level corresponds to AdES Signature with Long-Term Validation Material. This long-term validation material is placed inside the following *unsigned* properties:

   - `CertificateValues` contains certificates missing from existing elements such as `KeyInfo`.
   - `RevocationValues` contains revocation values corresponding to certificates inside the XAdES signature. Such revocation values are in the form of a CRL (Section 3.1.4.1) or an OCSP response (Section 3.1.4.2).

4) **B-LTA** (Figure 6.8): This level corresponds to AdES Signature providing Long-Term Availability and Integrity of Validation Material. The *archival* timestamp is placed inside the `ArchiveTimeStamp` element.

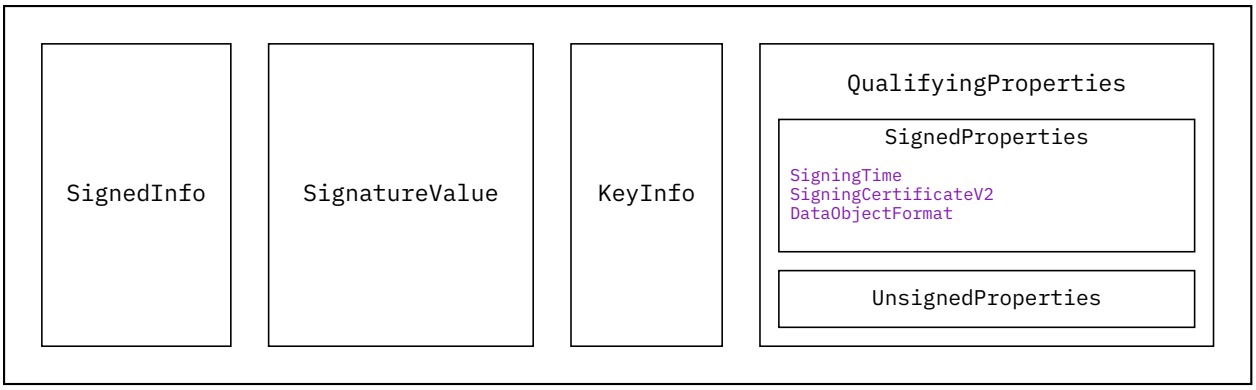



**Figure 6.5:** XAdES-B-B Structure

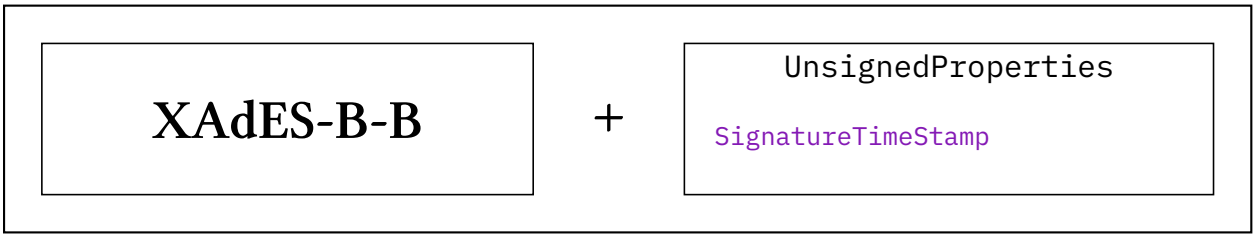



**Figure 6.6:** XAdES-B-T Structure

#### 6.1.3.3 Security

The security analyses from Sections 6.1.1.2 and 6.1.1.3 apply to XAdES. One minor difference is that the ECSDSA signature algorithm is not supported in XAdES, but this does not affect the outcome of our analyses. In conclusion, XAdES signatures are insecure in the presence of quantum adversaries and need to be migrated to post-quantum cryptography.

### 6.1.4 CMS Advanced Electronic Signatures (CAdES)

Similarly to the previous section, this section introduces CAdES signatures and discusses their security with respect to the analyses from Sections 6.1.1.2 and 6.1.1.3.

#### 6.1.4.1 CAdES Structure

CAdES [ETS23b] is an extension of CMS signatures described in RFC 5652 [Hou09]. CAdES signatures differ from "regular" CMS signatures in the same way that XAdES signatures (Section 6.1.3) differ from XML signatures; they are extended by the inclusion of additional data, called *attributes*.

Note that there exists another CAdES ETSI standard from the pre-eIDAS era [ETS13]. We do not consider this standard and note that CAdES signatures compliant with this standard are referred to as *legacy CAdES signatures*.

**CMS Signature Structure.** First, we briefly present "regular" CMS signatures as defined in [Hou09]. CMS signatures, unlike XML signatures, are not human-readable. They are binary blobs that are the result of BER/DER encoding [ITU21a] of ASN.1 structures [ITU21b]. X.509 certificates,

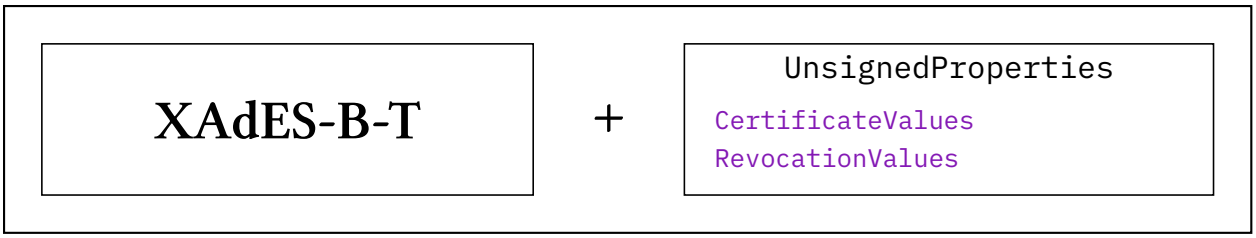



**Figure 6.7:** XAdES-B-LT Structure

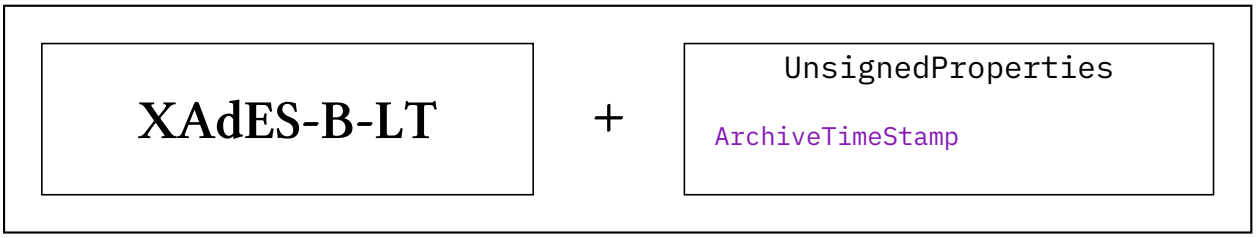



**Figure 6.8:** XAdES-B-LTA Structure

CRLs [Boe+08] or timestamps [Zuc+01] are also encoded using this procedure. For the rest of the section, we assume the reader is familiar with ASN.1.

The CMS signature is based on the following generic structure:

```
ContentInfo ::= SEQUENCE {
   contentType ContentType,
   content [0] SignedData
}
```

where

```
ContentType ::= OBJECT IDENTIFIER

--- and

SignedData ::= SEQUENCE {
   version CMSVersion,
   digestAlgorithms DigestAlgorithmIdentifiers,
   encapContentInfo EncapsulatedContentInfo,
   certificates [0] IMPLICIT CertificateSet OPTIONAL,
   crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
   signerInfos SignerInfos
}
```

**Remark**

In a CMS signature, the `ContentType` identifier is a constant: `1.2.840.113549.1.7.2`, which is an OID for the `SignedData` type. This is because CMS [Hou09] does not only define signatures, but also allows other content types such as encrypted data (Section 3.1.5.3).

`SignedData` has the following fields:

- `version`: This is a version identifier.
- `digestAlgorithms`: This field contains all hash algorithms used inside this signature.
- `encapContentInfo`: This field contains the signed data. The field is of type `EncapsulatedContentInfo`:

```
EncapsulatedContentInfo ::= SEQUENCE {
   eContentType ContentType,
   eContent [0] EXPLICIT OCTET STRING OPTIONAL
}
```

  where `eContentType` is an OID of the signed data and data bytes are in the `eContent` field. Note that `eContent` is optional, i.e., the signed data do not have to be inside the CMS signature. In this case, the CMS signature is a *detached/external signature*.[6]

- `certificates`: This field contains certificates needed to validate the signature.
- `crls`: This field contains revocation information corresponding to the certificates inside the `certificates` field. It can be in the form of CRLs or OCSP responses.
- `signerInfos`: This field is a set of values of type `SignerInfo`, where each `SignerInfo` corresponds to a separate signature of the same data. It has the following structure:

```
SignerInfo ::= SEQUENCE {
   version CMSVersion,
   sid SignerIdentifier,
   digestAlgorithm DigestAlgorithmIdentifier,
   signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
   signatureAlgorithm SignatureAlgorithmIdentifier,
   signature SignatureValue,
   unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL
}
```

  - `version`: This is a version identifier.
  - `sid`: This field contains information identifying the signer's certificate, i.e., the signer. It can contain the serial number and the issuer, or it can contain a subject key identifier (an X.509 extension).
  - `digestAlgorithm`: This identifies the hash algorithm used to produce the hash/digest that was signed.
  - `signedAttrs`: This is a set of attributes (of type `Attribute`) that are protected by the signature. It must always contain attributes `message-digest` and `content-type`. See Section 6.1.4.2 for further details.

[6] This is similar to detached XML/XAdES signatures.

- `signatureAlgorithm`: This identifies the signature algorithm used to produce the signature inside `signature`.
- `signature`: The signature value, i.e., the bytes returned by the signature algorithm.
- `unsignedAttrs`: This is a set of attributes not protected by the signature. See Section 6.1.4.2 for further details.

**Signing Procedure.** Let us briefly explain how CMS signatures are created. The signing procedure varies depending on the inclusion of `signedAttrs` and the signed data inside `encapContentInfo`.

If `signedAttrs` is present, *only* this field is the input into the hash function, specified in `digestAlgorithm`, and the resulting digest is the input into the signature algorithm, specified in `signatureAlgorithm`. This is because according to the standard [Hou09], if `signedAttrs` is present, it must contain the attribute `message-digest` (see Section 6.1.4.2), which contains the digest of `eContent` from `encapContentInfo`, i.e., the hash of signed data is part of `signedAttrs`.

If `signedAttrs` is not present, *only* the `eContent` field from `encapContentInfo` is the input into the hash function and the resulting digest is signed.

In the first case, the signed data is hashed "indirectly" by hashing the signed attributes and in the second case it is hashed directly. In both cases, the digest is signed and the result of the signing operation is placed inside `signature` inside `SignerInfo`.

**Remark**

`signerInfos` inside `SignedData` can contain multiple `SignerInfo` types. This means that a single CMS signature can contain multiple signatures of the *same data*.[a] This may be useful in cases where

- one signer wants to provide signatures using different algorithms for better interoperability, i.e., RSA signature and ECDSA signature, or
- there are multiple signers that want to sign the same data and produce a single CMS signature.

[a] Technically, each `SignerInfo` can have different `signedAttrs` fields, i.e., the signatures cover different data. However, it is assumed that they indirectly sign the same content in `encapContentInfo`.

**Remark**

In the case of *detached/external* CMS signatures, the field `eContent` in `encapContentInfo` is not present. Therefore, it is expected that the external data is referenced using the hash inside `signedAttrs`. However, the standard does not explicitly prohibit a CMS signature with no `eContent` and no `signedAttrs`; in this case, the verifier has to know what data was input into the signing algorithm. We consider this a "degenerate" case and we do not consider it in our work.

**Verification Procedure.** The verification procedure of CMS signatures mirrors the signing procedure. First, the verifier needs to compute the digest of the signed data. This is done either by hashing the `eContent` in `encapContentInfo` if it is present, or by hashing the external data.

If the `signedAttrs` is present, it is necessary to verify that the digest inside `signedAttrs` is identical to the just computed one. Then, `signedAttrs` is hashed and it is verified that the value inside `signature` is a valid signature of the computed digest.

If there is no `signedAttrs` field, then it is verified that the value inside `signature` is a valid signature of the computed digest of the signed data.

Of course, the authenticity of the verification public key must be also verified using the included certificate and CRL data.

**Remark**

It is also necessary to verify that the `content-type` signed attribute (Section 6.1.4.2) matches the one specified inside `encapContentInfo`. This could lead to vulnerabilities in which an adversary changes the meaning of the signed data by modifying the `eContentType` value.

#### 6.1.4.2 Signature Levels

Recall that CAdES signatures differ from "regular" CMS signature in the inclusion of attributes. These attributes can be *signed*, placed inside `signedAttrs`, or *unsigned*, placed inside `unsignedAttrs`. The CMS standard [Hou09] requires that if the CMS signature contains any signed attributes, attributes `content-type` and `message-digest` must be present.

Similar to XAdES signatures (Section 6.1.3), there are four levels of CAdES signatures that correspond to AdES levels introduced in Section 6.1.1.1.

1) **B-B**: This is the base level for CAdES signatures. It must contain the following signed attributes:

   - `content-type`: This attribute identifies the signed data. In CAdES, this value is always `id-data`, i.e., arbitrary data.
   - `message-digest`: This attribute contains the digest of the signed data.
   - `signing-certificate-v2`: This attribute contains the hash of the signing certificate, i.e., it ties the verification public key to the CAdES signature. However, the signing certificate itself is not included (it is in the `certificates` field of `SignedData`, see below).[7]
   - `signing-time`: This attribute contains the time when the signature was created.[8]

   Apart from these signed attributes, the CAdES signature must include all certificates needed to validate the signature in the `certificates` field of `SignedData`.

[7] CAdES also allows an older version of this attribute (`signing-certificate`), but we consider that one deprecated as it uses an insecure SHA-1 hash function.

[8] This value is set by the signer. Therefore, it is not an independent proof that the signature existed at that time, unlike a third-party timestamp.

**Remark**

Note that the `certificates` field is not protected by the signature. This allows the addition of new certificates required for successful validation, e.g., when the CAdES signature gets augmented into a higher level.

2) **B-T**: This level corresponds to the AdES Signature with Time. Consequently, a timestamp is included as an *unsigned* attribute `signature-time-stamp`. The timestamp is in the standard format described in Section 3.1.6.1. The value that is timestamped is the `signature` field in `SignerInfo`.

3) **B-LT**: This level corresponds to AdES Signature with Long-Term Validation Material. The long-term validation material is included either in the `certificates` field or in the `crls` field of `SignedData`.[9] The same rules apply as for the AdES level, i.e., it should include all the necessary certificates and revocation information needed to verify the document signature and the timestamp(s).

4) **B-LTA**: This level corresponds to AdES Signature providing Long-Term Availability and Integrity of Validation Material. The archival timestamp is in the form of an unsigned attribute `archive-time-stamp-v3`. The timestamp format is identical to the non-archival timestamp, but the timestamped data is the concatenation of

   - the `eContentType` field inside `encapContentInfo` of the CAdES signature,
   - the hash of the signed data, i.e., the hash of the `eContent` field, or of external data,
   - the fields `version`, `sid`, `digestAlgorithm`, `signedAttrs`, `signatureAlgorithm` and `signature` from `SignerInfo`
   - and the `ats-hash-index-v3` attribute, which is an unsigned attribute of the `archive-time-stamp-v3`.[10]

   The `ats-hash-index-v3` attribute is essentially a collection of hashes of the certificates and revocation values from `certificates` and `crls` fields from the CAdES `SignedData`. The purpose of the `ats-hash-index-v3` attribute is to assure that the values inside `certificates` and `crls` fields are also protected by the archival timestamp.

**Remark**

A reader might wonder why the `ats-hash-index-v3` attribute is used to link the certificate and revocation information to the archival timestamp as opposed to simply hashing those fields together with the list of above. According to the standard [ETS23b], this allows new values to be added into the `certificates` or `crls` fields without invalidating the archival timestamp, which can be useful in the case of multiple signers, etc.

[9] The `crls` field may include also OCSP responses despite the name.
[10] A timestamp is technically a CMS signature. Therefore, it can have signed and unsigned attributes.

#### 6.1.4.3 Security

As with XAdES, the security analyses from Sections 6.1.1.2 and 6.1.1.3 apply to CAdES as well. In conclusion, CAdES signatures are insecure in the presence of quantum adversaries and need to be migrated to post-quantum cryptography.

### 6.1.5 PDF Advanced Electronic Signatures (PAdES)

This section introduces PAdES signatures and analyzes the impact of quantum computers upon their security guarantees. PAdES [ETS24a] is an extension of CMS-based PDF signatures specified in the PDF standard [Ado08]. Compared to XAdES (Section 6.1.3) or CAdES (Section 6.1.4), PAdES signatures can only be used to sign PDF documents and are embedded in the signed PDF documents.

Similar to other formats, there also exist *legacy PAdES signatures*, defined in [ETS09], which are the predecessor of the eIDAS-compliant PAdES signatures. In this work, we focus only on the current (non-legacy) eIDAS-compliant PAdES signatures.

We briefly introduce the "regular" PDF signatures, defined in [Ado08], and then we discuss their differences from PAdES [ETS24a]. At the end, we discuss the security of PAdES with respect to the threat of quantum computers.

#### 6.1.5.1 PAdES Structure

First, we briefly present the PDF format and then PDF signatures in general. After that, we focus on PAdES and discuss how it differs from "regular" PDF signatures.

**PDF File Structure** A PDF file is a mixture of a binary format and a text-based format. Specifically, the majority of the PDF content (fonts, text, graphics) is typically encoded into binary and is not human-readable; however, the structural elements are strings of text.

A PDF has the following structure:

- **Header**: The header consists of two lines. The first line has the format `%PDF-{version}`, where `{version}` defines the version of the standard. The second line[11] consists of 4 bytes with values $> 127$, i.e., none of them correspond to ASCII characters. The purpose of the second line is to help detect that the file is a PDF file.

- **Body**: This is typically the largest section of a PDF. It consists of *indirect objects*. An *object* is a fundamental PDF data type that represents text or binary data, or other objects. An *indirect* object is an object with a unique identifier.

  Listing 6.2 shows an example of an indirect object. The first number is the identifier, the second number is the generation number (which is typically 0). Then, there is the string `obj`, which ends the "object header". What follows is the body of the object, which consists of other objects. At the end, the indirect object ends with the string `endobj`. This example object is the "catalog" object, which contains information about the pages of the PDF file

[11] On UNIX systems, a line is ended by the `\n` separator, i.e., the byte `0x0A`. Other operating systems follow similar conventions.

and other data. For example, the line `/Pages 6 0 R` says that information about the pages is contained in another indirect object with the ID 6.

```
39 0 obj
<<
/Type /Catalog
/Pages 6 0 R
/PageMode /UseOutlines
/OpenAction [1 0 R /XYZ null null 0]
/Lang (hu-HU)
/DSS 45 0 R
>>
endobj
```

**Listing 6.2:** Indirect Object.

Note that the body of an indirect object does not have to be readable and can be a sequence of bytes, which is typical as it allows compression, which reduces the file size.

- **Cross-reference (xref) table**: This section is a table that contains references to objects inside the body. The table begins with the keyword `xref` followed by the header specifying the starting object ID and the number of entries. Each entry consists of three values: the byte offset, the generation number and a flag. An example table is shown in Listing 6.3.

```
xref
0 2
0000000000 65535 f
0000166811 00000 n
```

**Listing 6.3:** Cross-Reference Table Example.

- **Trailer**: The trailer is located at the end of the PDF file. It begins with the string `trailer` and is followed by a sequence of objects, which specify the size of the xref table, the "root object," and other metadata. After these objects, there is a string `startxref` followed by the byte offset of the xref table in the file. The end of the trailer, and thus the end of the PDF file, is marked by the string `%%EOF`. An example trailer is depicted in Listing 6.4.

```
trailer
<<
/Size 57
/Root 39 0 R
/Info 40 0 R
/ID [<C547234ACB212B7CD65BA961F9E9ACAE> <0D7EDB787AF81196EB8D945B2EDF13B7>]
/AdditionalStreams [/application#2Fvnd.oasis.opendocument.text 57 0 R]
/Prev 166170
>>
startxref
186364
%%EOF
```

**Listing 6.4:** Trailer Example.

**Remark**

Since PDF version 1.5, a PDF file does not contain the strings `trailer` and `xref`, and it encapsulates the xref table data and the trailer data (except the last 3 lines) into regular indirect objects. The xref table may also be compressed into a sequence of bytes.

**Incremental Updates.** The PDF format supports so-called "incremental updates." This allows a PDF file to be updated without changing the existing file structure. The update consists of appending the following to the end of the file:

- **New body**: contains a set of new objects, e.g., additional text or images.
- **New cross-reference table**: contains references only to new/updated/removed objects.
- **New trailer**: contains information about the updated part, such as the byte offset of the new xref table or the root object, and also the byte offset of the old xref table.

Note that this mechanism is used in the context of PAdES signatures, e.g., when adding an archival timestamp to the file (see Section 6.1.5.2).

**PDF Signature.** A PDF signature can be *visual* or *invisible*. The difference between them is that the visual signature contains a visual representation inside the PDF. As we are interested in the technical cryptographic aspect, we do not discuss visual signatures further and focus on invisible signatures.

A digital signature inside a PDF is represented by an indirect object. The signature object usually contains the following parameters:

- **Type**: The type of the object, for signature it is `Sig`.
- **Contents**: The "raw" signature data. The value depends on the signature format; see the **SubFilter** parameter below. However, in our case, the value is a hexstring representing an encoded CMS signature.
- **Filter**: This specifies the "handler", i.e., software to use for validating the signature. This value is typically `Adobe.PPKLite`, which corresponds to the standard validator.
- **SubFilter**: This specifies the signature format/encoding. The possible values are:
  - `adbe.x509.rsa_sha1`: A legacy format, which did not use CMS [Hou09] and encoded the signature/certificates directly as separate entries in the signature object.
  - `adbe.pkcs7.sha1`: A legacy format, which used CMS, but encapsulated the document SHA-1 digest as the `eContent` value inside the CMS signature.
  - `adbe.pkcs7.detached`: A standard format, where CMS is used and the `eContent` is absent from the CMS signature, i.e., it is a detached CMS signature.

  Note that we focus only on the `adbe.pkcs7.detached` format from now on.
- **ByteRange**: This value is an array of byte ranges, which specify what parts of the PDF file are covered by the signature. In most cases, the signature covers the whole file except the not-yet-existent raw signature object (value of **Contents**). Therefore, the typical byte range has the following format `[0 a a+s b]`, where `a` is the number of bytes before the signature, `s` is the size of the hexstring inside the `Contents` field and `b` is the number of bytes after the signature.

Note that the **ByteRange** is also protected by the signature. Therefore, the size of the signature (value of `s`) must be known before the actual signature is computed. This is solved by estimating the size of the signature beforehand and filling the remaining bytes in `Contents` with the value `0x30` (representing the string `0`).

**Remark**

The fact that the size of the CMS signature in the PDF is upper bounded means that the CMS signature itself cannot be easily augmented, e.g., by adding a timestamp (as an unsigned parameter). This results in the need to use alternative mechanisms to "update" PDF signature. We discuss these in Section 6.1.5.1.

Let us describe the signing and verification procedure for the PDF signature of the type `adbe.pkcs7.detached`, which is almost identical to the format used in PAdES.

*Signing Procedure.* As hinted already, the signing procedure works as follows. First, the signature object is inserted somewhere inside the PDF and its parameters are set; most importantly the **ByteRange** is specified as described above (this involves estimating the future signature size). Then, a detached CMS signature is created, where the input is the bytes of the PDF file as specified in **ByteRange**. The CMS signature must contain the signing certificate as a signed attribute and may include other attributes such as CRLs or other certificates. The CMS signature is then encoded into a hexstring that is then placed as the value of **Contents** and the hexstring is filled in with zeros (as described above) so the size matches the estimated values in **ByteRange**.

*Verification Procedure.* As usual, the verification procedure is the inverse of the signing procedure. The signature is located inside the PDF; the input bytes are collected and verified against the CMS signature inside **Contents**, which may include verifying the signing certificate etc.

**PAdES Signature.** PAdES is an extension of the PDF signature format `adbe.pkcs7.detached`. It has its own format name: `ETSI.CAdES.detached` and imposes additional restrictions and requirements.

One such restriction is that inside the CMS signature, there may only be one `SignerInfo`, i.e., it does not allow multiple signers to sign the same document. So, if multiple signers are needed, the PDF file gets signed sequentially using the incremental update mechanism described in Section 6.1.5.1. However, this does not apply only to PAdES because this is also how it is typically done in "regular" PDF signature.[12]

The signing certificate must be included into the `certificates` field in the CMS signature and other certificates in the chain should also be included. Note that no requirements are listed regarding the `crls` field used for revocation information. This is because the `crls` field is not used in PAdES, and instead it uses the Document Security Store (DSS), which is a PDF object.

**Document Security Store (DSS).** DSS is an indirect PDF object, which (primarily) contains fields **Certs**, **OCSPs** and **CRLs**. These fields allow including additional certificates and revocation information, in the form of a CRL or OCSP response, into the PDF. As this information is typically

[12] To the best of our knowledge, no PDF reader supports multiple `SignerInfos` inside the CMS signature. The typical behavior is that one of the `SignerInfos` is ignored.

included after the signature creation, the DSS object is included into the PDF using the incremental update mechanism (Section 6.1.5.1).

**Remark**

One of the reasons for using DSS, instead of using the dedicated CMS signature fields, is that the CMS signature is limited in size at the creation and there may be not enough space to contain all necessary data. This is why the information (the DSS object) is external to the CMS signature.

Other PAdES requirements/restrictions depend on the signature levels described in Section 6.1.5.2.

#### 6.1.5.2 Signature Levels

Similar to XAdES (Section 6.1.3) and CAdES (Section 6.1.4), PAdES defines four signature levels corresponding to the AdES levels from Section 6.1.1.1.

1) **B-B**: This is the base level. There are several requirements imposed. As already noted, the `certificates` field in the CMS signature must contain the signing certificate. The CMS signature must also contain the signed attributes: `content-type`, `message-digest` and `signing-certificate-v2`. In the signature PDF object, there must also be a parameter **M** which contains the claimed signing time.

2) **B-T**: This is the level of a signature with a timestamp. The PAdES standard allows for the timestamp to be included in two ways:

   (a) As an unsigned CMS attribute `signature-time-stamp` in the same way as in CAdES (Section 6.1.4).

   (b) As an indirect PDF object of type **DocTimeStamp**, which is added to the PDF file using the incremental update mechanism (Section 6.1.5.1).

   The timestamp object has a similar structure to the PDF signature object (with the **ByteRange** parameter etc.). The value of **Contents** is a `TimeStampToken` (Section 3.1.6.1). The input into the timestamping procedure is the bytes specified in **ByteRange**.

3) **B-LT**: This level corresponds to the AdES Signature with Long-Term Validation Material. This level necessitates that the PAdES signature contains all certificates and revocation information needed to validate the signature and timestamp(s).

   Recall from Section 6.1.5.1, that PAdES allows to include certificates in the CMS signature (in the `certificates` field), but it also defines the Document Security Store (DSS), which is a PDF object that can contain certificates and revocation information. According to the standard, DSS must include the necessary certificates missing from the CMS signature, and it must include the revocation information either as CRLs or OCSP responses.

4) **B-LTA**: This is the level with the *archival* timestamp. Unlike the "regular" timestamp, the archival timestamp must be included as the separate PDF object **DocTimeStamp**. Otherwise, the process is identical to adding a "regular" timestamp. The archival timestamp

protects/certifies the certificate and revocation data in the DSS enabling the validation of the signature even after certain algorithms become weak etc.

#### 6.1.5.3 Security

As with XAdES and CAdES, the security analyses from Sections 6.1.1.2 and 6.1.1.3 apply to PAdES as well. In conclusion, PAdES signatures are insecure in the presence of quantum adversaries and need to be migrated to post-quantum cryptography.

### 6.1.6 JSON Advanced Electronic Signatures (JAdES)

In this section, we introduce the JSON Advanced Electronic Signature (JAdES). The JAdES standard [ETS24b] extends the JSON Web Signature (JWS) standard [JBS15a] by defining new properties which correspond to qualifying properties defined in other AdES formats. We assume the reader is familiar with the JSON format [Bra17].

#### 6.1.6.1 JAdES Structure

First, we introduce JSON Web Signatures (JWS), and then we describe how JAdES extends JWS.

**JSON Web Signature (JWS)** The JSON Web Signature standard [JBS15a] defines how to represent signed data as JSON objects. A JWS signature can be decomposed into three components:

- the header,
- the signed data (the payload),
- the signature value.

**Remark**

The JWS standard supports also MAC (Definition 4) schemes to produce the "signature" apart from regular signature schemes. In that case, the signature value is the tag (in the MAC terminology).

**The Header** The header can be further decomposed into *the protected header* and *the unprotected header*. As the name suggests, the protected header is signed together with the payload while the unprotected header is not. This is similar to AdES signed and unsigned qualifying properties.

The header contains metadata about the signature and the signed data. The JWS standard defines the following header parameters:

- `alg`: This parameter identifies the signing algorithm. It needs to be present inside every JWS.
- `jku`: This parameter contains a URI referring to the location of the public key corresponding to the signing key. The public key is in the JSON Web Key format [Jon15].

- `jwk`: This parameter contains the public key corresponding to the signing key. The public key is in the JSON Web Key format [Jon15].
- `kid`: This parameter contains a hint identifying the signing certificate.
- `x5u`: This parameter contains a URI referring to the location of the signing certificate (or the whole chain).
- `x5c`: This parameter contains the signing certificate (or the whole chain).
- `x5t`: This parameter contains a SHA-1 digest of the signing certificate.
- `x5t#S256`: This parameter contains a SHA-256 digest of the signing certificate.
- `typ`: This parameter describes the type of the signature. Its purpose is to differentiate different kinds of signatures.
- `cty`: This parameter describes the content type of the signed data.
- `crit`: This parameter is an array of header parameter names. Its purpose is to identify critical parameters, which must be present.

Except `alg`, all other parameters are optional. The standard also imposes a restriction that if `crit` is present, it must be inside the protected header.

The protected header and the unprotected header are regular JSON objects with properties defined above. See Listing 6.5 for an example header.

```
{
   "alg": "ES256",
   "typ": "JWT",
   "cty": "text/html",
   "kid": "example-key-123",
   "jwk":
   {
      "kty": "EC",
      "crv": "P-256",
      "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
      "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
      "kid": "example-key-123"
   },
   "x5u": "https://example.com/cert.pem",
   "x5c":
   [
      "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQ...",
      "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQ..."
   ],
   "x5t#S256": "47DEQpj8HBSa-_TImW-5JCeuQeRkm5NMpJWZG3hSuFU"
}
```

**Listing 6.5**: JWS Header Example

During processing, the header is split into the unprotected part and the protected part and treated as separate JSON objects.

**Encodings** The resulting JWS object needs to be serialized; the standard defines two types of serialization:

- Compact Serialization,
- JSON Serialization.

*Compact Serialization* This method is defined only for a JWS *without* an unprotected header. The result is a concatenation of three Base64URL [Jos06][13] strings with the character `"."` acting as the separator. Specifically, the encoded signature is computed as follows

```
B64(header) || "." || B64(signed data) || "." || B64(signature value)
```

where `B64` is the Base64URL encoding function. Listing 6.6 is an example JWS serialized using this method.

```
eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImpvaG4iLCJ
hZG1pbiI6dHJ1ZX0.wCuL5725P7j5dswhVQP8uHNMQyELcnCbI0h4QWirjSS79rav
UK777ODTREikVuYD7Egcg9bDa8PLRb5NHlLVoA
```

**Listing 6.6:** Example JWS serialized using the compact method.

*JSON Serialization* This method supports having a JWS with both the protected header and the unprotected header. It also supports multiple signatures with different headers. The result is a JSON object with two properties:

- `payload`: This property contains the Base64URL encoded signed data.
- `signatures`: This property is an array of objects consisting of three properties:
  - `protected`: This is the protected header value. It is a Base64URL string which is the encoded protected header object.
  - `header`: This is the unprotected header object.
  - `signature`: This is the signature value. It is a Base64URL encoded string.

Note that the protected or the unprotected header must be always present. Clearly, the header data is usually not identical for multiple signatures as it typically contains the information about the signing algorithm and the signing key.

The standard additionally defines a *flattened* version of this encoding for the case where there is only one signature; the `signatures` property is replaced with its three child properties. Listings 6.7 and 6.8 are examples of a JWS encoded using the JSON general serialization and the JSON flattened serialization, respectively.

[13] Base64URL is a version of Base64 encoding with a URL-friendly alphabet.

```
{
   "payload": "eyJpc3MiOiJqb2UiLA0KICJl....",
   "signatures":
   [
      {
         "protected": "eyJhbGciOiJSUzI1NiJ9",
         "header":
         {
            "kid": "2010-12-29"
         },
         "signature": "cC4hiUPoj9Eetdgtv3hF80EGrhu..."
      },
      {
         "protected": "eyJhbGciOiJFUzI1NiJ9",
         "header":
         {
            "kid": "abcd"
         },
         "signature": "DtEhU3ljbEg8L38VWAfUAqOyKA...."
      }
   ]
}
```

**Listing 6.7**: Example JWS serialized using the JSON general method taken from [JBS15a].

```
{
   "payload": "eyJpc3MiOiJqb2UiLA0KICJl....",
   "protected": "eyJhbGciOiJFUzI1NiJ9",
   "header":
   {
      "kid": "abcd"
   },
   "signature": "DtEhU3ljbEg8L38VWAfUAqOyKA...."
}
```

**Listing 6.8**: Example JWS serialized using the JSON flattened method taken from [JBS15a].

**Signing Procedure.** The signing procedure is fairly straightforward. The input into the signing algorithm is the following ASCII-encoded string

```
B64(protected header) || "." || B64(signed data)
```

Note that if the protected header is not present then it is an empty string. The signing algorithm must correspond to the `alg` header parameter. In case of multiple signatures, this process is repeated with different headers. At the end, the JWS is constructed based on the chosen encoding.

**Verification Procedure.** The verification procedure is as follows. Note that we consider there is only one signature inside the JWS. Otherwise, steps 2 to 5 are repeated. If all signatures are valid, the resulting JWS is valid. If all signatures are invalid, the resulting JWS is invalid. In cases where some are valid and some are invalid, the result depends on the application.

1) Deserialize the JWS into its components.
2) Verify that the unprotected and protected headers are valid JSON objects. Additionally, verify that they do not have duplicate properties.
3) Verify that there are no unknown properties marked as critical.
4) Retrieve the verification key. Part of this process entails the certificate verification if X.509 certificates are used.
5) Verify that the deserialized signature value is a valid signature for the signed data and the protected header.

**Detached Payload.** The JWS standard [JBS15a] (in its Appendix F) hints at the possibility of omitting the signed data/payload from the serialized signature, allowing for the payload to be detached. For compact serialization the payload string would then be an empty string and for JSON serialization the payload property is omitted. However, the standard does not discuss methods how to identify the signed payload.

**JAdES Extension.** Recall that JAdES [ETS24b] builds upon the JWS standard [JBS15a]. As with other formats, the difference is in the inclusion of additional properties. Specifically, some are signed properties inside the protected header, and some are unsigned properties that need to be placed inside the unprotected header. As a consequence, only the **B-B** level JAdES signature can be serialized using the compact method as higher levels require the presence of the unprotected header. Additionally, JAdES provides a more robust solution to detached payloads in JWS using the `sigD` unsigned property.

**Detached Payload in JAdES.** A JAdES with a detached payload is identified identically to a regular JWS with a detached payload; in the case of the compact serialization, the payload string is empty and in the case of the JSON serialization, the payload property is missing. Additionally, JAdES allows linking the detached payload (signed data) to the JAdES signature structure.

This is done using the *signed* header parameter `sigD`. In this document, we do not go into the details of the mechanism as it is very technical. In summary, the `sigD` parameter can refer to a detached payload in three ways:

1) Indicating that the payload is a certain subset of an HTTP header. This is a method tailored specifically for a use case where HTTP requests/responses are signed using JAdES.
2) Indicating that the payload is present at a certain location identified by a URI.
3) Similar to 2) but the reference also contains digest(s) of the payload.

In the case 2), the payload is an input into the signing/verification function of the signature scheme. In the case 3), since `sigD` contains a digest of the payload and it is a signed parameter, the payload does not need to be a direct input. However, during the verification, it must be verified that the digest inside `sigD` matches the payload.

#### 6.1.6.2 Signature Levels

As with other AdES formats, JAdES signatures have four levels which differ in the included additional data (parameters).

1) **B-B**: This is the JAdES base level signature. The requirement is that the following signed parameters are present:

   - `alg`: Defined as in JWS.
   - `x5t#o`: This is similar to `x5t#*` parameters defined in JWS, but this one is generalized as it allows specifying the hash function. According to the standard, this parameter may be replaced by others that serve the same function, which is to reference the signing certificate. The alternatives are: `x5t#S256`, `x5c` [JBS15a], `sigX5ts` [ETS24b].
   - `iat`[14]: This property contains the signing time. This property is defined in the JSON Web Token standard [JBS15b].

2) **B-T**: This is the signature with timestamp level. The timestamp must be an unsigned property as it is applied after the signature value is computed. To envelope all JAdES unsigned parameters, the standard defines a single unsigned (a part of the unprotected header) property `etsiU`. This property is an array of JAdES-defined unsigned properties.

   One such property is the signature timestamp `sigTsp`. This property has two sub-properties:

   - `canonAlg`: This is an identifier for a canonicalization algorithm.
   - `tstTokens`: This is an array of objects of type `tstToken`, which contains the Base64-encoded TSP timestamp and some metadata.

3) **B-LT**: This is the long-term validation level where necessary certificates and revocation data need to be included. JAdES defines multiple unsigned properties for this purpose:

   - `xVals`: This property is an array for the necessary certificates for validation.
   - `rVals`: This property is an object containing array(s) the revocation data such as OCSP responses, CRLs.
   - `anyValData`: This property is a single property which is essentially a union of `xVals` and `rVals`. In some use cases this may be useful.
   - `tstVD`: This property serves a similar purpose as the above properties except it contains only timestamp-related data.

[14] In previous version of the standard a custom JAdES-defined property `sigT` was used instead.

4) **B-LTA**: This is the archiving level where the archival timestamp is included. The archival timestamp property is `arcTst`, and it is of the same type as `sigTsp` except the timestamp is calculated over the whole signature. Additional timestamps are always appended into `etsiU` as the last element.

#### 6.1.6.3 Security

As with other AdES formats, JAdES is vulnerable to quantum adversaries as it currently supports only classical signatures.

### 6.1.7 Related Topics

In this section, we briefly touch upon Evidence Record Syntax (ERS) and Associated Signature Container (ASiC), which are standards and technologies that further extend AdES.

#### 6.1.7.1 Evidence Record Syntax

Evidence Record Syntax (ERS) [GBP07] defines a way to prove the existence and integrity of data over a possibly undetermined period of time.

It is similar to what the AdES LTA level provides. The main difference is that ERS provides a more efficient method in cases where there are multiple documents that need to be "future proofed" together. With **B-LTA** AdES, we apply the archival timestamp over each AdES signature and the same applies to renewals/extensions—we need to sign each document individually.

To solve this inefficiency, ERS uses Merkle trees. It builds a Merkle tree where the leaf nodes are hashes of the input documents and then only the root hash is timestamped. Then, each document signature is extended by including a *partial hash tree* and a timestamp. The partial hash tree contains all the necessary nodes (hashes) from the tree that are required to rebuild the root hash (assuming we have the signed document). During verification, the root hash is calculated, and then the timestamp is validated. The benefit of this approach is that the slower asymmetric operations of signing/verifications are replaced by faster hash calculations.

ERS [GBP07] also defines a way to handle cases where the signature algorithm (inside the timestamp) or the hash algorithm (used to build the Merkle tree) become weak. It is similar to the procedure of applying a new archival timestamp in AdES.

ETSI has defined a method for integrating ERS into AdES signatures, specifically for XAdES [ETS21] and for CAdES [ETS17a]. We do not focus on ERS in more detail as it is out of scope of this document, however, clearly ERS used with classical algorithms suffers from the same vulnerabilities which can be exploited by a quantum adversary.

#### 6.1.7.2 Associated Signature Containers

The Associated Signature Containers (ASiC) standard [ETS16b] defines how to couple together documents and their signatures. An AdES signature is either part of the same file as the signed document (referred to as *enveloping*/*enveloped* signature), or it is *detached*, i.e., the signature and the signed document are separate files. In certain applications it is necessary to use detached signatures because, e.g., it may not be possible to modify the original document. However, the

AdES formats do not provide a unified way to couple together the detached signature and the signed document; ASiC fills this gap.

An ASiC container is a regular ZIP file with a specific file structure. The standard [ETS16b] defines two kinds of ASiCs: the simple ASiC (ASiC-S) and the extended ASiC (ASiC-E). The main difference is that ASiC-S can contain only one signed document whereas the ASiC-E may contain multiple signed documents and their corresponding signatures.

As the ASiC standard does not use any cryptographic methods to achieve its goal, there is nothing to analyze with respect to its security against quantum adversaries.

## 6.2 Purely Post-Quantum Advanced Electronic Signatures

In this section, we present *purely post-quantum* AdES signatures. The integration of purely post-quantum signature schemes into AdES is straightforward, as the relevant standards [ETS22; ETS24a; ETS23b; ETS24b] are cryptographically agile in the sense that they are not dependent on a specific signature scheme.

The only relevant difference between classical and post-quantum signatures is that post-quantum signatures have larger public keys and produce larger signatures. This is not an issue as AdES standards do not restrict sizes of public keys or signatures. What remains is to integrate these new post-quantum signatures into existing libraries and standardize their identifiers.

On the other hand, an AdES signature is not only the raw signature of the signer's document. Its main purpose is to encapsulate additional properties tied to that signature such as certificates, timestamps etc. To obtain a proper purely post-quantum AdES signature, these components also need to be secured against quantum adversaries. Recall from Section 6.1 that these properties are:

- X.509 certificates,
- revocation values in the form of OCSP responses or CRLs,
- and timestamps.

Post-quantum solutions to PKI-related primitives such as revocation and certificates are presented in Chapter 3.

### 6.2.1 Example Purely Post-Quantum XAdES B-LTA

Listing 6.9 is an example XAdES B-LTA enveloping signature signed using the post-quantum ML-DSA-44 algorithm.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ds:Signature Id="id-1" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2023/02/xmldsig-pqc#ml-dsa-44"/>
      <ds:Reference Id="r-id-1-1" Type="http://www.w3.org/2000/09/xmldsig#Object" URI="#o-r-id-1-1">
         <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
         </ds:Transforms>
         <ds:DigestMethod Algorithm="http://www.w3.org/2007/05/xmldsig-more#sha3-512"/>
         <ds:DigestValue><!-- SHA3-512 digest of input document --></ds:DigestValue>
      </ds:Reference>
```

```
      <ds:Reference Type="http://uri.etsi.org/01903#SignedProperties" URI="#xades-id-1">
         <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         </ds:Transforms>
         <ds:DigestMethod Algorithm="http://www.w3.org/2007/05/xmldsig-more#sha3-512"/>
         <ds:DigestValue><!-- SHA3-512 digest of <SignedProperties> --></ds:DigestValue>
      </ds:Reference>
   </ds:SignedInfo>
   <ds:SignatureValue Id="value-id-1"><!-- ML-DSA-44 signature --></ds:SignatureValue>
   <ds:KeyInfo>
      <ds:X509Data>
         <!-- ML-DSA-44 X.509 certificate chain -->
      </ds:X509Data>
   </ds:KeyInfo>
   <ds:Object>
      <xades:QualifyingProperties Target="#id-1" xmlns:xades="http://uri.etsi.org/01903/v1.3.2#">
         <xades:SignedProperties Id="xades-id-1"><!-- standard signed properties --></xades:SignedProperties>
         <xades:UnsignedProperties>
            <xades:UnsignedSignatureProperties>
               <xades:SignatureTimeStamp Id="ts-id-1">
                  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  <xades:EncapsulatedTimeStamp Id="ets-id-1"><!-- ML-DSA-44 timestamp--></xades:EncapsulatedTimeStamp>
               </xades:SignatureTimeStamp>
               <xades:CertificateValues><!-- standard certificate values except containing ML-DSA-44 certificates --></
                    xades:CertificateValues>
               <xades:RevocationValues><!-- standard revocation values except containing ML-DSA-44 OCSP responses/CRLs -->
                    </xades:RevocationValues>
               <xades141:TimeStampValidationData Id="tsvd-id-1" xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
               <!-- standard TimeStampValidationData values except containing ML-DSA-44 protected values-->
               </xades141:TimeStampValidationData>
               <xades141:ArchiveTimeStamp Id="ts-id-2" xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
                  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  <xades:EncapsulatedTimeStamp Id="ets-id-2"><!-- ML-DSA-44 archival timestamp --></xades:
                       EncapsulatedTimeStamp>
               </xades141:ArchiveTimeStamp>
            </xades:UnsignedSignatureProperties>
         </xades:UnsignedProperties>
      </xades:QualifyingProperties>
   </ds:Object>
   <ds:Object Id="o-r-id-1-1"><!-- input document --></ds:Object>
</ds:Signature>
```

**Listing 6.9**: Purely Post-Quantum XAdES B-LTA Signature Example.

We do not provide visual examples for CAdES and PAdES, as there is no compact way to visualize these binary formats. For JAdES, we provide only an example of level B-LT, as the archiving level (B-LTA) is not fully specified at the time of writing due to the lack of a standard canonicalization algorithm for JSON.

### 6.2.2 Example Purely Post-Quantum JAdES B-LT

Listing 6.10 is an example JAdES B-LT signature signed using the post-quantum ML-DSA-44 algorithm. The protected header is Base64-encoded which complicates visualization; therefore, we present a decoded example separately in Listing 6.11.

```
{
   "payload": "<input document>",
   "protected": "<protected header header, see other listing for JSON structure>",
   "header":
   {
      "etsiU":
      [
         {
```

```
            "sigTst":
            {
               "tstTokens":
               [
                  {
                     "val": "<ML-DSA-44 timestamp>"
                  }
               ]
            }
         },
         {
            "xVals":
            [
               {
                  "x509Cert":
                  {
                     "val": "<ML-DSA-44 certificate>"
                  }
               }
            ]
         },
         {
            "rVals":
            {
               "ocspVals":
               [
                  {
                     "val": "<ML-DSA-44 OCSP response>"
                  },
                  {
                     "val": "<ML-DSA-44 OCSP response>"
                  }
               ]
            }
         },
         {
            "tstVD":
            {
               "xVals":
               [
                  {
                     "x509Cert":
                     {
                        "val": "<ML-DSA-44 timestamp certificate>"
                     }
                  }
               ],
               "rVals":
               {
                  "ocspVals":
                  [
                     {
                        "val": "<ML-DSA-44 OCSP timestamp response>"
                     }
                  ]
               }
            }
         }
      ]
   },
   "signature": "<ML-DSA-44 signature value>"
}
```

**Listing 6.10**: Purely Post-Quantum JAdES B-LT Signature Example.

```
{
   "alg": "ML-DSA-44",
   "cty": "text/xml",
```

```
   "kid": "MGIwXaRbMFkxF....",
   "x5t#o":
   {
      "digAlg": "S512",
      "digVal": "OCt591k-KegxZ-A_oqjVFvxkeXEsUslvFpfGs6oAaPEOyJPB4iKB5wWgxu7BHScYS_AG4_24IspGK3gGKQh_cA"
   },
   "x5c":
   [
      "<ML-DSA-44 signing certificate>"
   ],
   "typ": "jose+json",
   "iat": 1739976181
}
```

**Listing 6.11:** Example of a decoded protected header from a purely post-quantum JAdES B-LT.

## 6.3 Hybrid Post-Quantum Advanced Electronic Signatures

This section presents two possible approaches to achieving hybrid post-quantum security in AdES. Similar to our hybrid PKI approaches presented in Section 3.3, we present *composite hybrid post-quantum AdES* in Section 6.3.1, and alternative *backward-compatible post-quantum AdES* in Section 6.3.2.

### 6.3.1 Composite Hybrid Post-Quantum AdES

In this section, we introduce the hybrid post-quantum solution using a composite hybrid signature. Recall from Section 2.4.5.1 that a composite hybrid signature is a signature scheme where the public keys and signatures are concatenations (there can be other encodings) of two component signatures. The composite signature is constructed by signing the data with each component separately and concatenating the result. During verification, the verification algorithm is run separately on both components and returns success if and only if each component verifies successfully.

The main benefit of composite signatures is that it is a generic standalone signature scheme (as its components) and can be used in place of a classical signature while achieving hybrid post-quantum security, i.e., as long as the classical or the post-quantum signature is unforgeable, the hybrid remains unforgeable.

Consequently, composite hybrid signatures make the problem of finding a hybrid post-quantum solution equivalent to finding a purely post-quantum solution (Section 6.2). The only difference is that their public keys and signatures are *slightly* larger[15] compared to purely post-quantum signatures. Recall from Section 6.2 that AdES-related specifications do not restrict sizes of public keys or signatures. Therefore, the size increase is not an issue.

From an implementation perspective, composite hybrids also have the benefit that they can be implemented in a low-level cryptographic library (at the same level where classical signatures are implemented) and the higher level library only needs to be updated as if it was adding support for another regular signature. This is not true for other hybrid solutions which additionally require protocol/format modifications.

[15] Composites have increased size due to the extra classical component, but that one is typically significantly smaller compared to the post-quantum one.

The main drawback of these composite solutions is that they are not "backward compatible." Backward compatibility in the context of AdES means that the signer wants to issue AdES signatures that are hybrid post-quantum secure, but the signer wants that the signature is

- verifiable by a post-quantum verifier, i.e., a verifier able to process classical and post-quantum signatures,
- verifiable by a classical-only verifier, i.e., a verifier unable to process post-quantum signatures, at the cost of not achieving hybrid post-quantum security, but at least classical security.

Backward compatibility captures the realistic scenario where the signer does not have to worry about the capabilities of verifiers, e.g., some verifiers have already upgraded their systems to support post-quantum cryptography, but some have not.

The details of using composite signatures inside CMS is also still being discussed at the time of writing [Oun+25a], we recommend following the relevant RFCs once they are published.

**Remark**

We do not provide examples of composite hybrid post-quantum AdES signatures as their structure is identical to purely post-quantum AdES signatures from Section 6.2 which contains examples. In Section 6.4, we investigate the performance of these signatures.

### 6.3.2 Backward-Compatible Hybrid Post-Quantum AdES

Although Section 6.3.1 describes how to achieve hybrid post-quantum AdES, that solution is not backward compatible. Specifically, we consider a hybrid post-quantum AdES to be backward compatible if the resulting signature can be *partially* verified by a classical-only verifier. Such a verifier will not be able to verify the composite hybrid AdES as the signature algorithm is unknown to the verifier. Of course, the classical-only verifier is not protected against quantum adversaries, but the point is that hybrid verifiers, i.e., those who also support post-quantum signatures, are protected.

We begin this section by evaluating potential solutions to achieving backward-compatible hybrid post-quantum AdES using some existing mechanisms of AdES in Section 6.3.2.1. We then focus on one such solution using archival timestamps in Section 6.3.2.2.

#### 6.3.2.1 Non-Viable Solutions Using Existing AdES Mechanisms

This section discusses potential solutions using existing AdES mechanisms "repurposed" to achieve hybrid post-quantum security in AdES. We highlight the drawbacks of these potential solutions.

**Non-Archival Timestamps.** Consider a basic **B-B** level AdES signature containing a classical signature. AdES supports inclusion of timestamps as a method to certify that the signature existed at a given point in time. Under the hood, a timestamp is another signature where the signed data is the AdES signature value together with time. The timestamp cryptographically protects only the signature value and not the whole AdES signature, which contains the hash of the signer's document. Consequently, the timestamp signs the signer's document "indirectly."

Now consider applying a purely post-quantum timestamp (timestamp where the signature algorithm is a purely post-quantum signature) to a classical **B-B** AdES signature. A **B-B** level with a timestamp is a **B-T** level. According to **B-T** validation rules, the classical signature of the signer's document and the post-quantum timestamp both need to be valid for the **B-T** signature to be valid. Therefore, even if a quantum adversary is able to forge the classical signature, it will not be able to forge the post-quantum timestamp. On the other hand, if the post-quantum signature inside the timestamp becomes classically broken (but not the classical signature), a classical adversary is unable to forge the classical signature. Therefore, the adversary still cannot forge the **B-T** signature.

Consequently, we could consider a **B-T** AdES signature, with a post-quantum timestamp, a hybrid post-quantum signature. However, there are a few issues with this approach.

- This indirect signing by the timestamp is an example of a *weakly nested* hybrid signature (Definition 25). Recall from Section 2.4.5.1, that this construction does not guarantee hybrid security.
- Consider a case where a "backward-compatible" solution is used, i.e., a classical-only verifier will be able to validate the hybrid signature. The solution is to include a classical timestamp and a post-quantum timestamp. However, the classical-only verifier will consider the post-quantum timestamp invalid as it is an unknown algorithm for the verifier. Although the standard [ETS16a] does not demand it, certain validation policies may require all timestamps to be valid; therefore, the hybrid signature will be considered invalid.

**Remark**

Technically, PAdES supports two ways how to apply timestamps. If the timestamp is a separate **DocTimeStamp** object, then the timestamp signature covers the existing signature and the document—making it a *strongly nested* hybrid signature. However, this is only specific to PAdES, so we do not consider it as a generic solution.

**Content Timestamps.** AdES formats also support timestamps applied *only* to the signer's document (as opposed to the signature) called *content timestamps*, see `AllDataObjectsTimeStamp` in XAdES [ETS22] and `content-time-stamp` in CAdES/PAdES [ETS23b; ETS09]. These timestamps are *signed properties*, i.e., they need to be computed before the AdES signature is produced.

Consider a solution where the signer's document is first signed by a post-quantum signature to produce the content timestamp which is then included as a signed property. The classical signature is then applied over the signer's document and the signed properties which includes the content timestamp. From a security perspective, this is a viable solution as it is an instance of strongly nested hybrid signature, and it is also non-separable (see Section 2.4.5.1).

Nonetheless, it is not a *clean solution* as it reuses existing properties which may cause inconsistent behavior. Consider a classical-only verifier with a policy requiring content-timestamps to be valid if included. In that case, it will consider the AdES signature invalid as it will not recognize the post-quantum algorithm. More importantly, the post-quantum signature only protects the signer's document and not the potential metadata included in the AdES signature itself.

**Conclusion**

For reasons described above, we do not consider using (regular) timestamps as an appropriate way to achieve hybrid post-quantum AdES signatures.

**Countersignatures.** It was not mentioned in Section 6.1, but AdES formats also support so-called "countersignatures." A countersignature is generally an additional signature added to an existing signature. Countersignatures in AdES work similarly to timestamps discussed in Section 6.3.2.1 in the sense that the countersignature is calculated only over the existing signature and without the signer's document (see [ETS22, Section 5.2.7]).

Using a purely post-quantum countersignature would be in a sense cleaner compared to a purely post-quantum timestamp, because the time information inside a timestamp is redundant (only the signature is needed for hybrid security). However, this solution is also an instance of a *weakly nested* hybrid signature (Definition 25), which suffers from the same security issues as described in Section 6.3.2.1.

**Conclusion**

Countersignatures, as defined in the existing AdES standards, are not an appropriate way to achieve hybrid post-quantum AdES signatures.

#### 6.3.2.2 Viable Solution: Archival Timestamps

Recall from Section 6.1, that AdES supports so-called *archival timestamps*. Compared to a regular timestamp (Section 6.3.2.1), an archival timestamp applies its signature over the whole AdES structure, which includes the existing signature of the signer's document and also the signer's document, or its digest. This means that an AdES signature protected by a classical signature, that is augmented by a post-quantum archival timestamp, is a hybrid post-quantum signature of the signer's document. Specifically, it is an instance of a *strongly nested* hybrid post-quantum signature (Section 2.4.5.1). Strongly nested hybrids are proven to be secure hybrids under standard unforgeability assumptions as opposed to the *weakly nested* hybrids (cf. Section 2.4.5.1).

**Limitation.** A hybrid solution using archival timestamps does not suffer from the same security issues as the solution using regular timestamps (Section 6.3.2.1). However, it is not a "general solution" to hybrid post-quantum AdES as the inclusion of an archival timestamp into an AdES signature makes it into a **B-LTA** level signature:

- XAdES specification [ETS22] requirement "aa)" in Section 6.3 states:

  *Before generating and incorporating a new ArchiveTimeStamp qualifying property defined in the namespace whose URI is "http://uri.etsi.org/01903/v1.4.1#", all the validation material required for validating the XAdES signature shall be included.*

- CAdES specification [ETS23b] requirement "s)" in Section 6.3 states:

  *Before generating and incorporating an archive-timestamp-v3 attribute, all the validation material required for verifying the signature, which are not already in the signature, shall*

*be included. This validation material includes validation material used to validate previous archive time stamp.*

- PAdES specification [ETS09] requirement "x)" in Section 6.3 states:

  *Before generating and incorporating a document-time-stamp attribute, applications shall include all the validation material, which are not already in the signature, required for validating the signature. This validation material includes all the certificates and all certificate status information (like CRLs or OCSP responses) required for:...*

**Limitation Impact.** As explained in the previous section, this method cannot produce "lower level" (**B-B**, **B-T** and **B-LT**) hybrid post-quantum AdES. However, the fact that using post-quantum archival timestamps, as a backward-compatible method for achieving hybrid post-quantum security, necessarily makes it a **B-LTA** level signature, is not really a limitation in practice. This is because end-entity certificates have an expected lifetime between one and three years. For example, the validity of a **B-B** level signature cannot be assessed after the signing certificate expires as there is no proof the signature was not created using a compromised private key.

Similarly, **B-T** level contains additionally a timestamp, which proves that the *signature value* existed at a certain point in time. Therefore, if the validator has access to revocation data, that prove validity of the signing certificate at the timestamp's time, then it serves as a proof of validity of the signature, assuming the timestamp authority's certificate is valid at validation time. The issue is that such revocation data may not be available to a future validator. That is why the **B-LT** level is necessary for AdES signatures with an expected lifetime beyond the signing certificate's lifetime.

*Migration Timeline.* The sufficiency of this solution (post-quantum archival timestamps) relies on the assumption that producers of AdES signatures with expected lifetimes of a few years (levels **B-B** and **B-T**) begin using purely post-quantum solutions before quantum computers become a threat. Meanwhile, signatures with levels **B-LT** and **B-LTA** can be augmented with a post-quantum archival timestamp to gain hybrid post-quantum security.

More precisely, let us denote by $T_{QC}$ the point in time when quantum adversaries become a threat, by $L_B$ the expected lifetime of short-term signatures (**B-B** and **B-T**) and by $T_{PQ}$ the point in time when purely post-quantum signatures become trusted enough that hybrid post-quantum signatures are no longer necessary, then we assume that $T_{PQ} + L_B < T_{QC}$[16] for our solution to be sufficient. See Figure 6.9 for visualization.

**Concrete Solution.** There is also an important decision to make. Consider that we have a classical **B-LT** signature, and we want to apply the post-quantum archival timestamp. As noted above, we need to include the missing validation material, which is presumably also classical. The question is then, do we apply a classical archival timestamp before applying the post-quantum archival timestamp, or do we apply only the post-quantum one?

If we apply only a post-quantum archival timestamp, then the signature technically has a **B-LTA** structure, but a classical-only verifier will not be able to verify the archival timestamp. Depending on the validation policies, the verifier may consider it a valid **B-LT** signature (ignoring

[16] This is essentially a version of Mosca's equation.

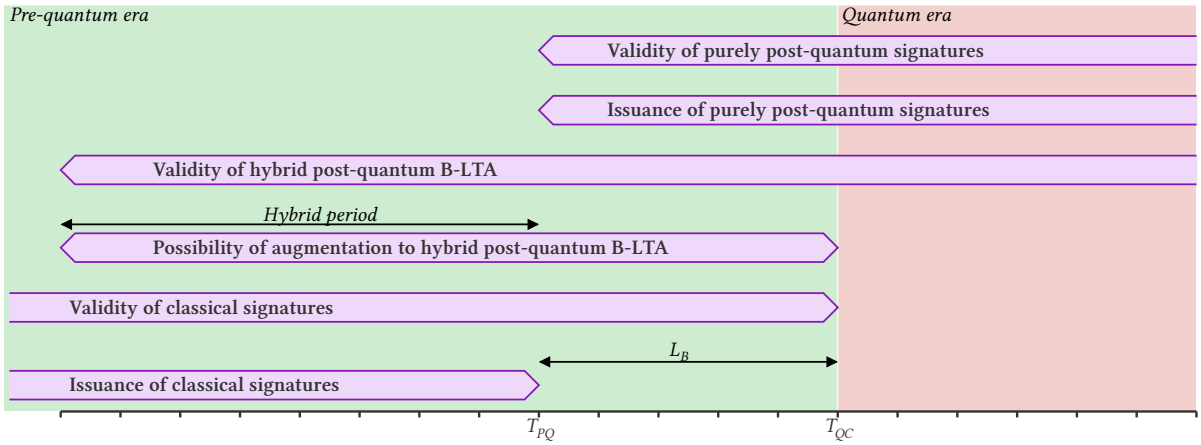



**Figure 6.9:** Transition timeline and the role of hybrid post-quantum **B-LTA** signatures.

the archival timestamp), or it may consider the signature invalid as it has a structure of a **B-LTA** signature, but the archival timestamp is invalid.

Omitting the classical archival timestamp also affects security. When the classical algorithm becomes compromised, then for a post-quantum verifier it is a valid **B-LTA** signature. On the other hand, if the post-quantum signature becomes compromised, then it is *at most*[17] a valid **B-LT** signature.

The inclusion of both the classical and post-quantum archival timestamps therefore provides expected security guarantees: if one of the algorithms becomes compromised, it is still a valid **B-LTA** for a hybrid verifier. Additionally, it is reasonable to assume that it improves backward compatibility as the classical-only verifier might reject a signature containing only one archival timestamp that is not valid from its point of view. If a classical-only verifier encounters two archival timestamps, one classical and one post-quantum, it will most likely simply ignore the one it cannot process/validate and use the time from the classical archival timestamp as the *best-signature-time*.[18] Of course, the exact behavior of the verifier depends on the validation policies, however, it is reasonable to assume that this is the default behavior. For example, this is how the official EU-provided validator [Eur25c] behaves.

**Conclusion**

We recommend using the post-quantum archival timestamp to provide hybrid post-quantum security. Specifically, we recommend to first augment the signature to a classical **B-LTA** level, and then apply the additional post-quantum archival timestamp. This solution provides proper hybrid post-quantum security and maximizes backward compatibility.

[17] Depending on the verifier's behavior upon encountering invalid archival timestamp.

[18] The term *best-signature-time* refers to the earliest point in time at which the existence of the signature can be proven. [ETS16a]

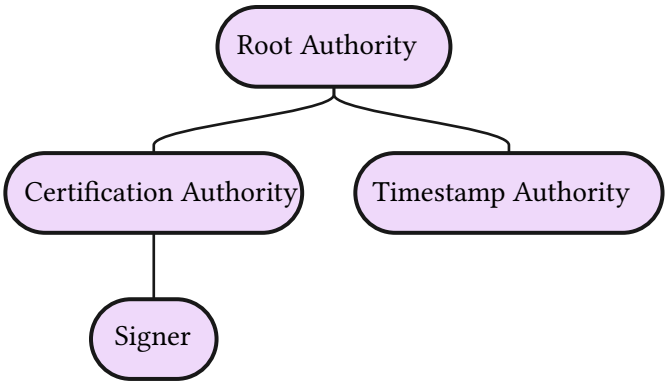



**Figure 6.10:** Experimental PKI Structure

## 6.4 Performance of Post-Quantum AdES

In this section, we present the performance of the proposed post-quantum AdES solutions in Sections 6.2 and 6.3. By performance, we refer to speed and size. Our specific configuration may not directly apply to real-world instances of AdES because there are many possible AdES configurations, so evaluating all of them is infeasible. Additionally, performance is typically not the most important factor for AdES signatures. This is because the creation and validation of AdES signatures are not as common as TLS handshakes.

### 6.4.1 Methodology

We used a simple PKI during our performance evaluation—its structure is visualized in Figure 6.10. Recall from Section 6.3.2 that to realize backward-compatible hybrid AdES, we require two PKIs: one classical and one post-quantum. They are separate and have the same structure.

We use OCSP as the revocation checking mechanism; the trust anchor (Root Authority) signs OCSP responses regarding the validity of the CA and the TSA while the CA signs OCSP responses stating the validity of the signer.

For each AdES format, we use the *enveloping* variant as it is supported by all tested formats. For XAdES and CAdES the input document is a 103-byte XML file, whereas for PAdES, it is a 39 044-byte PDF file.

Our experimental implementation uses forks of the DSS library, Apache Santuario, and BouncyCastle. Note that we do not test JAdES because its DSS implementation does not use BouncyCastle but rather a standalone JWS/JWT library, which makes integrating new algorithms more challenging. On top of that, the **B-LTA** level JAdES is not yet implemented in DSS.

For more details, please refer to the proof-of-concept implementation [Oup25a]. For example, there are many places where hash functions are used. We use 512-bit SHA-3/SHAKE256 with non-composite post-quantum signatures as ML-DSA internally uses SHAKE256 whereas we use SHA-256 for classical and composite hybrids. Recall from Section 1.4.2 that SHA-2 and SHA-3 are considered equally strong, however, SHA-3 is newer and is an integral part of many post-quantum algorithms. That is why we pair SHA-3 with post-quantum and SHA-2 with classical signatures.

**Remark**

SHAKE256 is an XOF (Definition 8), i.e., a hash function with configurable output length. SHAKE256 and the SHA-3 family are closely related as their main internal component is the Keccak permutation. A 512-bit SHAKE256 is between SHA3-256 and SHA3-512 in terms of security. For more details refer to [NIS15b].

Starting with classical signatures, we tested 256-bit ECDSA with SHA-256. Moving to (non-composite) post-quantum signatures, we tested the currently standardized (pure) ML-DSA-44 and (pure) SLH-DSA-SHAKE-256S and SLH-DSA-SHAKE-256F. ML-DSA-44 provides the lowest, but still acceptable, level of post-quantum security while the SHAKE256-based SLH-DSA variant represents the most conservative option possible (NIST level 5 security and based on SHAKE256). Lastly, for composite hybrid post-quantum signatures, we tested the combination of ML-DSA-44 with 256-bit ECDSA, both with SHA-256 "pre-hashing."

We ran 1000 repetitions (with 100 repetitions as a warm-up to minimize variance due to JVM optimizations) for each configuration. The machine used for the benchmark was a Google Cloud VPS with the ARM CPU Ampere Altra Q64-30 3 GHz on Ubuntu 24.04.

### 6.4.2 Results

We present results of our experiment in this section. Recall that backward-compatible hybrids (Section 6.3.2.2) only exist for the **B-LTA** level. We focus our further discussion on CAdES as it is the most efficient format; therefore, the results have the least noise.

#### 6.4.2.1 Size

Tables 6.1 to 6.3 show the resulting sizes of XAdES, CAdES and PAdES signatures, respectively. We see that using purely post-quantum AdES results in a significant increase in size compared to classical AdES, e.g., an ECDSA **B-LTA** CAdES takes up 11 kB, while an ML-DSA-44 one takes up 68 kB. Using the most conservative SLH-DSA-SHAKE-256 results in sizes of 0.5 MB for the "small" variant and nearly 1 MB for the "fast" variant.

Composite hybrid post-quantum signatures are obviously slightly larger than purely post-quantum ones. On the other hand, we can see that the backward-compatible hybrid post-quantum solution using archival timestamps is more size efficient than using a purely post-quantum signature. This is because most of the cryptographic material remains classical, such as certificates and revocation values. The only post-quantum signatures are inside the archival timestamp which consists of the timestamp signature and two certificates (the TSA's and the trust anchor's). Of course, when the archival timestamp expires and needs to be refreshed, it will require the addition of the certificates and revocation values from the post-quantum PKI, which will inflate the size significantly.

Note that Table 6.3 reports the same sizes for **B-B** and **B-T** levels. This is because the size of the resulting PAdES signature is pre-allocated, and we overestimated this value for some of the tested algorithms. Therefore, in theory it can be made smaller. Since PAdES is closely tied to CAdES, we recommend looking at CAdES values (Table 6.2) for better accuracy as it is the most size-efficient format.

**Table 6.1**: Post-Quantum XAdES Sizes

| Algorithm | XAdES-B-B | XAdES-B-T | XAdES-B-LT | XAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 4 571 | 7 484 | 14 358 | 17 200 |
| ML-DSA-44 | 17 397 | 33 080 | 78 118 | 93 731 |
| ML-DSA-65 | 22 657 | 43 601 | 104 428 | 125 302 |
| ML-DSA-87 | 29 641 | 57 565 | 139 332 | 167 186 |
| SLH-DSA (128,f) | 72 566 | 143 452 | 353 975 | 424 791 |
| SLH-DSA (128,s) | 35 638 | 69 597 | 169 332 | 203 220 |
| SLH-DSA (256,f) | 203 810 | 405 902 | 1 010 185 | 1 212 207 |
| SLH-DSA (256,s) | 123 554 | 245 390 | 608 885 | 730 651 |
| Comp. ECDSA + ML-DSA-44 | 17 915 | 34 141 | 80 643 | 96 797 |
| Comp. ECDSA + ML-DSA-65 | 23 484 | 45 229 | 108 430 | 130 104 |
| Comp. ECDSA + ML-DSA-87 | 30 464 | 59 187 | 143 328 | 171 979 |
| ECDSA + ML-DSA-44 | | | | 37 977 |
| ECDSA + ML-DSA-65 | | | | 43 230 |
| ECDSA + ML-DSA-87 | | | | 50 184 |
| ECDSA + SLH-DSA (128,f) | | | | 93 163 |
| ECDSA + SLH-DSA (128,s) | | | | 56 229 |
| ECDSA + SLH-DSA (256,f) | | | | 224 371 |
| ECDSA + SLH-DSA (256,s) | | | | 144 116 |

Sizes are in bytes. ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

**Table 6.2**: Post-Quantum CAdES Sizes

| Algorithm | CAdES-B-B | CAdES-B-T | CAdES-B-LT | CAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 2 050 | 3 944 | 8 565 | 11 039 |
| ML-DSA-44 | 11 674 | 23 144 | 56 392 | 68 446 |
| ML-DSA-65 | 15 621 | 31 038 | 76 130 | 92 129 |
| ML-DSA-87 | 20 855 | 41 506 | 102 300 | 123 532 |
| SLH-DSA (128,f) | 53 046 | 105 924 | 263 281 | 316 736 |
| SLH-DSA (128,s) | 25 350 | 50 527 | 124 797 | 150 556 |
| SLH-DSA (256,f) | 151 482 | 302 765 | 755 446 | 907 307 |
| SLH-DSA (256,s) | 91 289 | 182 379 | 454 471 | 546 140 |
| Comp. ECDSA + ML-DSA-44 | 12 043 | 23 921 | 58 263 | 70 724 |
| Comp. ECDSA + ML-DSA-65 | 16 230 | 32 247 | 79 115 | 95 713 |
| Comp. ECDSA + ML-DSA-87 | 21 465 | 42 716 | 105 289 | 127 120 |
| ECDSA + ML-DSA-44 | | | | 26 743 |
| ECDSA + ML-DSA-65 | | | | 30 684 |
| ECDSA + ML-DSA-87 | | | | 35 927 |
| ECDSA + SLH-DSA (128,f) | | | | 68 104 |
| ECDSA + SLH-DSA (128,s) | | | | 40 440 |
| ECDSA + SLH-DSA (256,f) | | | | 166 562 |
| ECDSA + SLH-DSA (256,s) | | | | 106 356 |

Sizes are in bytes. ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

**Table 6.3:** Post-Quantum PAdES Sizes

| Algorithm | PAdES-B-B | PAdES-B-T | PAdES-B-LT | PAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 50 012 | 50 012 | 56 795 | 63 910 |
| ML-DSA-44 | 92 015 | 92 015 | 134 632 | 165 752 |
| ML-DSA-65 | 110 017 | 110 017 | 167 536 | 208 656 |
| ML-DSA-87 | 130 017 | 130 017 | 207 156 | 258 276 |
| SLH-DSA (128,f) | 260 017 | 260 017 | 453 518 | 574 640 |
| SLH-DSA (128,s) | 160 017 | 160 017 | 251 960 | 313 080 |
| SLH-DSA (256,f) | 660 016 | 660 016 | 1 214 441 | 1 535 563 |
| SLH-DSA (256,s) | 420 016 | 420 016 | 753 723 | 954 843 |
| Comp. ECDSA + ML-DSA-44 | 92 015 | 92 015 | 136 082 | 167 201 |
| Comp. ECDSA + ML-DSA-65 | 110 017 | 110 017 | 169 796 | 210 915 |
| Comp. ECDSA + ML-DSA-87 | 130 017 | 130 017 | 209 421 | 260 540 |
| ECDSA + ML-DSA-44 | | | | 105 503 |
| ECDSA + ML-DSA-65 | | | | 115 526 |
| ECDSA + ML-DSA-87 | | | | 125 523 |
| ECDSA + SLH-DSA (128,f) | | | | 195 557 |
| ECDSA + SLH-DSA (128,s) | | | | 135 521 |
| ECDSA + SLH-DSA (256,f) | | | | 395 526 |
| ECDSA + SLH-DSA (256,s) | | | | 275 521 |

Sizes are in bytes. ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

#### 6.4.2.2 Signing Speed

Tables 6.4 to 6.6 show the signing speeds for each format and level.

We see that ML-DSA-44 sometimes outperformed ECDSA—this is most likely due to BouncyCastle's implementation as typically ECDSA over P-256 is faster than ML-DSA-44 at signing. We also see the significant difference between SLH-DSA "fast" and "small" variants. Recall from Table 6.2 that the small variant provides around 50% smaller signatures, but here we see that the size savings come at a significant speed cost: the small variant is 10–20× slower at signing. Note that even the fast version is over 100× slower than ML-DSA.

On the other hand, if SLH-DSA is used only in the archival timestamp (where there is only one invocation of SLH-DSA signing), then the fast variant is comparable to ML-DSA-44. Even with the most conservative *small variant* it takes around 1–2 seconds to apply the archival timestamp. We presume that in real-world scenarios, where there are non-ideal network conditions, the impact of the slow signing speed would be smaller compared to our results.

#### 6.4.2.3 Verification Speed

Tables 6.7 to 6.9 show verification speed results. The verification speed of ECDSA and ML-DSA-44 is comparable whereas SLH-DSA is still significantly slower, but the difference is not as large as for signing.

Additionally, we observe a strange artifact with the small variant of SLH-DSA. The verification speed of lower levels such as **B-B** and **B-T** is significantly slower than of the higher levels (**B-LT**

**Table 6.4:** Post-Quantum XAdES Signing Speed

| Algorithm | XAdES-B-B | XAdES-B-T | XAdES-B-LT | XAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 3,89 ± 0,86 | 7,79 ± 0,57 | 19,47 ± 0,63 | 21,90 ± 0,55 |
| ML-DSA-44 | 3,08 ± 0,55 | 7,93 ± 0,67 | 21,05 ± 1,01 | 26,84 ± 1,10 |
| ML-DSA-65 | 3,37 ± 0,69 | 8,78 ± 1,00 | 26,08 ± 1,43 | 33,42 ± 1,60 |
| ML-DSA-87 | 3,65 ± 0,77 | 9,67 ± 1,04 | 32,24 ± 1,88 | 40,84 ± 1,78 |
| SLH-DSA (128,f) | 64,11 ± 1,60 | 134,10 ± 0,76 | 407,31 ± 1,09 | 480,10 ± 1,36 |
| SLH-DSA (128,s) | 1 366,66 ± 2,25 | 2 737,16 ± 3,10 | 6 862,94 ± 5,98 | 8 234,73 ± 5,80 |
| SLH-DSA (256,f) | 209,60 ± 0,60 | 431,12 ± 0,94 | 1 178,04 ± 1,41 | 1 425,45 ± 1,61 |
| SLH-DSA (256,s) | 2 020,73 ± 3,52 | 4 048,32 ± 4,68 | 10 169,22 ± 7,65 | 12 213,83 ± 8,60 |
| Comp. ECDSA + ML-DSA-44 | 3,28 ± 0,44 | 8,64 ± 0,67 | 28,81 ± 1,03 | 34,44 ± 1,11 |
| Comp. ECDSA + ML-DSA-65 | 3,81 ± 0,70 | 10,78 ± 1,00 | 46,48 ± 1,68 | 54,14 ± 1,76 |
| Comp. ECDSA + ML-DSA-87 | 4,08 ± 0,71 | 11,68 ± 1,10 | 53,00 ± 1,68 | 61,65 ± 1,71 |
| ECDSA + ML-DSA-44 | | | | 48,41 ± 0,98 |
| ECDSA + ML-DSA-65 | | | | 48,32 ± 0,97 |
| ECDSA + ML-DSA-87 | | | | 48,56 ± 1,05 |
| ECDSA + SLH-DSA (128,f) | | | | 109,26 ± 0,82 |
| ECDSA + SLH-DSA (128,s) | | | | 1 412,70 ± 2,40 |
| ECDSA + SLH-DSA (256,f) | | | | 256,46 ± 2,40 |
| ECDSA + SLH-DSA (256,s) | | | | 2 066,35 ± 5,63 |

Signing times are in milliseconds (± standard deviation). ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

**Table 6.5:** Post-Quantum CAdES Signing Speed

| Algorithm | CAdES-B-B | CAdES-B-T | CAdES-B-LT | CAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 0,71 ± 0,06 | 3,39 ± 0,39 | 12,88 ± 0,33 | 14,05 ± 0,28 |
| ML-DSA-44 | 0,60 ± 0,06 | 3,22 ± 0,48 | 13,32 ± 0,84 | 15,53 ± 0,87 |
| ML-DSA-65 | 1,71 ± 1,16 | 4,18 ± 0,79 | 18,57 ± 1,68 | 21,97 ± 2,11 |
| ML-DSA-87 | 1,17 ± 0,13 | 5,14 ± 0,88 | 24,30 ± 1,78 | 27,82 ± 2,08 |
| SLH-DSA (128,f) | 60,99 ± 0,20 | 130,74 ± 0,41 | 398,16 ± 0,94 | 463,66 ± 2,04 |
| SLH-DSA (128,s) | 1 363,78 ± 2,44 | 2 732,13 ± 4,44 | 6 854,14 ± 5,69 | 8 351,32 ± 6,15 |
| SLH-DSA (256,f) | 205,97 ± 0,60 | 425,46 ± 0,88 | 1 166,69 ± 1,52 | 1 398,61 ± 1,62 |
| SLH-DSA (256,s) | 2 017,18 ± 3,94 | 4 043,24 ± 4,80 | 10 159,10 ± 7,90 | 12 327,44 ± 9,14 |
| Comp. ECDSA + ML-DSA-44 | 1,18 ± 0,33 | 4,42 ± 0,43 | 21,62 ± 0,86 | 24,08 ± 1,09 |
| Comp. ECDSA + ML-DSA-65 | 1,86 ± 0,38 | 7,54 ± 0,85 | 40,67 ± 1,57 | 44,46 ± 1,74 |
| Comp. ECDSA + ML-DSA-87 | 1,64 ± 0,14 | 8,35 ± 1,11 | 46,77 ± 1,64 | 50,97 ± 1,67 |
| ECDSA + ML-DSA-44 | | | | 39,60 ± 0,68 |
| ECDSA + ML-DSA-65 | | | | 39,76 ± 0,83 |
| ECDSA + ML-DSA-87 | | | | 39,96 ± 0,91 |
| ECDSA + SLH-DSA (128,f) | | | | 100,27 ± 0,65 |
| ECDSA + SLH-DSA (128,s) | | | | 1 425,11 ± 2,89 |
| ECDSA + SLH-DSA (256,f) | | | | 245,71 ± 0,72 |
| ECDSA + SLH-DSA (256,s) | | | | 2 056,29 ± 3,90 |

Signing times are in milliseconds (± standard deviation). ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

**Table 6.6:** Post-Quantum PAdES Signing Speed

| Algorithm | PAdES-B-B | PAdES-B-T | PAdES-B-LT | PAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 4,31 ± 0,70 | 5,97 ± 0,21 | 21,30 ± 0,44 | 26,05 ± 0,43 |
| ML-DSA-44 | 7,03 ± 0,45 | 9,01 ± 0,59 | 29,81 ± 1,15 | 41,27 ± 1,18 |
| ML-DSA-65 | 8,50 ± 0,72 | 11,03 ± 0,99 | 37,90 ± 2,05 | 52,68 ± 1,87 |
| ML-DSA-87 | 10,02 ± 0,76 | 12,96 ± 0,93 | 46,55 ± 1,59 | 64,86 ± 1,70 |
| SLH-DSA (128,f) | 77,99 ± 0,39 | 143,61 ± 0,52 | 440,18 ± 0,96 | 537,28 ± 1,19 |
| SLH-DSA (128,s) | 1 374,66 ± 2,19 | 2 741,30 ± 3,20 | 6 882,95 ± 5,76 | 8 266,47 ± 6,77 |
| SLH-DSA (256,f) | 249,04 ± 0,85 | 461,60 ± 1,04 | 1 276,56 ± 1,57 | 1 580,48 ± 3,05 |
| SLH-DSA (256,s) | 2 045,40 ± 3,28 | 4 068,13 ± 3,96 | 10 234,51 ± 7,60 | 12 310,48 ± 8,23 |
| Comp. ECDSA + ML-DSA-44 | 7,05 ± 0,45 | 9,72 ± 0,64 | 37,36 ± 1,01 | 49,23 ± 1,16 |
| Comp. ECDSA + ML-DSA-65 | 8,95 ± 0,72 | 13,12 ± 0,96 | 59,56 ± 1,92 | 74,19 ± 2,01 |
| Comp. ECDSA + ML-DSA-87 | 10,45 ± 0,70 | 15,11 ± 1,02 | 68,70 ± 1,74 | 86,77 ± 1,88 |
| ECDSA + ML-DSA-44 | | | | 58,87 ± 0,95 |
| ECDSA + ML-DSA-65 | | | | 60,06 ± 2,30 |
| ECDSA + ML-DSA-87 | | | | 60,84 ± 2,11 |
| ECDSA + SLH-DSA (128,f) | | | | 125,03 ± 0,69 |
| ECDSA + SLH-DSA (128,s) | | | | 1 425,01 ± 2,54 |
| ECDSA + SLH-DSA (256,f) | | | | 283,16 ± 0,97 |
| ECDSA + SLH-DSA (256,s) | | | | 2 086,89 ± 3,91 |

Signing times are in milliseconds (± standard deviation). ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

and **B-LTA**). We suspect this is due to some internal caching of the implementation.

## 6.5 Conclusion

This chapter analyzed the quantum vulnerabilities of Advanced Electronic Signatures (AdES) and proposed migration strategies for each AdES format, namely, XAdES, CAdES, PAdES and JAdES. Since the relevant AdES standards are crypto-agile, migration to purely post-quantum AdES is straightforward (assuming the existence of an appropriate post-quantum PKI). However, as with any purely post-quantum solution, it is not backward compatible, i.e., purely post-quantum AdES signatures are not verifiable by classical-only verifiers. To provide backward compatibility with hybrid post-quantum security, we adapt the existing AdES mechanism for archiving: archival timestamps. We showed that one can produce AdES signatures that are hybrid post-quantum-secure and potentially supported by legacy verifiers; we say "potentially" because support ultimately depends on the verifiers' existing validation policies. We have also studied the non-backward-compatible hybrid alternative that uses composite hybrid post-quantum signatures.

We also developed a proof-of-concept implementation of our solutions, which we used to conduct extensive performance benchmarks. Specifically, our results showed that purely post-quantum AdES is comparable to classical AdES in terms of signing and verification speed; in terms of size, however, post-quantum AdES signatures are significantly larger than classical AdES signatures. However, even though AdES contains multiple signatures and public keys in the forms of certificates, revocation data, or timestamps, the relative size increase is smaller than

**Table 6.7**: Post-Quantum XAdES Verification Speed

| Algorithm | XAdES-B-B | XAdES-B-T | XAdES-B-LT | XAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 12,91 ± 1,91 | 14,22 ± 0,76 | 15,75 ± 0,53 | 16,72 ± 1,18 |
| ML-DSA-44 | 11,45 ± 0,65 | 15,04 ± 1,36 | 18,67 ± 1,15 | 20,32 ± 0,97 |
| ML-DSA-65 | 13,11 ± 0,79 | 18,39 ± 1,08 | 23,02 ± 1,58 | 24,99 ± 1,40 |
| ML-DSA-87 | 15,75 ± 1,36 | 22,48 ± 1,17 | 28,88 ± 1,75 | 31,33 ± 1,48 |
| SLH-DSA (128,f) | 161,40 ± 0,57 | 247,59 ± 0,86 | 276,65 ± 1,01 | 225,77 ± 0,85 |
| SLH-DSA (128,s) | 2 746,66 ± 3,24 | 4 120,92 ± 4,33 | 1 399,24 ± 1,49 | 1 403,82 ± 2,25 |
| SLH-DSA (256,f) | 468,01 ± 0,92 | 712,50 ± 1,14 | 762,43 ± 1,59 | 363,37 ± 0,59 |
| SLH-DSA (256,s) | 4 066,75 ± 4,37 | 6 106,62 ± 5,61 | 2 091,09 ± 2,82 | 2 103,47 ± 3,33 |
| Comp. ML-DSA-44 + ECDSA | 14,19 ± 0,78 | 19,82 ± 0,93 | 24,38 ± 0,82 | 26,51 ± 1,19 |
| Comp. ML-DSA-65 + ECDSA | 20,27 ± 1,25 | 30,12 ± 1,38 | 37,40 ± 1,13 | 41,93 ± 1,32 |
| Comp. ML-DSA-87 + ECDSA | 22,74 ± 1,11 | 34,16 ± 1,29 | 43,49 ± 1,70 | 47,35 ± 1,28 |
| ECDSA + ML-DSA-44 | | | | 22,27 ± 0,58 |
| ECDSA + ML-DSA-65 | | | | 23,63 ± 0,70 |
| ECDSA + ML-DSA-87 | | | | 25,57 ± 0,81 |
| ECDSA + SLH-DSA (128,f) | | | | 109,02 ± 1,21 |
| ECDSA + SLH-DSA (128,s) | | | | 1 393,79 ± 2,72 |
| ECDSA + SLH-DSA (256,f) | | | | 269,17 ± 1,52 |
| ECDSA + SLH-DSA (256,s) | | | | 2 059,21 ± 3,60 |

Verification times are in milliseconds (± standard deviation). ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

**Table 6.8**: Post-Quantum CAdES Verification Speed

| Algorithm | CAdES-B-B | CAdES-B-T | CAdES-B-LT | CAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 6,54 ± 0,29 | 8,95 ± 0,29 | 10,58 ± 0,60 | 11,59 ± 0,72 |
| ML-DSA-44 | 6,83 ± 0,38 | 10,52 ± 0,81 | 11,91 ± 0,58 | 13,28 ± 1,07 |
| ML-DSA-65 | 9,41 ± 1,06 | 13,56 ± 0,94 | 15,72 ± 1,12 | 17,40 ± 1,21 |
| ML-DSA-87 | 11,43 ± 0,75 | 17,01 ± 1,11 | 20,04 ± 1,23 | 22,57 ± 1,38 |
| SLH-DSA (128,f) | 155,58 ± 0,66 | 239,81 ± 0,69 | 262,74 ± 0,61 | 274,37 ± 0,62 |
| SLH-DSA (128,s) | 2 742,10 ± 3,08 | 4 114,37 ± 4,02 | 1 390,78 ± 2,74 | 1 395,08 ± 2,86 |
| SLH-DSA (256,f) | 458,62 ± 1,04 | 697,60 ± 1,19 | 305,91 ± 1,24 | 534,35 ± 0,96 |
| SLH-DSA (256,s) | 4 063,01 ± 5,20 | 6 096,04 ± 5,99 | 2 070,97 ± 3,23 | 2 079,12 ± 2,74 |
| Comp. ML-DSA-44 + ECDSA | 10,01 ± 0,52 | 15,21 ± 0,78 | 17,86 ± 0,71 | 19,53 ± 0,73 |
| Comp. ML-DSA-65 + ECDSA | 15,81 ± 0,82 | 24,74 ± 1,17 | 30,16 ± 1,42 | 33,54 ± 1,30 |
| Comp. ML-DSA-87 + ECDSA | 18,50 ± 0,96 | 28,61 ± 1,20 | 34,82 ± 1,17 | 38,45 ± 1,52 |
| ECDSA + ML-DSA-44 | | | | 16,94 ± 0,55 |
| ECDSA + ML-DSA-65 | | | | 18,02 ± 0,50 |
| ECDSA + ML-DSA-87 | | | | 19,62 ± 0,57 |
| ECDSA + SLH-DSA (128,f) | | | | 102,02 ± 0,41 |
| ECDSA + SLH-DSA (128,s) | | | | 1 387,96 ± 2,12 |
| ECDSA + SLH-DSA (256,f) | | | | 258,54 ± 0,62 |
| ECDSA + SLH-DSA (256,s) | | | | 2 051,65 ± 3,70 |

Verification times are in milliseconds (± standard deviation). ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

**Table 6.9:** Post-Quantum PAdES Verification Speed

| Algorithm | PAdES-B-B | PAdES-B-T | PAdES-B-LT | PAdES-B-LTA |
|---|---|---|---|---|
| ECDSA | 30,68 ± 1,06 | 32,49 ± 0,61 | 34,12 ± 0,66 | 56,44 ± 1,11 |
| ML-DSA-44 | 32,05 ± 0,82 | 35,50 ± 1,73 | 37,50 ± 1,18 | 61,61 ± 1,06 |
| ML-DSA-65 | 35,04 ± 1,44 | 39,22 ± 1,29 | 41,97 ± 1,46 | 67,31 ± 1,23 |
| ML-DSA-87 | 37,88 ± 1,63 | 43,85 ± 1,35 | 46,99 ± 1,17 | 74,37 ± 1,74 |
| SLH-DSA (128,f) | 186,34 ± 0,70 | 271,32 ± 0,84 | 294,56 ± 1,12 | 340,94 ± 1,23 |
| SLH-DSA (128,s) | 2 769,37 ± 4,04 | 4 142,01 ± 3,92 | 1 418,66 ± 2,27 | 1 449,80 ± 2,35 |
| SLH-DSA (256,f) | 504,79 ± 1,03 | 745,97 ± 1,50 | 354,54 ± 1,20 | 426,38 ± 1,71 |
| SLH-DSA (256,s) | 4 099,97 ± 4,30 | 6 135,99 ± 5,25 | 2 110,01 ± 3,43 | 2 159,98 ± 3,25 |
| Comp. ML-DSA-44 + ECDSA | 35,09 ± 0,77 | 40,50 ± 0,97 | 43,11 ± 0,98 | 63,07 ± 0,89 |
| Comp. ML-DSA-65 + ECDSA | 41,68 ± 1,10 | 50,85 ± 2,02 | 56,40 ± 1,40 | 84,52 ± 1,59 |
| Comp. ML-DSA-87 + ECDSA | 44,84 ± 1,02 | 55,00 ± 1,14 | 61,38 ± 1,27 | 90,83 ± 1,48 |
| ECDSA + ML-DSA-44 | | | | 84,01 ± 0,84 |
| ECDSA + ML-DSA-65 | | | | 86,08 ± 1,25 |
| ECDSA + ML-DSA-87 | | | | 88,01 ± 1,22 |
| ECDSA + SLH-DSA (128,f) | | | | 172,73 ± 0,78 |
| ECDSA + SLH-DSA (128,s) | | | | 1 455,18 ± 4,90 |
| ECDSA + SLH-DSA (256,f) | | | | 337,18 ± 0,89 |
| ECDSA + SLH-DSA (256,s) | | | | 2 124,89 ± 4,53 |

Verification times are in milliseconds (± standard deviation). ECDSA means ECDSA with SHA256 over the curve P-256. SLH-DSA internal hash function is SHAKE. "Comp." refers to composite hybrid.

the "plain" signature increase; for example, an ML-DSA-44 CAdES-B-LTA signature is 6× larger than an ECDSA CAdES-B-LTA, whereas an ML-DSA-44 "plain" signature is 37× larger than an ECDSA signature. This is because AdES formats are already relatively size-inefficient; therefore, introducing post-quantum signatures does not change the relative size as dramatically.

Similar results also hold for hybrid solutions. One interesting aspect is that our backward-compatible hybrid post-quantum AdES signatures are smaller than purely post-quantum AdES signatures at the B-LTA level. This is because the hybrid solution only introduces a post-quantum archival timestamp, and the other cryptographic material is classical. This may seem counterintuitive, but this solution is only secure as long as classical signatures are still considered secure at the time of signing. It is therefore a transitional solution, and a purely post-quantum solution will eventually be necessary.

# Chapter 7

# Post-Quantum Migration Planning


7.1 **Existing Migration Guidance** . . . . . 200
7.1.1 EU-Proposed Roadmap . . . . . 200
7.1.2 Post-Quantum Primitives . . . . . 200
7.1.3 Other Guidance . . . . . 201
7.2 **Component Evaluation** . . . . . 201
7.2.1 Evaluation Framework . . . . . 201
7.2.2 Evaluation Results . . . . . 203
7.3 **Migration Roadmap** . . . . . 206
7.3.1 Post-Quantum TLS Key Exchange (target date: end of 2027) . . . . . 206
7.3.2 Post-Quantum PKI and Timestamping (target date: end of 2029) . . . . . 207
7.3.3 Post-Quantum AdES (target date: end of 2031) . . . . . 208
7.3.4 Post-Quantum SAML (target date: mid-2030) . . . . . 209
7.3.5 Post-Quantum TLS Authentication (target date: mid-2030) . . . . . 209
7.4 **Compliance Aspects** . . . . . 210
7.4.1 Hybrid Solutions and Compliance . . . . . 210
7.5 **Eventual Migration to Purely Post-Quantum Cryptography** . . . . . 210
7.6 **Conclusion** . . . . . 211


Although we have presented post-quantum solutions for the public-key infrastructure (Chapter 3), the TLS protocol (Chapter 4), XML and SAML (Chapter 5), and AdES (Chapter 6), we have not yet specified exactly which solutions are recommended. Furthermore, these cryptosystems typically do not exist in isolation, so an important aspect of migration is having a plan for the order in which to implement and deploy the selected solutions.

To address this question, this chapter provides an example post-quantum migration plan for an organization such as LuxTrust, which relies on the aforementioned cryptographic systems. Of course, each organization has different cryptographic systems, priorities, regulatory requirements,

etc. Consequently, the presented migration plan is not meant to be universal. However, it may serve as a blueprint for others planning a migration to post-quantum cryptography.

We begin by summarizing existing guidance on migration planning from regulatory bodies such as the European Commission in Section 7.1. In Section 7.2, we first present our evaluation framework, which helps us with developing the migration plan, and then we apply it to each discussed cryptographic system. Our example post-quantum migration roadmap is presented in Section 7.3; we conclude the chapter by discussing supplementary topics, such as compliance aspects and the eventual migration to purely post-quantum cryptography.

## 7.1 Existing Migration Guidance

Before presenting our own example migration roadmap, it is useful to consider existing guidance and roadmaps published by various organizations. In this section, we summarize the migration roadmap proposed by the European Commission [Eur25a] and touch upon roadmaps published by other organizations, such as the NSA. When developing a migration plan, it is important to be aware of these roadmaps, as some organizations may need to adhere to these guidelines and timelines.

### 7.1.1 EU-Proposed Roadmap

The EU roadmap essentially sets three milestones:

1) *By the end of 2026*: Migration plans will be developed by stakeholders.

2) *By the end of 2030*: The migration will be completed for high-risk use cases.

3) *By the end of 2035*: The migration will be completed for medium-risk use cases and "as much as feasible" for low-risk use cases.

"High-risk" refers to use cases where "compromising confidentiality after 10 years or more would still cause significant damage. This implies in particular that attackers have a strong motivation to capture and store such encrypted information even today in order to break the encryption in the future." However, the document also states that priority should also be given to complex systems such as PKI.

"Low-risk" use cases are not described in the document, but we presume the term refers to components which use symmetric cryptography where the migration is typically fairly straightforward and the attacks using Grover's algorithm are not expected to be as practical as attacks using Shor's algorithm.

This leaves *medium-risk use cases* between these two extremes, e.g., components which use public-key encryption, but the encrypted data is only considered sensitive for a short amount of time such as minutes, days, or a couple of months.

### 7.1.2 Post-Quantum Primitives

The high-level guidance [Eur25a] does not explicitly define which post-quantum KEMs or signatures are allowed—it merely states that hybrids are recommended. Regarding specific post-quantum

primitives, the guidance refers to the recommendations published by the European Union Agency for Cybersecurity (ENISA) [ENI25]. ENISA makes the following recommendations:

- *KEMs*: ML-KEM [NIS24b] and FrodoKEM [Fro25]. Both KEMs are recommended to be used in a hybrid mode with a classical scheme. The recommended parameters are the NIST security level 3 and 5 parameters.
- *Signatures*: ML-DSA [NIS24a] and SLH-DSA [NIS24c]. ML-DSA is recommended to be used in a hybrid mode, whereas SLH-DSA is not encouraged to be hybridized. The recommended parameters are the NIST security level 3 and 5 parameters.

### 7.1.3 Other Guidance

In addition to the EU, countries such as the US [NSA22] and the UK [NCS25b] have also published their own migration roadmaps regarding the transition to post-quantum cryptography. These roadmaps differ in several details, but ultimately both set the same deadline—the year 2035—for the completion of the migration.

One important aspect where the guidance differs is in the recommendation of hybrid solutions. EU guidance recommends hybrids, whereas the NSA and the NCSC do not—they do not explicitly prohibit them, but assert that purely post-quantum solutions are sufficiently secure.

Companies such as Google and Cloudflare have also published their roadmaps [Goo26c; Clo26b]. Cloudflare has already deployed hybrid post-quantum key exchange in TLS and plans to deploy purely post-quantum authentication in TLS by mid-2027. Furthermore, it aims to be "fully post-quantum secure" by 2029. Google has set "a timeline for post-quantum cryptography migration to 2029," but it has not explicitly stated what that entails. Previously, it announced its plan to deploy purely post-quantum authentication in TLS by the end of 2027 [Goo26b].

Notably, the 2029 migration targets of Google and Cloudflare are more aggressive than those of most other organizations, which have generally set 2030 as the migration target for critical systems and 2035 for the rest. Cloudflare explicitly mentions that recent advances optimizing Shor's algorithm have led it to adopt this aggressive target; see Section 1.4.1.1 for an overview of these advances. Google has not explicitly stated this, but since one of the recent advances [Bab+26] was made by its research teams, we suspect the reasoning was similar.

## 7.2 Component Evaluation

The first step in developing a migration plan is to evaluate each cryptographic system that is to be migrated. Such an evaluation is necessary because some cryptographic systems are more vulnerable than others or depend on other systems. In this section, we present our evaluation framework and apply it to the cryptographic systems that are part of our example migration. Note that there exist other frameworks such as the one presented in [Ama+24].

### 7.2.1 Evaluation Framework

We evaluate each system based on the following metrics:

**Quantum Threat:** This indicates the impact of quantum adversaries on the security of the cryptosystem. We define four levels:

1) *None*: The cryptosystem is not vulnerable to quantum adversaries—there is no need for migration.
2) *Low*: The cryptosystem is vulnerable to quantum adversaries, but only to attacks using Grover's algorithm. Recall from Section 1.4.2 that Grover-based attacks are expected to be significantly harder to execute than Shor-based attacks (Section 1.4.1).
3) *High*: The cryptosystem is vulnerable to quantum adversaries using Shor's algorithm; however, any such attack requires an *active* quantum adversary, i.e., the adversary can only perform the attack when it has access to a quantum computer (as opposed to the "store now, decrypt later" attack).
4) *Urgent*: The cryptosystem is vulnerable to quantum adversaries using Shor's algorithm, and it is vulnerable to the retroactive "store now, decrypt later" attack (Section 1.4.3). This threat is critical as the adversary can begin the first collection phase immediately, without requiring a quantum computer.

**Standardization:** This indicates whether standards covering post-quantum solutions exist for a given cryptosystem. We differentiate between three tiers:

1) *No*: There is no standardization effort at the moment.
2) *In progress*: Draft standards exist; however, they have not been finalized at the time of writing.
3) *Yes*: An official standard defining the post-quantum solution exists. We also consider "partial" standards, i.e., those that do not cover all proposed solutions, or that cover only certain post-quantum primitives.

**Implementation:** This indicates whether an implementation of the post-quantum solution exists for a given cryptosystem. We differentiate between three tiers:

1) *No*: There is no implementation at the moment.
2) *PoC*: There is a proof-of-concept, or experimental implementation.
3) *Yes*: The post-quantum solution has been integrated into an established library.

**Dependencies:** This indicates the cryptosystem's dependencies.

**Coordination:** This indicates whether the migration of a given cryptosystem can be executed independently, or whether it must be coordinated with other parties. In a sense, this metric reflects the backward compatibility of the post-quantum solution. We differentiate between two possibilities:

1) *Required*: The migration must be coordinated with other parties.
2) *Not required*: The migration can be executed independently.

Table 7.1: Cryptosystem Evaluation

| Component | Quantum Threat | Standardization | Implementation | Dependencies | Coordination |
|---|---|---|---|---|---|
| TLS (key exchange) | Urgent | In progress | Yes | None | Not required |
| TLS (auth.) | High | In progress | PoC | PKI | Not required |
| SAML (encryption) | Urgent | No | PoC | PKI | Required |
| SAML (auth., purely PQ) | High | No | PoC | PKI | Required |
| SAML (auth., hybrid PQ) | High | No | PoC | PKI | Not required |
| AdES (purely PQ) | High | In progress | PoC | PKI | Required |
| AdES (hybrid PQ) | High | In progress | PoC | PKI | Not required |
| PKI and Timestamping | High | In progress | PoC | None | Not required |

### 7.2.2 Evaluation Results

We present the results of our evaluation in Table 7.1. Let us discuss the results in more detail, as some nuance is warranted.

#### 7.2.2.1 TLS Key Exchange (Section 4.3)

- **Quantum threat**: We assign "Urgent" because TLS traffic can be recorded and decrypted later. The specific level of urgency depends on how easily the traffic can be captured and for how long the encrypted data are considered sensitive.
- **Standardization**: There exist draft RFCs for hybrid post-quantum key exchange in TLS 1.3 (ECDHE + ML-KEM hybrid) [Kwi+25; SFG25] and for purely post-quantum key exchange in TLS 1.3 with ML-KEM [Con25].
- **Implementation**: Many prominent TLS libraries, such as OpenSSL (version 3.5+) [Ope25], or BouncyCastle (version 1.81+) [Leg25a], already support hybrid post-quantum key exchange with the ECDHE + ML-KEM hybrid. The same applies for popular web browsers such as Google Chrome and Firefox.
- **Dependencies**: Post-quantum TLS key exchange does not have any dependencies.
- **Coordination**: Post-quantum TLS key exchange is backward compatible: clients that support it will use it, while classical-only clients will use a classical key exchange.

#### 7.2.2.2 TLS Authentication (Section 4.2)

- **Quantum threat**: We assign "High" because exploiting classical TLS authentication requires an active quantum adversary.
- **Standardization**: There exist draft RFCs for hybrid post-quantum authentication in TLS 1.3 [She+25] and for purely post-quantum authentication in TLS 1.3 with ML-DSA [HSW25].
- **Implementation**: There exists an OpenSSL fork [Ope23c] that supports post-quantum authentication in TLS. BouncyCastle provides experimental support for ML-DSA client authentication in TLS since version 1.82.

- **Dependencies**: Since TLS authentication relies on certificates, it inherently depends on a post-quantum PKI.
- **Coordination**: Post-quantum TLS authentication is also backward compatible (analogous to key exchange).

#### 7.2.2.3 SAML (Assertion) Encryption (Sections 5.2.2 and 5.4)

- **Quantum threat**: We assign *Urgent* because encrypted SAML assertions are vulnerable to "store now, decrypt later" attacks. Similar to TLS key exchange, the level of urgency ultimately depends on the nature of the encrypted data.
- **Standardization**: To the best of our knowledge, there is no ongoing standardization process to support post-quantum encryption in SAML. Note that by standardization in this context we mean the standardization of post-quantum KEM identifiers for XML/SAML, the processes and structures for encrypted assertions do not need to be changed.
- **Implementation**: We have developed an experimental implementation based on our design [Oup24c].
- **Dependencies**: SAML assertions may be encrypted with a public key that is either authenticated using a certificate—in which case a post-quantum PKI is required—or pre-shared and implicitly authenticated, in which case no PKI is needed. However, pre-sharing keys is impractical for large scale deployments; therefore, we assign a dependency on a PKI.
- **Coordination**: Deploying post-quantum-encrypted SAML assertions requires coordination with the party processing them. This is because encrypting data in a backward-compatible manner is inherently impossible—the processing party must support post-quantum SAML encryption.

#### 7.2.2.4 SAML Authentication (Sections 5.2.1 and 5.4)

- **Quantum threat**: We assign *High* because attacking SAML authentication requires an active quantum adversary.
- **Standardization**: To the best of our knowledge, there is no ongoing standardization process to support post-quantum authentication in SAML. Similar to SAML encryption, it is only necessary to standardize the identifiers of post-quantum signatures, since the post-quantum solutions do not require any other changes to the standards.
- **Implementation**: We have developed an experimental implementation based on our design [Oup24c].
- **Dependencies**: Analogous to SAML encryption, in setups where public keys may be pre-shared and authenticated via out-of-band mechanisms, there is no PKI dependency; however, in general, SAML relies on certificates and therefore needs a post-quantum PKI.

- **Coordination**: Post-quantum SAML authentication does not require coordination with the processing party when hybrid post-quantum authentication is used, as it is backward compatible—post-quantum parties can verify both classical and post-quantum signatures, while classical-only verifiers simply ignore the post-quantum signature. If one opts for a purely post-quantum solution, e.g., because the confidence in post-quantum signature has been established by that point, then the verifier must upgrade to support post-quantum signatures.

#### 7.2.2.5 AdES

- **Quantum threat**: We assign *High* because AdES signatures are threatened by active quantum adversaries.
- **Standardization**: Although not by ETSI—the author of the AdES standard—standardization of post-quantum versions of AdES components, such as CMS [SRG25], is ongoing.
- **Implementation**: We have developed an experimental implementation based on our design [Oup25a].
- **Dependencies**: AdES inherently requires a post-quantum PKI to function.
- **Coordination**: For AdES, the coordination requirement depends on the scenario in question: 1) for hybrid post-quantum AdES, which contains an additional post-quantum archival timestamp, no coordination is required, assuming classical-only verifiers simply ignore the extra timestamp; 2) for purely post-quantum AdES, coordination is required, as the verifier must support post-quantum signatures.

#### 7.2.2.6 PKI and Timestamping

- **Quantum threat**: We assign *High* because compromising PKI or timestamps requires an active quantum adversary.
- **Standardization**: An official RFC defining ML-DSA certificates [Mas+25] exists, and drafts for hybrid post-quantum certificates are available [Oun+25a].
- **Implementation**: We have developed an experimental implementation of a post-quantum PKI and timestamping service based on our design [Oup25a]. Note that our implementation depends on BouncyCastle, which already provides support for some post-quantum PKI-related cryptosystems, such as ML-DSA certificates etc.
- **Dependencies**: PKI and timestamping services have no dependencies. Recall that a timestamping service is essentially a specialized PKI.
- **Coordination**: Setting up a PKI or a timestamping service generally does not require coordination with other parties.

## 7.3 Migration Roadmap

In this section, we outline a migration roadmap based on the evaluation results from Section 7.2, including a discussion of the migration priority of each affected cryptosystem. Our roadmap aims to be compliant with the EU roadmap presented in Section 7.1.

The high-level overview of the migration roadmap is shown in Figure 7.1. We now explain the rationale behind the roadmap and its specific details.

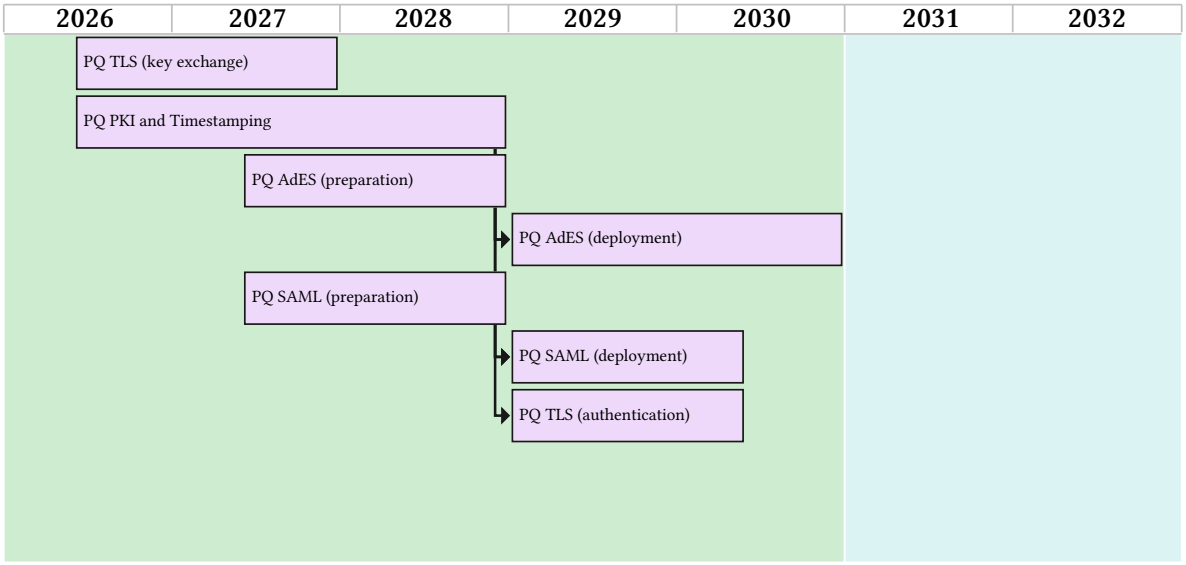



**Figure 7.1:** Migration roadmap. Green highlights the time period before the first EU migration deadline in 2031 (Section 7.1) while blue highlights the period before the second deadline in 2035.

We begin our migration timeline in mid-2026 with two components. The first component is post-quantum TLS key exchange (Section 7.3.1) and the second component is post-quantum PKI (Section 7.3.2). During this period, preparatory work on migration to post-quantum AdES and SAML (Sections 7.3.3 and 7.3.4) can begin. Once the post-quantum PKI is established, the roadmap concludes with finalizing the deployment of post-quantum AdES, SAML and post-quantum TLS authentication.

### 7.3.1 Post-Quantum TLS Key Exchange (target date: end of 2027)

We propose beginning the migration to post-quantum cryptography by upgrading TLS key exchange to post-quantum. This is because post-quantum key exchange for TLS is already mature—it is implemented in many libraries and is already enabled by default in many web browsers. Consequently, depending on the software stack, the migration may be as simple as updating the underlying TLS library, such as OpenSSL, and modifying a configuration file to enable post-quantum key exchange.

Upgrading to post-quantum key exchange does not require a post-quantum PKI. Additionally, it is backward compatible because TLS allows for specifying fallback algorithms; therefore, clients that do not support post-quantum key exchange are still supported. Note, however, that this fallback is optional and can be disabled.

We expect this migration to take no more than one and a half years, as the necessary implementations will be provided by third parties in the near future, if not already. However, there may be setbacks depending on the speed of third-party implementations—OpenSSL already supports it, but some TLS libraries may take longer to add support.

**Conclusion**

We recommend using a hybrid post-quantum key exchange in TLS 1.3, composed of ECDHE and ML-KEM—either the `X25519MLKEM768` hybrid or the more secure `SecP384r1MLKEM1024` [Kwi+25], depending on the use case.
In the future, additional hybrid combinations with HQC and FrodoKEM may become available. Recall from Chapter 4 that post-quantum key exchange in TLS is only considered for TLS 1.3; older TLS versions, such as 1.2, must be upgraded to 1.3.

### 7.3.2 Post-Quantum PKI and Timestamping (target date: end of 2029)

We recommend setting up a post-quantum PKI as soon as possible, as it is a critical dependency for other components, such as SAML and AdES. Furthermore, recall from Chapter 3 that PKI is a complex system that will take a considerable time to deploy.

In practice, PKIs rely on specialized hardware such as Hardware Security Modules (HSMs); consequently, the migration timeline depends on how quickly hardware manufactures can implement support for post-quantum signatures. Nonetheless, it is useful to begin experimenting with post-quantum PKI to get familiar with the new algorithms before post-quantum HSMs become readily available.

We expect the post-quantum PKI to operate alongside a classical PKI, forming the parallel hybrid post-quantum PKI discussed in Section 3.3.1. The classical PKI will eventually expire, though we do not expect this to occur before 2035—for example, roughly two thirds of root certificates distributed in macOS 15 expire after 2035. We also recommend that the first iteration of a post-quantum PKI have a lifetime of no more than 10 years, as the algorithms may be significantly weakened by unexpected cryptanalytic breakthroughs, or more efficient alternatives may emerge by that point.

**ML-DSA vs. FN-DSA.** FN-DSA (Falcon) and ML-DSA (Dilithium) are both lattice-based signatures. Recall from Section 2.4.3 that FN-DSA has smaller public keys and signatures than ML-DSA. Summing the sizes of a public key and a signature, FN-DSA offers size savings of around 60% (1563 bytes vs. 3732 bytes). However, FN-DSA is significantly slower at key generation (256×) and signing (3×), making it more suitable for use cases where signing and key generation speed is not crucial, such as a PKI.

The biggest obstacle to recommending FN-DSA is the lack of published standards—at the time of writing, NIST has not yet published a draft standard for FN-DSA, and we expect the final standard to be published no earlier than late 2026. The lack of finalized standards means that its integration into HSMs will likely come even later.

**Highest-Security Option.** ML-DSA is a well-rounded lattice-based signature in terms of performance and is also the most widely used at the time of writing. Recall from Section 2.4.3 that

SLH-DSA, a hash-based signature algorithm, has worse performance than ML-DSA, but it is considered a security-conservative choice because its security only relies on the security of the underlying hash function—unlike ML-DSA, whose security depends on both lattices *and* hash functions. We consider SLH-DSA to be a good fit for a PKI that prioritizes security over performance, e.g., a PKI used for archiving timestamps in AdES (Section 7.3.3). Regarding a specific SLH-DSA variant, we recommend using the (SHAKE, 256) variant, which offers the highest security. SLH-DSA also comes in small (s) and fast (f) variants; we leave the choice to the implementor, as both variants offer equivalent security.

**Timestamping Service.** Recall from Chapter 3 that a timestamping service is essentially an instance of a PKI. For certain components, such as AdES, it is crucial to have both a PKI and a timestamping service active at the same time. Consequently, we recommend setting up a timestamping authority alongside the post-quantum PKI.

**Conclusion**

We propose setting up a (purely) post-quantum PKI with ML-DSA-87 as the root signature algorithm, which offers the highest security for ML-DSA (NIST security level 5). For intermediate CAs, we recommend either ML-DSA-87, prioritizing security, or ML-DSA-65, prioritizing performance. For leaf certificates, we recommend using ML-DSA-44 for certificates with lifetimes of less than one year, and ML-DSA-65 for the rest.

### 7.3.3 Post-Quantum AdES (target date: end of 2031)

We recommend beginning to experiment with post-quantum AdES well before the deployment of the post-quantum PKI. The exact timeline depends on the development speed of relevant dependencies by third parties. In our proof-of-concept implementation [Oup25a], we utilize the DSS library [Eur25c], which itself depends on other libraries such as Apache Santuario [Apa25] for XML and BouncyCastle [Leg25b] for cryptographic primitives. While BouncyCastle is frequently updated and supports even experimental post-quantum implementations, Apache Santuario and DSS do not contain any post-quantum support at the time of writing.

Once a post-quantum PKI becomes available, post-quantum AdES may be readily deployed. Note that, similarly to PKI, AdES signatures typically need to be produced by certified HSMs; therefore, the eventual deployment also depends on the availability of these devices.

**Conclusion**

We recommend implementing post-quantum archival timestamps to provide hybrid post-quantum security with the ML-DSA-87 signature algorithm, as described in Section 6.3.2.2. Alternatively, SLH-DSA (SHAKE, 256) may be used as a more security-conservative choice. Note that, although the post-quantum solution is theoretically backward compatible, it ultimately depends on the verifiers' implementations and policies, as discussed in Section 6.3.2. Eventually, purely post-quantum AdES signatures are expected to be issued without reliance on classical signatures. For those, we recommend the same algorithms depending on the priority.

### 7.3.4 Post-Quantum SAML (target date: mid-2030)

The proposed post-quantum SAML migration timeline is similar to that of post-quantum AdES. The complete solution relies on a PKI; therefore, the final deployment of the post-quantum solution cannot occur before a post-quantum PKI has been set up. However, analogous to AdES, experimentation with post-quantum SAML may begin before that. We have provided our proof-of-concept implementation in [Oup24c].

We recommend deploying the hybrid post-quantum SAML signatures described in Section 5.4.2. The hybrid approach is backward compatible and compliant with existing regulations that may not be updated in time. On the other hand, if confidence in post-quantum algorithms is high and backward compatibility and regulatory compliance are not an issue, deploying purely post-quantum authentication straight away is reasonable.

Regarding post-quantum encryption in SAML, we specifically refer to the optional encryption of SAML assertions. In scenarios where SAML assertions contain sensitive data expected to remain sensitive over a span of 10–20 years, deploying hybrid post-quantum encryption as described in Section 5.4.2 is reasonable. Note, however, that hybrid post-quantum SAML encryption is inherently not backward compatible. In this case, backward compatibility must be negotiated out-of-band in advance, e.g., by configuring which relying parties can process hybrid post-quantum-encrypted assertions and which cannot (and will therefore not be protected against "store now, decrypt later" attacks).

If the encrypted SAML assertions do not contain long-term sensitive data, e.g., data intended to remain secret for only a few months, we recommend waiting and deploying purely post-quantum encryption once it becomes viable, i.e., once relying parties are able to process it.

**Conclusion**

We recommend ML-DSA-44 or ML-DSA-65 as the post-quantum signature algorithm, combined with the existing classical signature if hybrid authentication is deployed. For encryption, we recommend using ML-KEM-768 or ML-KEM-1024, combined with the existing classical public-key encryption algorithm if hybrid encryption is required.

### 7.3.5 Post-Quantum TLS Authentication (target date: mid-2030)

Deploying post-quantum authentication in TLS will be straightforward, similar to post-quantum key exchange in TLS (Section 7.3.1). Once a post-quantum web PKI operational, TLS servers may begin serving post-quantum certificates. Note that, as with key exchange, TLS contains a negotiation mechanism for signatures; therefore, post-quantum authentication can be deployed while maintaining backward compatibility.

As mentioned in Section 7.2.2, standardization efforts are currently underway for both purely post-quantum and hybrid post-quantum authentication in TLS, the latter using dual certificates. Similar to our discussion of hybrid authentication in SAML (Section 7.3.4), we recommend deploying hybrid post-quantum authentication in TLS—using dual certificates relying on parallel hybrid post-quantum PKI—for highly sensitive use cases. If confidence in the security of post-quantum signatures is not undermined in the near future, hybrid authentication in TLS may prove unnecessary, and purely post-quantum authentication may be deployed instead.

**Conclusion**

We recommend using ML-DSA-65 or ML-DSA-87 for post-quantum authentication in TLS, possibly in parallel with an existing classical signature.

## 7.4 Compliance Aspects

In this section, we briefly discuss the impact of regulations on migration to post-quantum cryptography.

Some components, such as the PKI, must adhere to certain regulations, e.g., that CA keypairs must be generated using a Hardware Security Module (HSM). Other regulations specify the cryptographic algorithms and parameters that may be used.

### 7.4.1 Hybrid Solutions and Compliance

One issue that may arise during migration is that the current regulations do not consider post-quantum algorithms to be compliant, as they have been conceived before their existence. Some regulations may get updated in time, but others are expected not to be, as regulatory processes tend to be slow.

Hybrid post-quantum solutions may alleviate these issues—a hybrid post-quantum solution is composed of a compliant classical algorithm and a non-compliant post-quantum algorithm. Since the post-quantum algorithm provides extra protection on top of the compliant component, the hybrid post-quantum solution can be considered compliant as well. This allows early adopters to offer post-quantum security while remaining compliant.

Note that the use of hybrid post-quantum solutions is also motivated by a lack of trust in the novel post-quantum algorithms—many in the cryptographic community feel that they have not been "battle-tested" enough. Hybrid post-quantum solutions therefore hedge against a potential break of post-quantum algorithms by *classical* computers.[1]

## 7.5 Eventual Migration to Purely Post-Quantum Cryptography

In our migration plan (Section 7.3), we recommend hybrid post-quantum solutions for some components. Although the motivation behind using hybrid post-quantum algorithms is to hedge against unexpected classical attacks against post-quantum algorithms, in the next decade or so, post-quantum algorithms will become sufficiently battle-tested to be used on their own as purely post-quantum solutions. Because of this eventual development, we recommend hybrid solutions that allow for an easy migration from hybrid post-quantum security to purely post-quantum security. Specifically:

- **TLS**: It is straightforward to change the preferred key exchange/signature algorithm from hybrid post-quantum to purely post-quantum.

[1] It does not make sense to consider a break of a "post-quantum" algorithm by a quantum computer, since we know for certain that quantum computers can break classical algorithms. In that scenario, even hybrids do not provide any security.

- **PKI and timestamping**: We recommend using the parallel hybrid post-quantum PKI, which is effectively two separate PKIs. Consequently, participants using it in parallel hybrid mode can simply be instructed to ignore the classical PKI—it will eventually expire, as discussed in Section 3.3.1.4.
- **AdES**: For AdES, we propose to use the (purely) post-quantum archival timestamp as a method to achieve hybrid post-quantum security. This method is essentially purely post-quantum in nature.
- **SAML**: For SAML, we have also described purely post-quantum solutions in Section 5.4.1. The migration is also expected to be fairly straightforward.

## 7.6 Conclusion

A successful migration to post-quantum cryptography requires more than just replacing algorithms; it demands a strategic, prioritized roadmap. In this chapter, we proposed a lightweight evaluation methodology based on several metrics. Using this methodology, we proposed an example roadmap for an organization that uses the cryptosystems discussed in previous chapters. We discussed in detail the reasoning behind these decisions. For instance, due to the immediate threat of "store now, decrypt later" attacks and the relative ease of deployment, a hybrid post-quantum TLS key exchange must be the immediate priority. This should be followed by the establishment of a post-quantum PKI and Timestamping Authority, which serve as foundational dependencies for other cryptosystems. Only once the PKI is established can other migrations, such as those for SAML and AdES, be finalized. Finally, we discussed external factors that may affect migration timelines, such as regulatory compliance and the availability of specialized hardware.

# Chapter 8

# Post-Quantum Password-Authenticated Public-Key Encryption


**8.1 Decryption Robustness . . . 215**
8.1.1 Strong Robustness Implies Decryption Robustness . . . 215
8.1.2 Decryption-Robust PKE from a Prefix-Hashing FO KEM . . . 219
8.1.3 Hashed-Public-Key Transform (HPK) . . . 223
8.1.4 Decryption Robustness of DHIES* . . . 228
**8.2 Password-Authenticated Public-Key Encryption . . . 229**
8.2.1 Universal Composability . . . 231
8.2.2 PAPKE Security . . . 234
8.2.3 Idealized Models . . . 236
8.2.4 PAPKE Constructions . . . 240
**8.3 PAPKE-HIC: Post-Quantum PAPKE . . . 244**
8.3.1 Proof of Theorem 9 . . . 245
**8.4 PAPKE-HIC Performance . . . 261**
8.4.1 PAPKE-Eligible KEMs . . . 262
8.4.2 PAPKE Implementations . . . 263
8.4.3 PAPKE Performance Results . . . 264
**8.5 Conclusion . . . 267**


Passwords are a common method of user authentication. We use them instead of cryptographic keys because of convenience: it is easier to remember a password, unlike a 256-bit key. By definition, "memorability" is negatively correlated with entropy: the higher the entropy, the harder it is to remember. This makes passwords less secure since they are easier to guess.

In the cryptographic context, the term "key" implies a value with "high entropy," meaning it is infeasible to iterate over all possible values. On the other hand, a *password* is assumed to have

"low entropy," meaning the space of possible values is small enough that it is feasible to iterate over them.

In this chapter, the topic is Password-Authenticated Public-Key Encryption (PAPKE), which is a novel cryptographic primitive introduced in [Bra+19b]. It is a public-key encryption scheme augmented with password-based authentication. Specifically, each keypair is tied to a password and the encrypting party must know the corresponding password to be able to encrypt.

A PAPKE can be used as an alternative to public-key fingerprint verification in end-to-end encrypted messaging apps, such as Signal [Sig26] or WhatsApp [Wha26]. In these apps, each communicating party has its own long-term public key that is distributed by a central server controlled by the provider. This allows a potentially malicious central server to mount a man-in-the-middle attack by distributing its own keys. To achieve proper end-to-end security, users must verify these distributed long-term public keys using other channels. For example, Signal provides an interface where users can compare, for instance, in person, their public keys by comparing their fingerprints.[1] Instead of comparing fingerprints, which are tedious to verify, the parties could agree on a password (even before starting the communication) and generate their PAPKE keypairs. These PAPKE public keys can be distributed by the central server as usual, but in this case a malicious central server cannot mount a man-in-the-middle attack since it cannot generate valid keys unless it knows the password.

A related construction to PAPKE is the Password-Authenticated Key Exchange (PAKE). A PAKE is a key exchange protocol between two parties, i.e., the goal of the protocol is for the two parties to agree on a shared secret. The main difference between a PAKE and other authenticated key exchange protocols, such as TLS (Chapter 4), is the method of authentication. TLS uses signature schemes, which require a PKI, whereas a PAKE uses pre-shared passwords. There are numerous classical and post-quantum PAKE protocols [HvO22; Aln+25] varying in their performance, security properties, etc. As illustrated in [Bra+19b], there is a straightforward way to build a PAKE from a PAPKE. Notably, a PAKE built from a PAPKE has a "re-use" property: the requestor's first message can be re-used across multiple sessions.

The authors of [Bra+19b] proposed two PAPKE constructions: PAPKE-IC and PAPKE-FO. Both of these constructions have limitations that make them unsuitable for realizing a quantum-safe PAPKE. Specifically, PAPKE-IC is inefficient and PAPKE-FO is a Diffie-Hellman-based construction which makes it automatically not post-quantum-compatible. In this chapter, we focus on developing a new post-quantum-compatible PAPKE construction called PAPKE-HIC.

We begin this chapter by recalling public-key encryption (PKE) security properties, presented in Section 2.3, and introducing a novel PKE property, which we call "Decryption Robustness" in Section 8.1. These PKE properties are essential for the security analysis of PAPKE constructions. We introduce the PAPKE primitive in Section 8.2 and discuss the post-quantum viability of existing PAPKE constructions. In Section 8.3, we present our novel post-quantum-compatible PAPKE construction, PAPKE-HIC, and prove that it is secure. We conclude this chapter by comparing the performance of post-quantum PAPKE-IC and PAPKE-HIC constructions in Section 8.4.

[1] A public-key fingerprint is a hash of the public key. It is typically represented as a QR code [Sig16].

**Remark**

The content of this chapter is based on our published work titled "HIC Is All You Need: Practical Post-Quantum Password-Authenticated Public Key Encryption" [Arr+26].

## 8.1 Decryption Robustness

This section studies a new public-key encryption (PKE) property that we refer to as *decryption robustness* (DROB-CCA). Informally, decryption robustness means that it is hard to create a valid ciphertext without knowing the public key. At first glance, it may seem like a counterintuitive property because *public* keys are typically assumed to be *public*. However, in the PAPKE (and PAKE) context, the public key is usually hidden from non-participants. Our motivation behind defining decryption robustness is presented in Section 8.3—we use it to prove the security of PAPKE-HIC and to correct the PAPKE-IC proof from [Bra+19b].

Decryption robustness is similar to the PKE properties of *strong robustness* (SROB-CCA) and *anonymity* (ANO-CCA), formally defined in Figure 2.4. Recall that SROB-CCA means that it is hard to create a ciphertext that decrypts to $\not\perp$ under two keypairs, whereas ANO-CCA means that a ciphertext does not leak the public key that was used to produce it.

In Section 8.1.1, we formally define decryption robustness and show that it reduces to strong robustness; however, the reduction is not tight. This leads us to study decryption robustness more directly. Concretely, we show in Section 8.1.2 that PKE schemes built using the KEM+DEM construction (Definition 12) satisfy decryption robustness with a tighter bound if the KEM is built using a specific variant of the Fujisaki-Okamoto (FO) transform and the DEM is INT-CTXT-secure (Figure 2.1). For KEMs that do not satisfy that requirement, in Section 8.1.3 we present a simple transform called the *hashed-public-key* (HPK) transform, that allows those KEMs to be used to build DROB-CCA-secure PKEs as well. Finally, in Section 8.1.4 we show that the classical DHIES$^*$ [ABR99; ABN10] PKE also satisfies DROB-CCA.

We work with two variants of decryption robustness: *decryption robustness* (DROB) and *decryption robustness under chosen-ciphertext attack* (DROB-CCA). See Figure 8.1 for formal definitions of DROB and DROB-CCA. DROB is a weaker variant of DROB-CCA where the adversary does not have access to a decryption oracle, i.e., it receives only the security parameter and must produce a valid ciphertext. In the DROB-CCA experiment, the adversary has access to the decryption oracle. However, the decryption oracle does not give the adversary any additional information since the adversary must query a valid ciphertext, which is the winning condition, to get any meaningful information using the oracle.

### 8.1.1 Strong Robustness Implies Decryption Robustness

It turns out that DROB and DROB-CCA are (polynomially) equivalent properties. The implication DROB-CCA $\implies$ DROB is trivial: if there exists an efficient DROB adversary $\mathcal{A}$, then this adversary is also an efficient DROB-CCA adversary as its winning ciphertext also wins the DROB-CCA game without even using the decryption oracle. The implication DROB $\implies$ DROB-CCA is proven in Theorem 2.

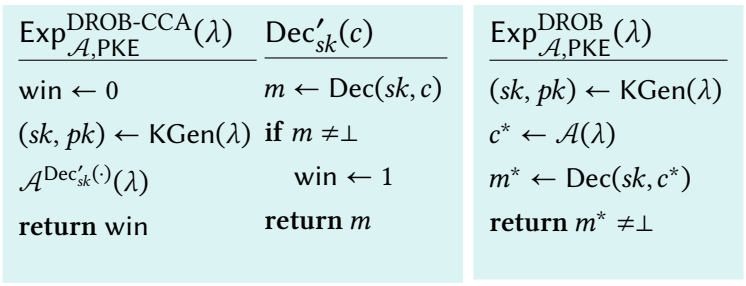



**Figure 8.1:** DROB-CCA and DROB security experiments for a public-key encryption scheme PKE = (KGen, Enc, Dec).

**Theorem 2.** *If a public-key encryption scheme* PKE *satisfies* DROB*, then it also satisfies* DROB-CCA*. Specifically, for any PPT adversary $\mathcal{A}$ against* DROB-CCA*, there exists a PPT adversary $\mathcal{B}$ against* DROB *such that*

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq q_D \cdot \mathsf{Adv}^{\text{DROB}}_{\mathcal{B},\mathsf{PKE}}(\lambda),$$

*where $q_D$ is the number of $\mathcal{A}$'s* Dec *oracle queries.*

*Proof.* We show that we can build a DROB adversary $\mathcal{B}$ given a DROB-CCA adversary $\mathcal{A}$. $\mathcal{B}$ randomly picks an index $i \in \{1, \ldots, q_D\}$, where $q_D$ is the number of Dec oracle queries made by $\mathcal{A}$, and runs $\mathcal{A}$ while answering $\mathcal{A}$'s first $i-1$ queries with $\perp$. When $\mathcal{A}$ makes the $i$-th query $c_i$, $\mathcal{B}$ outputs $c_i$ as its output.

By the definition of DROB-CCA, any winning $\mathcal{A}$ must make at least one Dec oracle query where the submitted $c$ decrypts to a valid message ($\mathsf{Dec}(sk, c) \neq \perp$). Therefore, there exists an index $i_F \in \{1, \ldots, q_D\}$ that is the index of the first query made by $\mathcal{A}$ that decrypts to a valid message. If $i = i_F$, then the simulation is perfect from $\mathcal{A}$'s view, as it received $\perp$ for all queries prior to $i_F$.

Therefore, if $\mathcal{B}$ correctly guessed $i_F$ and $\mathcal{A}$ wins, then $\mathcal{B}$ wins. In other words,

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq q_D \cdot \mathsf{Adv}^{\text{DROB}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

□

**Remark**

Since both notions are (polynomially) equivalent, we simply use the term "decryption robustness" to refer to both properties unless stated otherwise.

Decryption robustness being intuitively similar to strong robustness (SROB-CCA) is not a coincidence—strong robustness implies decryption robustness as we show in Theorem 3. Note that Theorem 3 works with the DROB property; by combining it with Theorem 2, we get the analogous result, formulated as Corollary 1, for DROB-CCA.

**Theorem 3.** *If a public-key encryption scheme* PKE *satisfies* SROB-CCA*, then it also satisfies* DROB*. In other words, for any PPT adversary $\mathcal{A}$ against* DROB*, there exists a PPT adversary $\mathcal{B}$ against* SROB-CCA *such that*

$$\mathsf{Adv}^{\text{DROB}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \sqrt{\mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda)}.$$

*Proof.* We prove the contrapositive: if there is an efficient adversary $\mathcal{A}$ against DROB, then there is an efficient adversary $\mathcal{B}$ against SROB-CCA.

The SROB-CCA adversary $\mathcal{B}$ receives the security parameter $\lambda$, two public keys $pk_0, pk_1$ corresponding to the private keys $sk_0, sk_1$, and access to two decryption oracles $\mathrm{Dec}(sk_0, \cdot), \mathrm{Dec}(sk_1, \cdot)$. $\mathcal{B}$ simply runs $\mathcal{A}$ with $\lambda$ as input and once $\mathcal{A}$ outputs $c^*$, $\mathcal{B}$ submits it to the challenger. Let us analyze the advantage of $\mathcal{B}$.

We need to show that $f(\lambda) \coloneqq \Pr\left[\mathsf{Exp}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) = 1\right]$ is *non-negligible*; specifically that

$$\exists d \in \mathbb{N}, \forall n_0 \in \mathbb{N}, \exists \lambda \geq n_0 \ : \ f(\lambda) > \frac{1}{\lambda^d}.$$

By assumption, $g(\lambda) \coloneqq \Pr\left[\mathsf{Exp}^{\text{DROB}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = 1\right]$ is non-negligible, i.e., there exists $d_{\text{DROB}} \in \mathbb{N}$ s.t. for any $n_0 \in \mathbb{N}$ there exists $\lambda \geq n_0$ for which $g(\lambda) > \frac{1}{\lambda^{d_{\text{DROB}}}}$. Note that

$$\begin{aligned} g(\lambda) &= \Pr_{\substack{(pk,sk)\sim\mathsf{KGen}(\lambda) \\ r\sim U}}[\mathrm{Dec}(sk, \mathcal{A}(\lambda, r)) \neq \perp] \\ &= \sum_{c\in\mathcal{C}} \left( \Pr_{r\sim U}[\mathcal{A}(\lambda, r) = c] \cdot \Pr_{(pk,sk)\sim\mathsf{KGen}(\lambda)}[\mathrm{Dec}(sk, c) \neq \perp] \right) \\ &> \frac{1}{\lambda^{d_{\text{DROB}}}}, \end{aligned}$$

where $\mathcal{C}$ is the set of ciphertexts and $r$ denotes the random coins for $\mathcal{A}$ from the uniform distribution over $\{0,1\}^{\mathrm{poly}(\lambda)}$ (denoted as $U$). By explicit definition of $\mathcal{B}$ in terms of $\mathcal{A}$, we have

$$\begin{aligned} f(\lambda) &= \Pr\left[\mathsf{Exp}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) = 1\right] \\ &= \Pr_{\substack{(pk_0,sk_0)\sim\mathsf{KGen}(\lambda) \\ (pk_1,sk_1)\sim\mathsf{KGen}(\lambda) \\ r\sim U}}[\mathrm{Dec}(sk_0, \mathcal{A}(\lambda, r)) \neq \perp \wedge \mathrm{Dec}(sk_1, \mathcal{A}(\lambda, r)) \neq \perp] \\ &= \sum_{c\in\mathcal{C}} \left( \Pr_{r\sim U}[\mathcal{A}(\lambda, r) = c] \cdot \Pr_{\substack{(pk_0,sk_0)\sim\mathsf{KGen}(\lambda) \\ (pk_1,sk_1)\sim\mathsf{KGen}(\lambda)}}[\mathrm{Dec}(sk_0, c) \neq \perp \ \wedge \mathrm{Dec}(sk_1, c) \neq \perp] \right). \end{aligned}$$

For a fixed $c$, the probabilities that $\mathrm{Dec}(sk_0, c) \neq \perp$ and $\mathrm{Dec}(sk_1, c) \neq \perp$ are independent since $sk_0$ and $sk_1$ are independently sampled; therefore

$$f(\lambda) = \sum_{c\in\mathcal{C}} \left( \Pr_{r\sim U}[\mathcal{A}(\lambda, r) = c] \cdot \left( \Pr_{(pk,sk)\sim\mathsf{KGen}(\lambda)}[\mathrm{Dec}(sk, c) \neq \perp] \right)^2 \right). \tag{8.1}$$

Jensen's inequality gives us that $\psi(\mathbb{E}[X]) \leq \mathbb{E}(\psi(X))$, where $\psi$ is a real convex function and $X : \Omega \to \mathcal{X}$ is a random variable. Furthermore, for any $h : \mathcal{X} \to \mathbb{R}$, $h(X)$ is a random variable and $\mathbb{E}[h(X)] = \sum_{x\in\mathcal{X}} h(x) \Pr[X = x]$. In our case $\psi(x) \coloneqq x^2$, $X \coloneqq \mathcal{A}(\lambda, r)$ where $r$ is sampled uniformly at random from the appropriate domain and

$$h(c) \coloneqq \Pr_{(sk,pk)\sim\mathsf{KGen}(\lambda)}[\mathrm{Dec}(sk, c) \neq \perp].$$

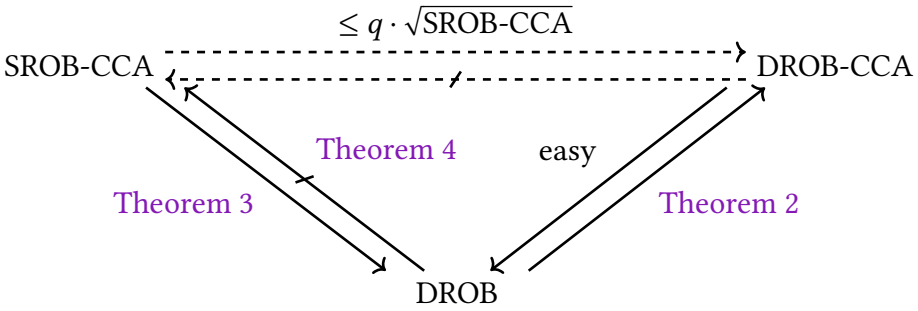



**Figure 8.2:** Relations among DROB, DROB-CCA, and SROB-CCA.

Therefore, by applying Jensen's inequality to Equation (8.1) we get

$$\begin{aligned} f(\lambda) \geq & \left( \sum_{c \in \mathcal{C}} \Pr_{r \sim U}[\mathcal{A}(\lambda, r) = c] \cdot \Pr_{(sk,pk) \sim \mathsf{KGen}(\lambda)}[\mathsf{Dec}(sk, c) \neq \perp] \right)^2 \\ = & g(\lambda)^2 > \frac{1}{\lambda^{2 \cdot d_{\mathrm{DROB}}}}. \end{aligned}$$

In other words, we have found $d := 2 \cdot d_{\mathrm{DROB}}$ which proves that $f(\lambda)$ is non-negligible. □

**Corollary 1.** *If a public-key encryption scheme* PKE *satisfies* SROB-CCA*, then it also satisfies* DROB-CCA*. More precisely, for any PPT adversary $\mathcal{A}$ against* DROB-CCA *there exists a PPT adversary $\mathcal{B}$ against* SROB-CCA *such that*

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq q_D \cdot \sqrt{\mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda)},$$

*where $q_D$ is the number of $\mathcal{A}$'s* Dec *oracle queries.*

One may wonder whether SROB-CCA and DROB-CCA are not equivalent notions as well. We prove that this is not the case in Theorem 4. We summarize the results of this section in Figure 8.2.

**Theorem 4.** *There exists a public-key encryption scheme that is* DROB*-secure, but is not* SROB-CCA*-secure. In other words,* DROB *does not imply* SROB-CCA.

*Proof.* We build a scheme PKE from a DROB-secure scheme PKE′ and show that PKE is DROB-secure, but not SROB-CCA-secure. Without loss of generality we assume that PKE′ is perfectly correct. Define PKE as follows:

- PKE.KGen($\lambda$): $(sk', pk') \leftarrow$ PKE′.KGen($\lambda$), **return** $((sk', pk'), pk')$;
- PKE.Enc($pk, m$): $c' \leftarrow$ PKE′.Enc($pk, m$), **return** $(c', pk)$;
- PKE.Dec($sk, c$): $(sk', pk') \leftarrow sk, (c', pk) \leftarrow c$,
  - **if** $pk = pk'$: **return** 1,
  - **else** : $m' \leftarrow$ PKE′.Dec($sk', c'$), **return** $m'$.

$\underline{\mathsf{pFO}[\mathsf{PKE}_1, \mathsf{ID}, \mathsf{F}].\mathsf{KGen}(\lambda)}$

$(pk, sk') \leftarrow \mathsf{PKE}_1.\mathsf{KGen}(\lambda)$
$s \leftarrow\!\$\ \mathcal{M}$
$sk \leftarrow (sk', s, pk)$
**return** $(pk, sk)$

$\underline{\mathsf{pFO}[\mathsf{PKE}_1, \mathsf{ID}, \mathsf{F}].\mathsf{Encaps}(pk)}$

$m \leftarrow\!\$\ \mathcal{M}$
$(K, r) \leftarrow \mathsf{F}(\mathsf{ID}(pk), m)$
$c \leftarrow \mathsf{PKE}_1.\mathsf{Enc}(pk, m, r)$
**return** $(K, c)$

$\underline{\mathsf{pFO}[\mathsf{PKE}_1, \mathsf{ID}, \mathsf{F}].\mathsf{Decaps}(sk, c)}$

$(sk', s, pk) \leftarrow sk$
$m' \leftarrow \mathsf{PKE}_1.\mathsf{Dec}(sk', c)$
$(K, r) \leftarrow \mathsf{F}(\mathsf{ID}(pk), m')$
$\bar{K} \leftarrow \mathsf{F}(s, c)[0 : k]$
**if** $m' = \perp \vee \mathsf{PKE}_1.\mathsf{Enc}(pk, m', r) \neq c$
  **return** $\bar{K}\boxed{/\perp}$
**return** $K$

**Figure 8.3:** $\mathsf{pFO}[\mathsf{PKE}_1, \mathsf{ID}, \mathsf{F}]$ is the Prefix-Hashing Fujisaki-Okamoto transform inspired by [Dum+21, Figure 4]. $\mathsf{PKE}_1$ is a PKE, ID is a fixed-length output function and F is a random oracle. The inclusion of the boxed code defines the explicitly rejecting transform.

PKE is DROB-secure because, compared to DROB-secure PKE′, the decryption is modified so that it allows an additional way to produce a valid ciphertext; however, this additional way requires the knowledge of the public key. Observe that DROB implies high min-entropy of public keys, i.e., it is infeasible for an adversary to guess the public key for PKE′.

PKE is not SROB-CCA-secure, because an adversary $\mathcal{A}$, given $(pk_0, pk_1)$, that produces the following ciphertext $c^*$ wins the SROB-CCA game: $\mathcal{A}$ encrypts a message $m$ using $pk_0$ with PKE′, gets $c' \leftarrow \mathsf{PKE}'.\mathsf{Enc}(pk_0, m)$ and then sets $c^* := (c', pk_1)$. $c^*$ is a PKE ciphertext, $m_0 \leftarrow \mathsf{PKE}.\mathsf{Dec}(sk_0, c^*)$ where $m_0 = m \neq \perp$ due to correctness of PKE′ and $m_1 \leftarrow \mathsf{PKE}.\mathsf{Dec}(sk_1, c^*)$ where $m_1 = 1 \neq \perp$ because $sk_1 = (sk_1', pk_1)$. □

### 8.1.2 Decryption-Robust PKE from a Prefix-Hashing FO KEM

In Section 8.1.1, we have shown that SROB-CCA implies DROB-CCA. However, the reduction is not tight; specifically, there is at least a quadratic security loss. This motivates a direct study of decryption robustness.

In Theorem 5, we prove that the standard hybrid KEM+DEM public-key encryption construction $\mathsf{PKE}^{hy}$ (Definition 12) directly satisfies DROB-CCA, provided that 1) the DEM is INT-CTXT-secure and 2) the KEM is constructed using the Prefix-Hashing Fujisaki-Okamoto transform pFO (Figure 8.3).

The pFO transform is a modified version of the transform introduced in [Dum+21, Figure 4]; the modification is that this transform does not derive the "rejection key" $\bar{K}$ from $\mathsf{ID}(pk)$, which exactly models KEMs such as ML-KEM [NIS24b]. Note that in the proof and the description of the transform (Figure 8.3) we use Python-like syntax to indicate truncated outputs, e.g., the first 128 bits of the value $\mathsf{F}(x)$ are denoted as $\mathsf{F}(x)[0 : 128]$.

We prove Theorem 5 in the Quantum Random Oracle Model (QROM) [Bon+11] using the following lemma from [SXY17, Lemma 2.2], which shows that a quantum random oracle can be used as a PRF (Definition 6).

**Lemma 2.** *Let $l \in \mathbb{N}$, and let $\mathsf{F} : \{0,1\}^l \times \mathcal{X} \to \mathcal{Y}$ and $\mathsf{H} : \mathcal{X} \to \mathcal{Y}$ be two independent random oracles. If an unbounded-time quantum adversary $\mathcal{D}$ makes at most $q$ queries to its quantum-accessible*

*random oracles, then we have*

$$\left|\Pr\left[1 \leftarrow \mathcal{D}^{\mathsf{F},\mathsf{F}(t,\cdot)}() | t \leftarrow\!\!\$ \{0,1\}^l\right] - \Pr\left[1 \leftarrow \mathcal{D}^{\mathsf{F},\mathsf{H}}()\right]\right| \leq \frac{2 \cdot q}{\sqrt{2^l}}.$$

**Theorem 5.** *Let* $\mathsf{PKE} = \mathsf{PKE}^{hy}[\mathsf{KEM}, \mathsf{DEM}]$ *be a hybrid public-key encryption scheme constructed from an implicitly rejecting key encapsulation mechanism* KEM *and a data encapsulation mechanism* DEM*. If* KEM *is constructed using the Prefix-Hashing FO transform* pFO *(Figure 8.3), i.e.,* $\mathsf{KEM} = \mathsf{pFO}[\mathsf{PKE}_1, \mathsf{ID}, \mathsf{F}]$ *where* $\mathsf{PKE}_1$ *is a public-key encryption scheme,* $\mathsf{ID} : \mathcal{PK} \to \{0,1\}^n$ *is a fixed-length output function,* $\mathsf{F} : \{0,1\}^* \to \{0,1\}^k \times \{0,1\}^{\bar{r}}$ *is a quantum random oracle and* DEM *is* INT-CTXT*-secure, then for any QPT adversary* $\mathcal{A}$ *against* DROB-CCA *of* PKE *there exists a QPT adversary* $\mathcal{B}$ *against* INT-CTXT *of* DEM *such that*

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{4 \cdot q_{\mathsf{F}}}{\sqrt{2^{\min(m,k)}}} + q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda),$$

*where* $m = H_\infty(\mathsf{ID}(pk))$ *is the min-entropy of* $\mathsf{ID}(pk)$ *for* $(pk, sk) \leftarrow \mathsf{PKE}_1.\mathsf{KGen}(\lambda)$*,* $q_{\mathsf{F}}$ *denotes the number of queries to the quantum random oracle* F *and* $q_D$ *denotes the number of queries to the (classical) oracle* Dec*.*

*Proof.* Let $\mathcal{A}$ be an adversary against DROB-CCA of PKE. Denote by $\Pr[\mathsf{Game}_i]$ the probability that $\mathcal{A}$ wins the $i$-th game. Figure 8.4 provides a detailed description of the games.

$\mathsf{Game}_0$. This is the unmodified DROB-CCA game with an unpacked PKE, i.e.,

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = \Pr[\mathsf{Game}_0].$$

$\mathsf{Game}_1$. In $\mathsf{Game}_1$, we replace the KEM key and randomness derivation in the decryption oracle $(K, r) \leftarrow \mathsf{F}(\mathsf{ID}(pk), m_1)$ with $(K, r) \leftarrow \mathsf{H}_1(m_1)$ where $\mathsf{H}_1$ is an internal random oracle. This change is undetectable by $\mathcal{A}$ with probability given by Lemma 2.

We build a distinguisher $\mathcal{D}$ from Lemma 2 as follows. $\mathcal{D}$ has access to two quantum random oracles $\mathsf{F}, \mathsf{F}'$ where $\mathsf{F}'(\cdot) = \mathsf{F}(t, \cdot)$ for $t \leftarrow\!\!\$ \{0,1\}^m$, or $\mathsf{F}'(\cdot) = \mathsf{H}_1(\cdot)$. $\mathcal{D}$ runs $\mathcal{A}$ in a modified $\mathsf{Game}_0$ where $(K, r) \leftarrow \mathsf{F}(\mathsf{ID}(pk), m_1)$ in the decryption oracle is replaced with $(K, r) \leftarrow \mathsf{F}'(m_1)$. $\mathcal{D}$ outputs 1 if $\mathcal{A}$ wins the game, i.e., submits a decrypting ciphertext. Clearly, if $\mathsf{F}'(\cdot) = \mathsf{F}(t, \cdot)$ then the game corresponds to $\mathsf{Game}_0$ since $t$ has the same min-entropy as $\mathsf{ID}(pk)$ and if $\mathsf{F}'(\cdot) = \mathsf{H}_1(\cdot)$, then it corresponds to $\mathsf{Game}_1$.

By Lemma 2, $|\Pr[\mathsf{Game}_0] - \Pr[\mathsf{Game}_1]| =$

$$\left|\Pr\left[1 \leftarrow \mathcal{D}^{\mathsf{F},\mathsf{F}(t,\cdot)}() | t \leftarrow\!\!\$ \{0,1\}^m\right] - \Pr\left[1 \leftarrow \mathcal{D}^{\mathsf{F},\mathsf{H}_1}()\right]\right| \leq \frac{2 \cdot q_{\mathsf{F}}}{\sqrt{2^m}}.$$

$\mathsf{Game}_2$. In $\mathsf{Game}_2$, we replace the rejection KEM key derivation in the decryption oracle $K' \leftarrow \mathsf{F}(s, c_1)$ with $K' \leftarrow \mathsf{H}_2(c_1)$ where $\mathsf{H}_2$ is an internal random oracle. Analogously to the previous game hop, by Lemma 2 we get

$$|\Pr[\mathsf{Game}_1] - \Pr[\mathsf{Game}_2]| \leq \frac{2 \cdot q_{\mathsf{F}}}{\sqrt{2^k}}.$$

$\mathsf{Game}_3$. In $\mathsf{Game}_3$, we reorganize the decryption oracle code. $K$ is sampled from the same distribution regardless of whether $\mathsf{H}_1$ or $\mathsf{H}_2$ is called. Therefore, we can "unroll" $\mathsf{H}_1$ and $\mathsf{H}_2$ and replace them with sampling of $K \leftarrow\!\!\$\ \{0,1\}^k$ with record keeping. We also remove the "rejection if-branch."

We keep a list $\mathsf{List}_C$ with pairs $(c_1, K)$ where $c_1$ is the queried KEM ciphertext. Observe that keeping the list indexed by $c_1$ is enough to preserve consistency. Consider that $\mathcal{A}$ queries $c = (c_1, c_2)$ and $c' = (c_1', c_2')$ to the Dec oracle:

- If $c_1 = c_1'$, then the lookup in $\mathsf{List}_C$ assures consistency.
- If $c_1 \neq c_1'$ and $\mathsf{PKE}_1.\mathsf{Dec}(sk, c_1) \neq \mathsf{PKE}_1.\mathsf{Dec}(sk, c_1')$, then $\mathsf{Game}_2$ also derives independent DEM keys.
- If $c_1 \neq c_1'$ and $\mathsf{PKE}_1.\mathsf{Dec}(sk, c_1) = m' = \mathsf{PKE}_1.\mathsf{Dec}(sk, c_1')$, then observe that the derived DEM keys are also independent because $c_1$ or $c_1'$ must fail the FO re-encryption check given that $\mathsf{PKE}_1.\mathsf{Enc}(pk, m', r)$ is deterministic.

Since this game is just a code reorganization, we have $\Pr[\mathsf{Game}_2] = \Pr[\mathsf{Game}_3]$.

Observe that $\mathcal{A}$ wins $\mathsf{Game}_3$ if and only if it queries $c = (c_1, c_2)$ where $\mathsf{DEM}.\mathsf{Dec}(K, c_2) \neq \perp$ where $K \leftarrow\!\!\$\ \{0,1\}^k$. We show that we can build an INT-CTXT adversary $\mathcal{B}$ using $\mathcal{A}$ in $\mathsf{Game}_3$.

Let $i \in \{1, \ldots, q_D\}$ be the index of the first query from $\mathcal{A}$ that decrypts. $\mathcal{B}$ guesses $i \in \{1, \ldots, q_D\}$ and runs $\mathcal{A}$ in a modified $\mathsf{Game}_3$ where $\mathcal{B}$ answers with $\perp$ to the first $i-1$ queries. During the $i$-th query, it uses its Dec oracle from the INT-CTXT challenger instead of calling $\mathsf{DEM}.\mathsf{Dec}(K, \cdot)$ with $K \leftarrow\!\!\$\ \{0,1\}^k$. Observe that $K$ is sampled from the same distribution in $\mathsf{Game}_3$ and in the INT-CTXT game, and $\mathcal{B}$ perfectly simulates $\mathsf{Game}_3$ if it guessed $i$. Therefore, $\mathcal{B}$ always wins if it correctly guessed $i$ and $\mathcal{A}$ wins. In other words,

$$\Pr[\mathsf{Game}_3] \leq q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda).$$

In total, we have

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{4 \cdot q_{\mathsf{F}}}{\sqrt{2^{\min(m,k)}}} + q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda).$$

□

**Explicitly Rejecting KEMs.** A variant of Theorem 5 also holds for a PKE constructed from an explicitly rejecting KEM. The proof is almost identical except that we do not need to consider the rejection key. As a consequence, there is no $s \in \{0,1\}^k$ and the bound is $\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{2 \cdot q_{\mathsf{F}}}{\sqrt{2^m}} + q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda)$.

**Games 0-3**

$\mathsf{Exp}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}^{hy}[\mathsf{pFO}[\mathsf{PKE}_1,\mathsf{ID},\mathsf{F}],\mathsf{DEM}]}(\lambda)$

$\mathsf{win} \leftarrow 0$
$(pk, sk') \leftarrow \mathsf{PKE}_1.\mathsf{KGen}(\lambda)$
$s \leftarrow\!\$\ \{0,1\}^k$
$sk \leftarrow\!\$\ (sk', s, pk)$
$\boxed{\mathsf{List}_C \leftarrow \emptyset}$ $\mathsf{G}_3$
$\mathcal{A}^{\mathsf{Dec}(sk,\cdot),\mathsf{F}}(\lambda)$
**return** win

$\mathsf{F}(x)/\mathsf{H}_1(x)$

**if** $\exists(x, K, r) \in \mathsf{List}_{\mathsf{F}/\mathsf{H}_1}$
  **return** $(K, r)$
$(K, r) \leftarrow\!\$\ \{0,1\}^k \times \{0,1\}^{\bar{r}}$
add $(x, K, r)$ to $\mathsf{List}_{\mathsf{F}/\mathsf{H}_1}$
**return** $(K, r)$

$\mathsf{H}_2(x)$

**if** $\exists(x, K) \in \mathsf{List}_{\mathsf{H}_2}$
  **return** $K$
$K \leftarrow\!\$\ \{0,1\}^k$
add $(x, K)$ to $\mathsf{List}_{\mathsf{H}_2}$
**return** $K$

$\mathsf{Dec}(sk, c)$

$(c_1, c_2) \leftarrow c$
$(sk', s, pk) \leftarrow sk$
$m' \leftarrow \mathsf{PKE}_1.\mathsf{Dec}(sk', c_1)$
$\boxed{(K, r) \leftarrow \mathsf{F}(\mathsf{ID}(pk), m')}$ $\mathsf{G}_0$
$\boxed{(K, r) \leftarrow \mathsf{H}_1(m')}$ $\mathsf{G}_{1,2}$
**if** $m' = \perp \vee \mathsf{PKE}_1.\mathsf{Enc}(pk, m', r) \neq c_1$
  $\boxed{K \leftarrow \mathsf{F}(s, c_1)[0 : k]}$ $\mathsf{G}_{0,1}$
  $\boxed{K \leftarrow \mathsf{H}_2(c_1)}$ $\mathsf{G}_2$
  $\boxed{\begin{array}{l}\textbf{if } \exists(c_1, K') \in \mathsf{List}_C \\ \quad K \leftarrow K' \\ \textbf{else} \\ \quad K \leftarrow\!\$\ \{0,1\}^k \\ \quad \text{add } (c_1, K) \text{ to } \mathsf{List}_C\end{array}}$ $\mathsf{G}_3$
$m \leftarrow \mathsf{DEM}.\mathsf{Dec}(K, c_2)$
**if** $m \neq \perp$
  win $\leftarrow 1$
**return** $m$

**Figure 8.4:** Sequence of games for the proof of Theorem 5.

#### 8.1.2.1 Application of Theorem 5 to Existing KEMs

Recall that compared to the original Prefix-Hashing FO transform from [Dum+21], the pFO transform (Figure 8.3) is modified so that it directly models post-quantum KEMs such as ML-KEM [NIS24b] and FrodoKEM [Fro25]. However, if we consider the original transform, which also derives the rejection key from $\mathsf{ID}(pk)$, the bound from Theorem 5 becomes tighter: we get $4 \cdot q_{\mathsf{F}} \cdot \sqrt{2^{-m}}$ instead of $4 \cdot q_{\mathsf{F}} \cdot \sqrt{2^{-\min(m,k)}}$.

As discussed in [Dum+21], the function ID may be, for example, a hash function or a function that selects the random portion of a public key. In ML-KEM, ID is instantiated as SHA3-256, meaning the entire public key is hashed, including a 256-bit random seed. In FrodoKEM, the key derivation is more involved, however, it still incorporates a hash of the entire public key.

#### 8.1.2.2 Impact of Theorem 5 on Concrete Security

We compare the concrete DROB-CCA security bounds implied by Theorem 3 and by Theorem 5 considering a hybrid PKE instantiated with ML-KEM [NIS24b].

From [GMP22, Theorem 2 + Theorem 12], we have $\mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{4 \cdot q_{\mathsf{F}}}{2^{128}} + \epsilon$, where $q_{\mathsf{F}}$ denotes the number of queries to the quantum-accessible random oracle and $\epsilon \ll 2^{-128}$ is a negligible term. By Corollary 1, we have $\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq q_D \cdot \sqrt{\mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda)}$, where $q_D$ is the number of decryption queries. Replacing the $\mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$ term with the bound above, we get $\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq q_D \cdot \sqrt{\frac{4 \cdot q_{\mathsf{F}}}{2^{128}} + \epsilon}$. While Corollary 1 shows that DROB-CCA and SROB-CCA are asymptotically equivalent, a concrete security analysis reveals that even for arguably modest upper bounds on $q_{\mathsf{F}}$ and $q_D$ (say $q_{\mathsf{F}} = 2^{40}$ and $q_D = 2^{16}$), the combination of [GMP22, Theorem 2 + Theorem 12] and Corollary 1 provides limited insight into the actual security guarantees that ML-KEM delivers in terms of decryption robustness, as it does not yield an upper bound that would be regarded as meaningful under modern security standards.

In contrast, the direct analysis of the Prefix-Hashing FO transform allows us to establish security bounds via Theorem 5 that remain meaningful in practical settings, i.e., we have that $\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{4 \cdot q_{\mathsf{F}}}{\sqrt{2^{\min(m,k)}}} + q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda)$, where $m = k = 256$ for ML-KEM. Since the DEM can be selected such that $\mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda)$ can be made negligible, this gives a meaningful upper bound on the decryption-robustness advantage against an ML-KEM-based PKE.

### 8.1.3 Hashed-Public-Key Transform (HPK)

A lot of post-quantum KEMs, such as ML-KEM and FrodoKEM, are built using the Prefix-Hashing FO transform presented in Figure 8.3. Therefore, they can be used to build a DROB-CCA-secure PKE (Theorem 5).

However, what about KEMs that are not constructed using the Prefix-Hashing FO transform? For example, Classic McEliece does not derive the symmetric key from the public key.[2] As noted in [GMP22], there exists a significant amount of "trivial" ciphertexts that decapsulate to the same value regardless of the private key which prevents Classic McEliece from achieving decryption

[2]Presumably, this is an intentional design since Classic McEliece public keys are large and hashing them would hinder performance.

| HPK[KEM, ID, H].KGen($\lambda$) | HPK[KEM, ID, H].Encaps($pk$) | HPK[KEM, ID, H].Decaps($sk, c$) |
|---|---|---|
| $(pk, sk) \leftarrow$ KEM.KGen($\lambda$) | $(c, k) \leftarrow$ KEM.Encaps($pk$) | $k \leftarrow$ KEM.Decaps($sk, c$) |
| **return** $(pk, sk)$ | $k' \leftarrow \mathsf{H}(\mathsf{ID}(pk), k)$ | **if** $k = \perp$ **then** **return** $\perp$ |
| | **return** $(c, k')$ | $k' \leftarrow \mathsf{H}(\mathsf{ID}(pk), k)$ |
| | | **return** $k'$ |

**Figure 8.5:** Description of the hashed-public-key KEM transform HPK[KEM, ID, H] where KEM is a KEM, $\mathsf{ID} : \mathcal{PK} \to \{0,1\}^n$ is a fixed-length output function and $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$ is a random oracle. The inclusion of the boxed code defines the transform for explicitly rejecting KEMs.

robustness. Consequently, Classic McEliece needs to be modified such that it uses the public key during the symmetric-key derivation.

For that purpose, we define the *Hashed-Public-Key* KEM transform HPK that can be applied to any KEM and the transformed KEM can then be used to build a DROB-CCA-secure PKE as proven in Theorem 6. The transform is described in detail in Figure 8.5—it additionally hashes the symmetric key together with $\mathsf{ID}(pk)$. We use the generalization $\mathsf{ID}(pk)$ from [Dum+21] because, for example, in the case of Classic McEliece, it is reasonable to define $\mathsf{ID}(pk)$ to be a precomputed hash of the public key since the public key is large.

**Theorem 6.** *Let* $\mathsf{PKE} = \mathsf{PKE}^{hy}[\mathsf{HPK}[\mathsf{KEM}, \mathsf{ID}, \mathsf{H}], \mathsf{DEM}]$ *be a hybrid public-key encryption scheme constructed from the KEM* HPK[KEM, ID, H]*, where* KEM *is a KEM and a* DEM *is a DEM. If* $\mathsf{ID} : \mathcal{PK} \to \{0,1\}^n$ *is a fixed-length output function,* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$ *is a quantum random oracle and* DEM *is* INT-CTXT*-secure, then for any QPT adversary* $\mathcal{A}$ *against* DROB-CCA *of* PKE *there exists a QPT adversary* $\mathcal{B}$ *against* INT-CTXT *of* DEM *such that*

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{2 \cdot q_{\mathsf{H}}}{\sqrt{2^m}} + q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda),$$

*where* $m = H_\infty(\mathsf{ID}(pk))$ *is the min-entropy of* $\mathsf{ID}(pk)$ *for* $(pk, sk) \leftarrow \mathsf{KEM.KGen}(\lambda)$*, and* $q_{\mathsf{H}}$ *and* $q_D$ *denote the number of oracle queries to* H *and* Dec*, respectively.*

*Proof.* We omit this proof as it is essentially the same as the proof of Theorem 5. □

#### 8.1.3.1 HPK Preserves KEM Properties

Ultimately, our goal is to build a PKE, from a KEM and a DEM, that is IND-CCA, ANO-CCA, SROB-CCA and DROB-CCA secure so that it can be used in PAPKE-IC (Section 8.2) and PAPKE-HIC (Section 8.3). To achieve these properties from the building blocks KEM and DEM, we utilize the following theorems:

1) Theorem 1: $\mathsf{PKE}^{hy}$ constructed from an IND-CCA-secure KEM and an IND-CCA-secure DEM is an IND-CCA-secure PKE.

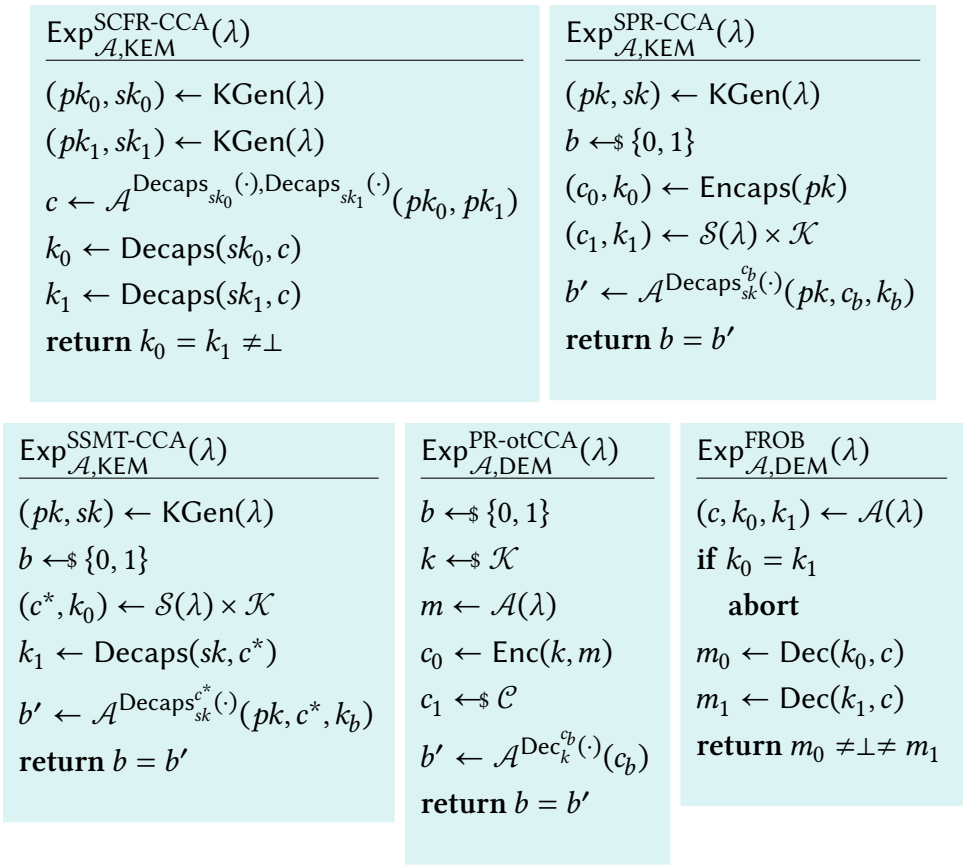



**Figure 8.6:** Security experiments defining KEM security properties SCFR-CCA, SPR-CCA and SSMT-CCA, and DEM security properties PR-otCCA and FROB.

2) [GMP22, Theorem 2]: $\mathsf{PKE}^{hy}$ constructed from a SCFR-CCA-secure KEM and a FROB-secure DEM is a SROB-CCA-secure PKE.

3) [Xag21, Theorem 3.2]: $\mathsf{PKE}^{hy}$ constructed from a SPR-CCA-secure and SSMT-CCA-secure KEM and a PR-otCCA-secure DEM is a SPR-CCA-secure PKE.

4) [Xag21, Theorem 2.5]: If a PKE/KEM is SPR-CCA-secure, then it is ANO-CCA-secure.

We have already defined some of the mentioned properties in Figures 2.1, 2.4 and 2.5, but we have not defined the following properties:

- SCFR-CCA [GMP22]: This KEM property, *strong collision-freeness under chosen-ciphertext attack*, captures that it is hard to produce a ciphertext that decapsulates to the same key under two generated keypairs.
- SPR-CCA [Xag21]: This KEM property, *strong pseudo-randomness under chosen-ciphertext attack*, captures that KEM ciphertexts are pseudorandom. Formally, this means that there exists an efficient simulator $\mathcal{S}$ that given only the security parameter can produce ciphertexts indistinguishable from real ones.

- SSMT-CCA [Xag21]: This KEM property, *strong smoothness under chosen-ciphertext attack*, captures that it is hard to distinguish a random key from a decapsulated key given a KEM ciphertext.
- PR-otCCA [Xag21]: This DEM property, *pseudo-randomness of ciphertexts under one-time chosen-ciphertext attack*, captures that DEM ciphertexts are pseudorandom.
- FROB [Xag21]: This DEM property, *full robustness*, captures that it is hard to produce a valid ciphertext that decrypts under two different keys—essentially strong robustness for DEMs.

See Figure 8.6 for formal definitions of the above-mentioned properties.
We need to show that the HPK transform preserves those properties. Specifically:

$$\begin{aligned}
\text{IND-CCA KEM} &\implies \text{IND-CCA HPK[KEM, ID, H]},\\
\text{SCFR-CCA KEM} &\implies \text{SCFR-CCA HPK[KEM, ID, H]},\\
\text{SSMT-CCA KEM} &\implies \text{SSMT-CCA HPK[KEM, ID, H]},\\
\text{SPR-CCA KEM} &\implies \text{SPR-CCA HPK[KEM, ID, H]}.
\end{aligned}$$

All of these are simple exercises: in every experiment we can simulate the "hashed version" of the experiment since we always have access to ID, H and the public key. Recall that strong pseudo-randomness (SPR-CCA) implies anonymity (ANO-CCA) ([Xag21, Theorem 2.5]).

Observe that HPK[KEM, ID, H] is a SCFR-CCA-secure KEM (regardless of whether KEM is) assuming $\mathsf{ID}(pk)$ has large min-entropy and that H is a random oracle—either there is a collision of honestly generated public keys, or there exists a collision for H. Similar result was shown in [Xag21, Theorem K.3]. This means that HPK-transformed Classic McEliece can be used in PAPKE-(H)IC. As noted in [GMP22; Xag21], the plain Classic McEliece is not collision-free, so we cannot directly apply 2).

Note that showing that the HPK transform preserves anonymity is not as trivial as for the other properties. This is because an adversary $\mathcal{A}$ against ANO-CCA of KEM (Figure 2.5) receives $(pk_0, pk_1, c^*, k^*)$ from its challenger, and it needs to act as a challenger for an adversary $\mathcal{B}$ against HPK[KEM, ID, H]. However, $\mathcal{A}$ does not know which public key to use to derive $k^\dagger = \mathsf{H}(\mathsf{ID}(pk), k^*)$ for $\mathcal{B}$.

In Theorem 7, we show that the HPK transform indeed preserves anonymity if the KEM is also OW-CCA-secure (Figure 2.5). Since OW-CCA is not a standard KEM notion, we first show that IND-CCA $\implies$ OW-CCA in Lemma 3.

**Lemma 3.** *If a key encapsulation mechanism* KEM *is* IND-CCA*-secure, then it is* OW-CCA*-secure. Specifically, for any PPT adversary $\mathcal{A}$ against* OW-CCA*, there exists a PPT adversary $\mathcal{B}$ against* IND-CCA *such that*

$$\mathsf{Adv}^{\text{OW-CCA}}_{\mathcal{A},\mathsf{KEM}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda) + \frac{1}{|\mathcal{K}|},$$

*where $\mathcal{K}$ is the symmetric key space of* KEM.

*Proof.* The IND-CCA adversary $\mathcal{B}$ works as follows. It receives $(pk, c^*, k)$ from its challenger, where $k$ is either the actual key encapsulated in $c^*$ (if $b = 0$), or a random key (if $b = 1$). $\mathcal{B}$ runs $\mathcal{A}$ with input $(pk, c^*)$ and responds to $\mathcal{A}$'s queries using its own Decaps oracle. Eventually, $\mathcal{A}$ outputs a key $k'$. $\mathcal{B}$ outputs $b' = 0$ if $k = k'$, otherwise it outputs $b' = 1$.

Let us analyze the success probability of $\mathcal{B}$.

$$2 \cdot \mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda) = \left|\Pr[0 \leftarrow \mathcal{B} \mid b = 0] - \Pr[0 \leftarrow \mathcal{B} \mid b = 1]\right|.$$

First, consider the game where $b = 0$. $\Pr[0 \leftarrow \mathcal{B} \mid b = 0]$ is the probability that $\mathcal{B}$ outputs 0 if $b = 0$. This occurs when $\mathcal{A}$ correctly guesses the key. Therefore,

$$\Pr[0 \leftarrow \mathcal{B} \mid b = 0] = \mathsf{Adv}^{\text{OW-CCA}}_{\mathcal{A},\mathsf{KEM}}(\lambda).$$

Now consider the game where $b = 1$, $\Pr[0 \leftarrow \mathcal{B} \mid b = 1]$ is the probability that $\mathcal{B}$ outputs 0 if $b = 1$. In this game, $k$ is a random key independent of $c^*$. Therefore, the probability that $\mathcal{A}$ correctly guesses the key is $\leq \frac{1}{|\mathcal{K}|}$. In other words,

$$\Pr[0 \leftarrow \mathcal{B} \mid b = 1] \leq \frac{1}{|\mathcal{K}|}.$$

In total, we have

$$2 \cdot \mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda) \geq \left|\mathsf{Adv}^{\text{OW-CCA}}_{\mathcal{A},\mathsf{KEM}}(\lambda) - \frac{1}{|\mathcal{K}|}\right|$$
$$\Longrightarrow$$
$$\mathsf{Adv}^{\text{OW-CCA}}_{\mathcal{A},\mathsf{KEM}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda) + \frac{1}{|\mathcal{K}|}.$$

□

**Theorem 7.** *If* KEM *is an anonymous* (ANO-CCA) *and one-way secure* (OW-CCA) *KEM,* ID *is a fixed-length output function and* H *is a random oracle, then* HPK[KEM, ID, H] *is an anonymous KEM. In other words,*

$$\text{ANO-CCA} + \text{OW-CCA KEM} \implies \text{ANO-CCA HPK[KEM, ID, H]}.$$

*Proof.* Let $\mathsf{KEM}' := \mathsf{HPK}[\mathsf{KEM}, \mathsf{ID}, \mathsf{H}]$. We show that for any PPT adversary $\mathcal{A}$

$$\left|\Pr\left[\mathsf{Exp}^{\text{ANO-CCA}}_{\mathcal{A},\mathsf{KEM}'}(\lambda) = 1 \mid b = 0\right] - \Pr\left[\mathsf{Exp}^{\text{ANO-CCA}}_{\mathcal{A},\mathsf{KEM}'}(\lambda) = 1 \mid b = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

We prove this by a sequence of 4 games.

$\mathsf{Game}_0$. This is the $\mathsf{Exp}^{\text{ANO-CCA}}_{\mathcal{A},\mathsf{KEM}'}(\lambda)$ game where $b = 0$. Note that here $k^\dagger \leftarrow \mathsf{H}(\mathsf{ID}(pk_0), k^*)$ where $(c^*, k^*) \leftarrow \mathsf{KEM.Encaps}(pk_0)$.

| $\mathsf{DHIES}^*.\mathsf{KGen}(\lambda)$ | $\mathsf{DHIES}^*.\mathsf{Enc}(pk, m)$ | $\mathsf{DHIES}^*.\mathsf{Dec}(sk, c)$ |
|---|---|---|
| $sk \leftarrow\!\!\$\ \mathbb{Z}_p$ | $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$ | $(c_1, c_2) \leftarrow c$ |
| $pk \leftarrow g^{sk}$ | $c_1 \leftarrow g^r$ | **if** $c_1 = 1_{\mathbb{G}}$ **then** |
| **return** $(pk, sk)$ | $k \leftarrow \mathsf{H}(pk^r)$ | **return** $\perp$ |
| | $c_2 \leftarrow \mathsf{DEM.Enc}(k, m)$ | $k \leftarrow \mathsf{H}(c_1^{sk})$ |
| | **return** $(c_1, c_2)$ | $m \leftarrow \mathsf{DEM.Dec}(k, c_2)$ |
| | | **return** $m$ |

**Figure 8.7:** Description of $\mathsf{DHIES}^*$ [ABN10]. $p$ is a prime, $\mathbb{G}$ is a group of order $p$, $1_{\mathbb{G}} \neq g \in \mathbb{G}$, DEM be a DEM with a key space $\mathcal{K}$, and $\mathsf{H} : \mathbb{G} \to \mathcal{K}$.

$\mathsf{Game}_1$. This game modifies $\mathsf{Game}_0$ s.t. $k^\dagger$ is sampled randomly. $\mathcal{A}$ can only detect this change if it queries $\mathsf{H}(\cdot, k^*)$. This game is indistinguishable from $\mathsf{Game}_0$ due to OW-CCA-security of KEM.

$\mathsf{Game}_2$. This game modifies $\mathsf{Game}_1$ s.t. $c^*$ is now produced using $pk_1$ instead of $pk_0$. This game is indistinguishable from $\mathsf{Game}_1$ due to ANO-CCA-security of KEM.

$\mathsf{Game}_3$ . This game modifies $\mathsf{Game}_2$ s.t. $k^\dagger$ is not sampled randomly, but as in the standard $\mathsf{Exp}^{\text{ANO-CCA}}_{\mathcal{A},\mathsf{KEM}'}(\lambda)$ with $b = 1$, i.e., $k^\dagger \leftarrow \mathsf{H}(\mathsf{ID}(pk_1), k^*), (c^*, k^*) \leftarrow \mathsf{KEM.Encaps}(pk_1)$.
This concludes the proof. □

### 8.1.4 Decryption Robustness of DHIES*

In this section, we prove decryption robustness of the $\mathsf{DHIES}^*$ public-key encryption scheme [ABR99; ABN10] since it was used to originally instantiate PAPKE-IC (Section 8.2.4.1) in [Bra+19a]. It is no surprise that the $\mathsf{DHIES}^*$ satisfies DROB-CCA—it was proven to be SROB-CCA-secure [ABN10]. However, we can get a tighter security bound by directly proving DROB-CCA security.

$\mathsf{DHIES}^*$ (Figure 8.7) is a modification of the original DHIES [ABR99] introduced in [ABN10] in order to provide SROB-CCA security. The difference is that $\mathsf{DHIES}^*$ does not consider the identity element of the Diffie-Hellman group to be a valid "ephemeral public key," i.e., it rejects ciphertexts with the identity element.

**Theorem 8.** *Let* PKE *be the* $\mathsf{DHIES}^*$ *construction from an* INT-CTXT*-secure DEM* DEM*, a prime-order group* $\mathbb{G}, |\mathbb{G}| = p$ *with a generator* $g$*, and a random oracle* $\mathsf{H} : \mathbb{G} \to \{0,1\}^k$*. For any PPT adversary* $\mathcal{A}$ *against* DROB-CCA *of* $\mathsf{DHIES}^*$*, there exists a PPT adversary* $\mathcal{B}$ *against* INT-CTXT *of* DEM *s.t.*

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{q_\mathsf{H} \cdot q_D}{p} + q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda),$$

*where* $q_\mathsf{H}$ *and* $q_D$ *are the numbers of oracle queries to* H *and* Dec*, respectively.*

*Proof.* In this proof we use a similar strategy as in the proof of Theorem 5. Figure 8.8 provides a detailed description of the games. Let $\mathsf{Game}_0$ be the original DROB-CCA game for DHIES*.

$\mathsf{Game}_1$. In $\mathsf{Game}_1$, we replace the random oracle H used in Dec with an internal random oracle F that is inaccessible to $\mathcal{A}$. The two games are identical unless $\mathcal{A}$ queries $c_1^{sk}$ to H where $c_1$ is the first part of a ciphertext submitted to Dec during the game. There are at most $q_D$ different values for $c_1$. Therefore, for each H query, the $\mathcal{A}$ has probability at most $\frac{q_D}{p}$ of hitting any $c_1^{sk}$ since $sk \leftarrow\!\!\$\ \mathbb{Z}_p$, and it is not used anywhere else in the experiment. In other words,

$$|\Pr[\mathsf{Game}_1] - \Pr[\mathsf{Game}_0]| \leq \frac{q_{\mathsf{H}} \cdot q_D}{p}.$$

Similarly as in the proof of Theorem 5, observe that any successful $\mathcal{A}$ in $\mathsf{Game}_1$ must submit a decrypting ciphertext $(c_1, c_2)$ to Dec during the experiment. In $\mathsf{Game}_1$, the DEM key $k$ is derived using the internal random oracle F; therefore, $\mathcal{A}$ has no information about the key. Therefore, $\mathcal{A}$ can be used by an adversary $\mathcal{B}$ against INT-CTXT of DEM in the same manner as in the proof of Theorem 5. So,

$$\Pr[\mathsf{Game}_1] \leq q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda).$$

In total, we have

$$\mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \frac{q_{\mathsf{H}} \cdot q_D}{p} + q_D \cdot \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B},\mathsf{DEM}}(\lambda).$$

□

## 8.2 Password-Authenticated Public-Key Encryption

A Password-Authenticated Public-Key Encryption scheme (PAPKE) is a novel cryptographic primitive introduced in [Bra+19b]. PAPKE is closely related to the public-key encryption primitive; specifically, it extends the standard PKE scheme with a password-based authentication mechanism. The authentication mechanism ensures that the encrypting party must know the password, which was used by the recipient.

We define PAPKE formally in Definition 40.

**Definition 40** (Password-Authenticated Public-Key Encryption Scheme). Let $\mathcal{SK}$, $\mathcal{APK}$, $\mathcal{PW}$, $\mathcal{M}$, $\mathcal{C}$ be finite sets. A *password-authenticated public-key encryption scheme* is a triple of PPT algorithms (KGen, Enc, Dec) s.t.

- $\mathsf{KGen} : \mathbb{N} \times \mathcal{PW} \to \mathcal{SK} \times \mathcal{APK}$ is the *key generation algorithm*. It takes a security parameter $\lambda \in \mathbb{N}$ and a password $pw \in \mathcal{PW}$ as inputs and outputs an *authenticated keypair* $(sk, apk) \in \mathcal{SK} \times \mathcal{APK}$; $sk$ is called the *secret key* or the *private key*, $apk$ is called the *authenticated public key*.

**Games 0-2**

$\mathsf{Exp}^{\text{DROB-CCA}}_{\mathcal{A},\mathsf{PKE}^{hy}[\mathsf{DHIES}^*,\mathsf{DEM}]}(\lambda)$

$\mathsf{win} \leftarrow 0$

$sk \leftarrow\!\!\$\ \mathbb{Z}_p$

$pk \leftarrow g^{sk}$

$\mathcal{A}^{\mathsf{Dec}_{sk},\mathsf{H}}(\lambda)$

**return** win

$H(x)$

**if** $\exists(x, K) \in \mathsf{List}_\mathsf{H}$

  **return** $K$

$K \leftarrow\!\!\$\ \{0,1\}^k$

add $(x, K)$ to $\mathsf{List}_\mathsf{H}$

**return** $K$

$\mathsf{Dec}_{sk}(c)$

$(c_1, c_2) \leftarrow c$

**if** $c_1 = 1$ **then**

  **return** $\perp$

$k \leftarrow \mathsf{H}(c_1^{sk})$ $\quad \mathsf{G}_0$

$k \leftarrow \mathsf{F}(c_1^{sk})$ $\quad \mathsf{G}_{\geq 1}$

$m \leftarrow \mathsf{DEM.Dec}(k, c_2)$

**return** $m$

$F(x)$

**if** $\exists(x, K) \in \mathsf{List}_\mathsf{F}$

  **return** $K$

$K \leftarrow\!\!\$\ \{0,1\}^k$

add $(x, K)$ to $\mathsf{List}_\mathsf{F}$

**return** $K$

**Figure 8.8:** Sequence of games for the proof of Theorem 8.

- Enc : $\mathcal{APK} \times \mathcal{PW} \times \mathcal{M} \to \mathcal{C}$ is the *encryption algorithm*. It takes an authenticated public key $apk \in \mathcal{APK}$, a password $pw \in \mathcal{PW}$ and a *plaintext* $m \in \mathcal{M}$ as inputs and outputs a *ciphertext* $c \in \mathcal{C}$.
- Dec : $\mathcal{SK} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ is the *decryption algorithm*. It takes a private key $sk \in \mathcal{SK}$ and a *ciphertext* $c \in \mathcal{C}$ as inputs and outputs a *plaintext* $m \in \mathcal{M}$, or a *decryption error* denoted as $\perp$.

$\mathcal{SK}$ is called the *private key space*, $\mathcal{APK}$ is called the *authenticated public key space*, $\mathcal{PW}$ is called the *password space* or the *password dictionary*, $\mathcal{M}$ is called the *plaintext space* or the *message space* and $\mathcal{C}$ is called the *ciphertext space*.

The authors [Bra+19b] provide two categories of security definitions for PAPKE.

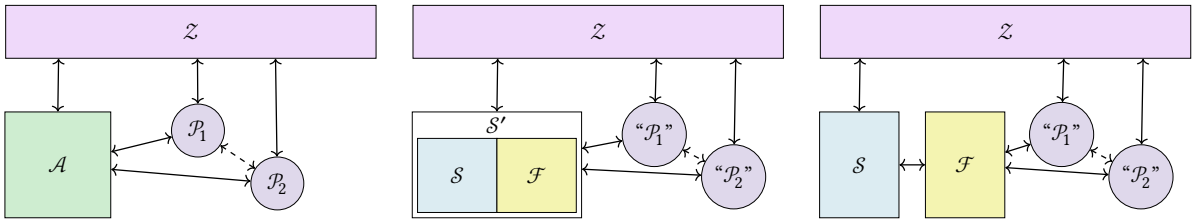
The first category is "standard" game-based definitions, i.e., those that we have used so far to define security of PKEs, KEMs and others. Game-based definitions are formulated using a security experiment/game that is "played" by an adversary. The scheme is said to be secure if any efficient adversary cannot win the game with a probability $> \mathsf{negl}(\lambda) + T$, where, in the case of password-based schemes, $T$ is some value that is tied to the password distribution.

The second category is security definitions within the Universal Composability (UC) framework [Can01]. UC security PAPKE definitions are stronger than game-based definitions—UC security properties imply game-based properties [Bra+19b, Theorem 1]. Consequently, we only focus on UC PAPKE properties, which we present in Section 8.2.2. Let us first introduce the UC framework in Section 8.2.1.

### 8.2.1 Universal Composability

The Universal Composability (UC) framework is a security framework introduced in [Can01]. The aim of the framework is to unify description and security properties of cryptographic protocols and primitives. This unification allows for modular security analysis. For example, let us consider a PAKE protocol built using the PAPKE primitive. As we present later in Sections 8.2.4 and 8.3, there are multiple PAPKE variants: PAPKE-IC, PAPKE-FO, and PAPKE-HIC. In other security models, each PAPKE-based PAKE protocol variant would need to be analyzed individually whereas in UC a single analysis is enough. Another benefit of UC is that it models stronger security guarantees; for example, the game-based PAKE model does not model correlated passwords—a situation where a user executes the protocol twice, but mistypes the password once.

In UC, the security properties of a primitive, or a protocol, are expressed as an *ideal functionality*. The ideal functionality describes the interface and the idealized behavior of the primitive. Note that ideal functionalities are typically in the form of pseudocode. To prove security in the UC framework, we must show that the specific construction, a primitive or a protocol, is indistinguishable from the ideal functionality. We say that a construction $\Pi$ *UC-realizes* $\mathcal{F}$, meaning that the construction $\Pi$ is indistinguishable from the ideal functionality $\mathcal{F}$. The distinguisher in UC is referred to as the *environment*, typically denoted as $\mathcal{Z}$. The goal is to show that no efficient environment can distinguish between the "real world," in which the environment interacts with parties executing the real protocol $\Pi$, and the "ideal world," where the environment interacts with a simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}$.




(a) Real World (b) Pre-Ideal World (c) Ideal World

**Figure 8.9:** UC-Proof Strategy.

**Remark**

An ideal functionality models both a protocol, which is a set of interactive algorithms, and a primitive/scheme, which is a set of non-interactive algorithms. In UC, schemes are transformed into protocols. See [Can05, Section 7] for some example transformations.

Let us elaborate more on the difference between the real world and the ideal world which is visualized in Figure 8.9.

In the *real world* (Figure 8.9a), there is the *environment* $\mathcal{Z}$, the *real-world adversary* $\mathcal{A}$ and parties $\mathcal{P}_1, \mathcal{P}_2, \ldots$ The parties $\mathcal{P}_i$ are initialized by $\mathcal{Z}$, they execute the protocol $\Pi$ honestly and output the result to $\mathcal{Z}$. However, the parties can communicate only through the real-world adversary $\mathcal{A}$, which acts as a man-in-the-middle, i.e., it can delay, suppress, read, insert and modify those messages. The real-world adversary $\mathcal{A}$ also has access to the inputs used by $\mathcal{Z}$ since without loss of generality $\mathcal{A}$ can be considered a "dummy" adversary controlled by $\mathcal{Z}$ (see Section 8.2.1.1 for details). The real-world adversary $\mathcal{A}$ can also "corrupt" the protocol parties—the definition of corruption depends on the protocol.

In the *ideal world* (Figure 8.9c), there is the environment $\mathcal{Z}$, the *simulator* $\mathcal{S}$, which is sometimes referred to as the *ideal-world adversary*, the ideal functionality $\mathcal{F}$ and "dummy" parties. The dummy parties, denoted as "$\mathcal{P}_i$", only forward messages between $\mathcal{Z}$ and $\mathcal{F}$. The simulator's purpose is to simulate the real-world adversary while interacting only with the ideal functionality $\mathcal{F}$ and $\mathcal{Z}$. The simulator $\mathcal{S}$ essentially serves as a translation layer between the environment and the ideal functionality.

The crux is that the simulator $\mathcal{S}$ does not have access to the secret initial inputs provided by $\mathcal{Z}$ (those are used by $\mathcal{F}$), and it must simulate the real-world adversary without them. Intuitively, the existence of a simulator implies that any real-protocol adversary is unable to get any additional information other than what is specified by the ideal functionality.

The typical UC-proof strategy is to consider a sequence of games where the first game corresponds to the real-world protocol and that protocol description gets progressively transformed into a "super-simulator" $\mathcal{S}'$ which can be decomposed into two parts: the ideal functionality $\mathcal{F}$ and the ideal-world simulator $\mathcal{S}$. Our proof of PAPKE-HIC (Section 8.3) follows this strategy visualized in Figure 8.9.

**Definition 41** (UC-Security). We say that a scheme/protocol $\Pi$ *UC-realizes* $\mathcal{F}$, if for any efficient real-world adversary $\mathcal{A}$ there exists an efficient simulator $\mathcal{S}$ s.t. for any efficient adversary $\mathcal{Z}$,

referred to as the *environment*, the following holds

$$\left|\Pr\left[\mathsf{Real}_{\mathcal{Z},\mathcal{A},\Pi}(\lambda)\right] - \Pr\left[\mathsf{Ideal}_{\mathcal{Z},\mathcal{S},\mathcal{F}}(\lambda)\right]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Real}_{\mathcal{Z},\mathcal{A},\Pi}(\lambda)$ denotes the event when $\mathcal{Z}$ outputs 1 while interacting with $\mathcal{A}$ and honest parties executing the scheme/protocol $\Pi$, and $\mathsf{Ideal}_{\mathcal{Z},\mathcal{S},\mathcal{F}}(\lambda)$ denotes the event when $\mathcal{Z}$ outputs 1 while interacting with $\mathcal{S}$, $\mathcal{F}$ and "dummy parties," which only forward messages between $\mathcal{Z}$ and $\mathcal{F}$.

The unfortunate consequence of UC's universality and modularity is that sometimes it is difficult to intuitively understand the security properties implied by the ideal functionality compared to game-based definitions. On top of that, the universal nature of UC leads to more complex proofs compared to standard game-based proofs.

The UC model allows the environment to run many concurrent sessions of the protocol. Formally, this is achieved by each protocol session having a unique session identifier (*sid*). Note that the model assumes that session parties know this session ID in advance. Therefore, it is typically implicitly assumed that the protocol execution is preceded by an exchange of nonces. This requirement makes real-world implementations more complex, and it is generally unknown whether the omission of the identifier has an impact on the security of the protocol.

#### 8.2.1.1 Dummy Adversary

In the standard UC security definition (Definition 41), the simulator $\mathcal{S}$ depends on $\mathcal{A}$, but is independent of $\mathcal{Z}$, i.e., the simulator knows the code[3] of $\mathcal{A}$, but not of $\mathcal{Z}$.

The definition (Definition 41) can be simplified by fixing the real-world adversary $\mathcal{A}$ to be a "dummy adversary" that simply forwards messages between the environment and the protocol parties. Intuitively, the reason why the definitions are equivalent is that the dummy adversary is the hardest to simulate since everything is controlled by the independent environment. See [Can01, Section 4.3.1] for details. However, modeling $\mathcal{A}$ and $\mathcal{Z}$ as separate provides a better intuition about the model's security implications. For example, the environment may choose to not share some information with $\mathcal{A}$, who in the real-world has access to the protocol, and expect a specific reply, which implies that the simulator in ideal-world, without access to the same protocol messages (only the "ideal messages"), must also simulate such reply correctly.

#### 8.2.1.2 Hybrid Protocols

The modularity of UC allows a protocol $\Pi$, that UC-realizes $\mathcal{F}$, to depend on some other ideal "sub-functionality" $\mathcal{F}'$. For example, if a protocol uses an ideal cipher (Section 8.2.3.2), then the protocol is modeled as having access to the ideal-cipher functionality $\mathcal{F}_{\mathsf{IC}}$. Such protocol $\Pi$ is called a *hybrid protocol*—we say that $\Pi$ UC-realizes $\mathcal{F}$ in the $\mathcal{F}'$*-hybrid model*.

Compared to the "regular" UC model, the hybrid model is modified as follows. In the real world, the protocol parties can additionally directly communicate with the ideal sub-functionality $\mathcal{F}'$. The sub-functionality may "leak" some information to $\mathcal{A}$, which also has access to the sub-functionality. In the ideal world, however, since the dummy parties only relay messages between the "main" ideal functionality $\mathcal{F}$ and the environment, the dummy parties do not communicate

[3] Recall that formally every entity is a Turing machine defined by a set of instructions.

with the ideal sub-functionality. Consequently, only the simulator communicates with the sub-functionality, emulating the real-world $\mathcal{A}$ queries. This implies that the simulator can record all inputs and outputs to the sub-functionality and the simulator can actually emulate the sub-functionality itself. Therefore, the sub-functionality does not actually exist in the ideal world. We illustrate this in Figure 8.17.

#### 8.2.1.3 Parameterized Ideal Functionalities

Ideal functionalities are typically "parameterized." For example, the original $\mathcal{F}_{\mathsf{HIC}}$ functionality from [SGJ23] is said to be parameterized by the domain $\mathcal{D}$, meaning that formally there is not one ideal functionality covering all possible domains, but each is formally distinct depending on the domain. This distinction allows for finer and more realistic modeling. For instance, if $\mathcal{F}_{\mathsf{HIC}}$ is not parameterized over the domain $\mathcal{D}$, it would model that each HIC instantiation supports any group. This would make it impossible to prove that a specific construction, such as M2F, UC-realizes $\mathcal{F}_{\mathsf{HIC}}$.

**Remark**

In our ideal functionality descriptions, we follow the more traditional description where the adversary is indicated as $\mathcal{A}$. However, this is not entirely accurate since the ideal-world adversary is the simulator $\mathcal{S}$. On the other hand, in the $\mathcal{F}$-hybrid model, the ideal sub-functionality communicates with the real-world adversary $\mathcal{A}$.

### 8.2.2 PAPKE Security

Informally, we require the following properties from a PAPKE scheme [Bra+19b]:

- *Secrecy*: Honestly encrypted ciphertexts hide the message against a chosen-ciphertext adversary who also knows the password.
- *Offline-Dictionary Attack Resistance*: Given a ciphertext or an authenticated public key (these values are tied to the password), an adversary is unable to recover the corresponding password using an offline-dictionary attack.
- *Active-Attack Resistance*: An authenticated public key $apk$ is tied to one password $pw$ and all ciphertexts encrypted using this $apk$, but with a different password $pw' \neq pw$, hide the encrypted message. This implies that an active adversary can make at most one password guess per malicious $apk$.
- *Authenticity*: A ciphertext encrypted under an honest authenticated public key, but with a wrong password, is invalid, i.e., it decrypts to $\perp$.

The PAPKE ideal functionality is presented in Figure 8.10. Compared to the original functionality in [Bra+19a], we slightly modify $\mathcal{F}_{\mathsf{PAPKE}}$ to ensure that the messages ENC-M and ENC-L also send $apk'$ to $\mathcal{A}$ together with $m$ and $|m|$, respectively. We believe this is an accidental omission in the original definition, since in the proofs of PAPKE-IC and PAPKE-FO the authors write "... (ENC-M, $sid$, $apk'$, $m$) from $\mathcal{F}_{\mathsf{PAPKE}}$ ..." In our proof of PAPKE-HIC (Section 8.3), we observe

**Key Generation**. On query (KEYGEN, $sid$, $pw$) from $\mathcal{P}$:

If $sid \neq (\mathcal{P}, sid')$ or a record $(sid, \cdot, \cdot) \in \text{List}_{\text{keyrec}}$ exists, ignore.

Send (KEYGEN, $sid$) to $\mathcal{A}$ and wait for (KEYCONF, $sid, apk, \cancel{\mathcal{M}}$) from $\mathcal{A}$.

If a record $(sid, apk, \cdot) \in \text{List}_{\text{badkeys}}$ exists, abort.

Create a record $(sid, apk, pw)$ in $\text{List}_{\text{keyrec}}$.

Output (KEYCONF, $sid, apk$) to $\mathcal{P}$.

**Encryption**. On query (ENCRYPT, $sid, apk', pw', m$) from $\mathcal{Q}$ where $m \in \mathcal{M}$:

If a record $(sid, apk', pw) \in \text{List}_{\text{keyrec}}$ exists and $\mathcal{P}$ is honest where $(\mathcal{P}, sid') \leftarrow sid$, then:

Send (ENC-L, $sid, apk$', $|m|$) to $\mathcal{A}$ and wait for (CIPHERTEXT, $sid, c$) from $\mathcal{A}$.

If a record $(sid, \cdot, c) \in \text{List}_{\text{encrec}}$ exists, abort.

Create a record $(sid, m', c)$ in $\text{List}_{\text{encrec}}$ where $m' \leftarrow m$ if $pw = pw'$, and $m' \leftarrow \perp$ otherwise.

Else:

If a record $(sid, apk', pw) \in \text{List}_{\text{badkeys}}$ exists, set $pw^* \leftarrow pw$.

Else, send (GUESS, $sid, apk'$) to $\mathcal{A}$, wait for (GUESS, $sid$, $pw^*$) from $\mathcal{A}$ and create a record $(sid, apk', pw^*)$ in $\text{List}_{\text{badkeys}}$.

If $pw' = pw^*$, send (ENC-M, $sid, apk$', $m$) to $\mathcal{A}$ and wait for (CIPHERTEXT, $sid, c$) from $\mathcal{A}$.

Else, send (ENC-L, $sid, apk$', $|m|$) to $\mathcal{A}$ and wait for (CIPHERTEXT, $sid, c$) from $\mathcal{A}$.

Output (CIPHERTEXT, $sid, c$) to Q.

**Decryption**. On query (DECRYPT, $sid, c$) from $\mathcal{P}$:

If $sid \neq (\mathcal{P}, sid')$ or no record $(sid, apk, pw) \in \text{List}_{\text{keyrec}}$ exists, ignore.

If a record $(sid, m, c) \in \text{List}_{\text{encrec}}$ and $m \in \mathcal{M} \cup \{\perp\}$, then:

Output (PLAINTEXT, $sid, m$) to $\mathcal{P}$.

Else:

Send (DECRYPT, $sid, c$) to $\mathcal{A}$ and wait for (PLAINTEXT, $sid, m^*, pw^*$) from $\mathcal{A}$.

If $pw = pw^*$, set $m \leftarrow m^*$, and $m \leftarrow \perp$ otherwise.

Create a record $(sid, m, c) \in \text{List}_{\text{encrec}}$.

Output (PLAINTEXT, $sid, m$) to $\mathcal{P}$.

**Figure 8.10:** PAPKE ideal functionality $\mathcal{F}_{\text{PAPKE}}$. The gray text indicates patches to the (incorrect) original functionality from [Bra+19b]. The functionality is parametrized by the message space $\mathcal{M}$.

that sending $apk'$ in ENC-M is necessary because the simulator must encrypt the message honestly and otherwise cannot determine with certainty the correct $apk'$ to use.

Note that $\mathcal{F}_{\text{PAPKE}}$ implies all the security properties mentioned above. Let us discuss why. $\mathcal{F}_{\text{PAPKE}}$ has three interfaces corresponding to the PAPKE primitive interface: KEYGEN ENCRYPT and DECRYPT. As standard in UC, the interfaces require a session ID *sid* as input. In $\mathcal{F}_{\text{PAPKE}}$ the $sid = (\mathcal{P}, sid')$ is composed of a UC-standard unique string $sid'$ and a sender identity $\mathcal{P}$. Observe that the KEYGEN and DECRYPT interfaces are only accessible to the party indicated in the *sid* whereas ENCRYPT is accessible by any party—anyone with an authenticated public key can encrypt. This is a consequence of the "ignore" instructions.

The secrecy of ciphertexts is implied by what the functionality sends to the adversary $\mathcal{A}$ inside ENCRYPT.

If the functionality is queried to encrypt $m$ under an honestly generated authenticated public key, the adversary only learns the length of the message (content of the message ENC-L). The adversary produces a corresponding ciphertext that is saved for decryption; the ciphertext is "marked as valid" only if the queried password matches the honest password—this matches the scenario where an adversary attempts to encrypt a message using an honest $apk$ and a password guess.

If the functionality is queried to encrypt $m$ under a dishonest authenticated public key $apk'$, the adversary gets a single password guess (the message GUESS) for this $apk'$. The fact that the adversary gets a single password guess per authenticated public key is achieved by keeping the "bad keys" records. If the adversary guesses the correct password, it learns the content of the message. This corresponds to the case in which an adversary crafts a malicious $apk'$ under a password guess and an honest party uses it with its password—if the adversary decrypts the ciphertext into a valid message, then it knows the password guess is correct.

The authenticity of ciphertexts is implied by the behavior of DECRYPT. If a ciphertext is a result of honest encryption (an encrec record exists), the functionality outputs $m \neq \perp$ if and only if the password used to encrypt was correct. Otherwise, decryption outputs $m \neq \perp$ only in the case that the encryption was done using a dishonest ciphertext and the adversary guessed the correct password.

The password-guess resistance is indicated by the two messages, containing a password guess $pw^*$, the functionality receives from $\mathcal{A}$: GUESS inside ENCRYPT and PLAINTEXT inside DECRYPT. This models the two possible online attacks: the adversary crafts a malicious authenticated public key using a password guess, gets an honest party to use it for encryption and attempts to decrypt it, and the adversary uses the honest authenticated public key and uses it to encrypt a message with a password guess and tests the guess by having an honest party decrypt it.

### 8.2.3 Idealized Models

In the field of provable security, it is sometimes necessary to work with idealized models of real-world constructions. The security of the PAPKE constructions that we present in this chapter relies on such idealized constructions. We first present the established Random Oracle Model (ROM) in Section 8.2.3.1, and then we introduce the Ideal Cipher Model (ICM) in Section 8.2.3.2 and the novel Half-Ideal Cipher Model (HICM) in Section 8.2.3.3.

#### 8.2.3.1 Random Oracle Model

The most established idealized model is the *Random Oracle Model* (ROM) introduced in [BR93]. By working in the ROM, we assume that there exists an idealized version of a hash function called *random oracle*. It is an oracle, accessible to all parties, that implements a uniformly random function, that is, a function sampled uniformly from the set of all functions over a specific domain and codomain. The random oracle can be thought of as a black box that outputs a uniformly random value from the set of possible outputs on each input while record keeping (ensuring the same input leads to the same output).

Although this is how we typically imagine secure hash functions operate, the existence of a random oracle is a strictly stronger assumption than the existence of a collision-resistant and pre-image-resistant hash function.[4] Additionally, it has been shown that there exist constructions proven secure in the ROM that cannot be securely instantiated [CGH04]. Nonetheless, the ROM is a widely accepted assumption and random oracles are instantiated using secure hash functions, such as SHA-2 and SHA-3.

Note that proofs that do not consider the ROM (or any other idealized assumptions) are said to be in the *standard model*.

#### 8.2.3.2 Ideal Cipher Model

Analogous to the random oracle, which is an idealized model of a secure hash function, the *Ideal Cipher Model* (ICM) considers the existence of an ideal symmetric encryption scheme.

Formally, an ideal cipher is a family of permutations over some message space $\mathcal{M}$ indexed by a key $k \in \mathcal{K}$, where the family is chosen uniformly at random from the set of all such families, i.e., an ideal cipher is a function $\mathsf{IC} : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ s.t. $\forall k \in \mathcal{K} : \mathsf{IC}(k, \cdot)$ is a random permutation over $\mathcal{M}$. We can think of an ideal cipher as a black box that samples a fresh ciphertext/plaintext for each encryption/decryption query and keeps records for consistency. See Figure 8.11 for its corresponding UC functionality.

Similar to the ROM, the ICM becomes problematic when we want to instantiate the ideal cipher in practice. The difficulty of instantiating an ideal cipher depends on the message space $\mathcal{M}$. While a symmetric cipher is considered the real-world equivalent of an ideal cipher, symmetric ciphers, such as AES, operate over bitstrings of fixed size, e.g., the AES block size is 128 bits. Consequently, if we need an ideal cipher over the message space $\mathcal{M} = \{0,1\}^n, n < 128$, then the instantiation is "easy." However, instantiating an ideal cipher over any other message space becomes more complex.

Let us distinguish between bitstring message spaces $\mathcal{M} = \{0,1\}^n, n > 128$ and group message spaces $\mathcal{M} = \mathbb{G}$, where $\mathbb{G}$ is a group. The existence of an ideal cipher over a specific group is a common assumption because of Diffie-Hellman-based constructions. Note that the bitstring space is also a group.

**Ideal Cipher over Bitstrings.** There are essentially two approaches to instantiating an ideal cipher over $\mathcal{M} = \{0,1\}^n, n > 128$.

[4] For example, a random oracle output is assumed to be unpredictable whereas a secure hash function can have a fixed (predictable) prefix.

**Encryption**. On query $(\text{ENCRYPT}, sid, k, m)$ from $\mathcal{P}$ where $k \in \mathcal{K}, m \in \mathcal{M}$:

If a record $(sid, k, m, c) \in \text{List}_{\text{IC}}$ exists, output $(\text{CIPHERTEXT}, sid, c)$ to $\mathcal{P}$.

Else:

Do:

$c \leftarrow\$ \mathcal{M}$.

While: A record $(sid, k, *, c) \in \text{List}_{\text{IC}}$ exists.

Create a record $(sid, k, m, c)$ in $\text{List}_{\text{IC}}$.

Output $(\text{CIPHERTEXT}, sid, c)$ to $\mathcal{P}$.

**Decryption**. On query $(\text{DECRYPT}, sid, k, c)$ from $\mathcal{P}$ where $k \in \mathcal{K}, c \in \mathcal{M}$:

If a record $(sid, k, m, c) \in \text{List}_{\text{IC}}$ exists, output $(\text{PLAINTEXT}, sid, m)$ to $\mathcal{P}$.

Else:

Do:

$m \leftarrow\$ \mathcal{M}$.

While a record $(sid, k, m, *) \in \text{List}_{\text{IC}}$ exists.

Create a record $(sid, k, m, c) \in \text{List}_{\text{IC}}$.

Output $(\text{PLAINTEXT}, sid, m)$ to $\mathcal{P}$.

**Figure 8.11:** Ideal Cipher (IC) ideal functionality $\mathcal{F}_{\text{IC}}$. The functionality is parametrized by the key space $\mathcal{K}$ and the message space $\mathcal{M}$.

1) *Domain Extender*: The first approach is to use the *Domain Extender* construction proposed in [Cor+10] and generalized in [GL15]. The Domain Extender construction takes an ideal cipher with a smaller domain and extends the domain by a specific factor $t$. In practice, we use the baseline ideal cipher with 128-bit blocks (AES) to build an ideal cipher with $(128 \cdot t)$-bit blocks. The original construction from [Cor+10] has complexity $\mathcal{O}(t^{\log_2(3)})$ whereas the generalized construction [GL15] is linear ($\mathcal{O}(t)$).

2) *8-Round Feistel*: The second approach is to use the *8-Round Feistel* construction proposed in [DS16]. It has been shown that an 8-round Feistel network, where each round function is an independent random function $\{0,1\}^n \to \{0,1\}^n$, is an ideal cipher over $\{0,1\}^{2 \cdot n}$. Assuming that an XOF (Definition 8) behaves as a random oracle (Section 8.2.3.1), we can build an ideal cipher over arbitrary bitstrings by instantiating the round functions using the XOF (with a prefix consisting of the key and domain separation constants). The complexity of this construction is constant in the number of XOF invocations, however, the actual complexity is hidden inside the XOF—the speed of XOF invocation depends on the output size.

**Ideal Cipher over Groups.** Let $\mathbb{G}$ be a group. The naive approach to building an ideal cipher over $\mathbb{G}$ is to canonically encode its elements as bitstrings of length $n$ for some $n \in \mathbb{N}$ and use an ideal cipher over bitstrings of length $n$. Unless the size of $\mathbb{G}$ is a power of 2, the size of the sets $\mathbb{G}$ and $\{0,1\}^n$ does not match ($|\mathbb{G}| < 2^n$). Consequently, any encoding cannot be a bijection. This means that there exist bitstrings that do not represent valid elements of the group. This is a problem in scenarios where the ideal cipher key is a password, i.e., a low-entropy value. As a result, an

adversary, given a ciphertext $c$, can do an offline-dictionary attack: discard candidate passwords that decrypt $c$ into invalid bitstrings.

Essentially, there are three approaches to securely instantiating an ideal cipher over a group:

1) *Loop-Sampling*: This approach, proposed in [BR02], fixes the naive approach by requiring that every valid ciphertext must represent a group element. Let $g \in \mathbb{G}$ and denote by $c_0 \in \{0,1\}^n$ its bitstring representation. Let $\mathsf{IC}$ be an ideal cipher over bitstrings of length $n$ and $k \in \mathcal{K}$ its corresponding key. To encrypt $g$, one iteratively encrypts $c_i$ until $c_{i+1} \leftarrow \mathsf{IC.Enc}(k, c_i), i \in \mathbb{N}$ represents a group element. Analogously, one iteratively decrypts $c$ until it decrypts into a valid group element.

   The problem with this approach is that its efficiency depends on the gap size between $\{0,1\}^n$ and $\mathbb{G}$ and that it is not constant-time, which makes it vulnerable to side-channel attacks—one can discard candidate passwords based on the time it takes to decrypt.

2) *Quasi-Bijective Encoding*: This approach assumes the existence of a "better" encoding between group elements and bitstrings, compared to the naive (canonical) encoding used in previous approaches. Such an encoding is called a *quasi-bijective encoding* [GJK21]. We do not define it formally here, but essentially, a quasi-bijective encoding maps the uniform distribution on group elements onto a uniform distribution on bitstrings of appropriate size. Consequently, a uniformly sampled ciphertext (a bitstring) will be "decodable" with overwhelming probability, which prevents the offline-dictionary attack. In total, the approach is to combine a quasi-bijective encoding with an ideal cipher over bitstrings.

   The simplest example of a quasi-bijective encoding is the canonical encoding between $\mathbb{Z}_q$ and $\{0,1\}^n$ where $n = \left\lceil \log_2 q \right\rceil$ when "$q$ is close to $2^n$." In that case, the ratio of "invalid" bitstrings compared to "valid" bitstrings is $\frac{2^n - q}{q}$ which is "negligible" for large $q$—a uniform bitstring is valid with overwhelming probability.

   For elliptic curve groups, there exist two such encodings: Elligator [Ber+13] and Elligator Squared [Tib14]. Elligator works only for certain type of elliptic curves and only half of the group elements are "encodable" whereas Elligator Squared works for any curve, but it is more complex and produces larger bitstrings.

   For ML-KEM public keys, which form a group, there also exists a quasi-bijective encoding called Kemeleon [GSV24; GSV26].

   Ultimately, the downside of this approach is that there is no general approach how to construct such an encoding and the encoding process is typically the bottleneck.

3) *8-Round Feistel over Groups*: This approach is a modification of the 8-Round Feistel approach presented in the previous section as a way to construct an ideal cipher over arbitrary bitstrings. The 8-round Feistel network can be modified to carry a group element on one wire instead of the standard bitstring. Consequently, this requires replacing four of the round functions over $\{0,1\}^n$ with *hash-to-group* functions. A hash-to-group function is an analog of a hash function where the output space is a group.

   Observe that we can build a hash-to-group function by combining the decoding part of a quasi-bijective encoding with an XOF, assuming the XOF is a random oracle: we feed

the input through the XOF to produce sufficiently many unpredictable bits which are then decoded into a random group element.

We compare the performance of some of these approaches in Section 8.4.

#### 8.2.3.3 Half-Ideal Cipher Model

The *Half-Ideal Cipher Model* (HICM) is a relaxation of the Ideal Cipher Model introduced in [SGJ23]. The *Half-Ideal Cipher* (HIC) operates over an extended domain $\mathcal{D} = \mathcal{R} \times \mathbb{G}$, where $\mathcal{R} = \{0,1\}^n, n > 2 \cdot \lambda$ is the *randomness space*, compared to an ideal cipher operating over a group $\mathbb{G}$ (the HIC is meant to be a replacement for IC over groups in certain contexts).

A HIC is a function $\mathsf{HIC} : \mathcal{K} \times \mathcal{R} \times \mathbb{G} \to \mathcal{R} \times \mathbb{G}$, s.t. $\forall k \in \mathcal{K} : \mathsf{HIC}(k, \cdot, \cdot)$ is a permutation over $\mathcal{D}$, but it is not a *random* permutation. It is not a random permutation because the model allows an adversary to choose part of the ciphertext. See Figure 8.12 for the HIC ideal functionality $\mathcal{F}_{\mathsf{HIC}}$.

Where applicable, a HIC is superior to an IC because there exists a more efficient construction called the *Modified 2-Feistel* (M2F) [SGJ23] defined in Definition 42.

**Definition 42.** Let $\mathsf{H} : \mathcal{PW} \times \mathcal{R} \to \mathbb{G}$ and $\mathsf{G} : \mathcal{PW} \times \mathbb{G} \to \mathcal{R}$ be random oracles and let $\mathsf{IC} = (\mathsf{IC.Enc}, \mathsf{IC.Dec})$ be an ideal cipher over $\mathcal{R}$. A Modified 2-Feistel (M2F) construction is a tuple of PPT algorithms $\mathsf{M2F} = \mathsf{M2F}[\mathsf{H}, \mathsf{G}, \mathsf{IC}] = (\mathsf{Enc}, \mathsf{Dec})$ s.t. $\mathsf{M2F.Enc}/\mathsf{M2F.Dec} : \mathcal{PW} \times \mathcal{R} \times \mathbb{G} \to \mathcal{R} \times \mathbb{G}$ where $\mathcal{PW}$ is the password dictionary, $\mathcal{R} = \{0,1\}^n$, $\mathbb{G}$ is a group and $\mathcal{K}$ is the IC key space. The descriptions of M2F.Enc and M2F.Dec are in Figure 8.13.

M2F uses three components: a hash-to-group H, a "regular" hash function G and an ideal cipher IC. Both H and IC operate over relatively small domains, e.g., over 256-bit strings at 128-bit security. Let us compare M2F and ideal cipher over group constructions from Section 8.2.3.2.

Compared to approach 1), M2F is superior because it is constant-time, assuming its components (an IC and hashes) are constant-time. It also outperforms approach 2) as it does not require the existence of a quasi-bijective encoding and an IC over potentially large bitstrings. Compared to approach 3), M2F requires only one hash-to-group invocation instead of four.

### 8.2.4 PAPKE Constructions

This section presents two approaches to constructing the PAPKE primitive proposed by the PAPKE authors in [Bra+19b]. The first approach is the PAPKE-IC construction (Section 8.2.4.1) and the second approach is the PAPKE-FO construction (Section 8.2.4.2).

#### 8.2.4.1 PAPKE-IC

The PAPKE-IC construction combines two primitives: a public-key encryption (PKE) scheme (Definition 9) and an ideal cipher (IC) over the public key space (Section 8.2.3.2). The construction is fairly straightforward: the PKE is modified s.t. the authenticated public key is produced by encrypting the regular public key using the IC and the password; during PAPKE encryption, the authenticated public key is decrypted using the IC and the password, and the resulting public key is then used as the PKE public key. The decryption process is identical to the standard PKE

*Interface for honest parties:.*

**Encryption**. On query $(\text{Enc}, sid, pw, M)$ from $\mathcal{P}$ where $M \in \mathbb{G}$:

$r \leftarrow\!\!\$\ \mathcal{R}$

If a record $(sid, pw, (r, M), c) \in \text{List}_{\text{HIC}}$ exists, output $(\text{CIPHERTEXT}, sid, c)$ to $\mathcal{P}$.

Else:

Do:

$c \leftarrow\!\!\$\ \mathcal{D}$.

While: A record $(sid, pw, *, c) \in \text{List}_{\text{HIC}}$ exists.

Create a record $(sid, pw, (r, M), c)$ in $\text{List}_{\text{HIC}}$.

Output $(\text{CIPHERTEXT}, sid, c)$ to $\mathcal{P}$.

**Decryption**. On query $(\text{Dec}, sid, pw, c)$ from $\mathcal{P}$ where $c \in \mathcal{D}$:

$(r, M) \leftarrow \mathcal{F}_{\text{HIC}}.\text{AdvEnc}(sid, pw, c)$. / Call the internal function AdvDec from the adversarial interface.

Output $(\text{PLAINTEXT}, sid, M)$ to $\mathcal{P}$.

*Interface for the ideal-world adversary and corrupt parties:.*

**Encryption**. On query $(\text{AdvEnc}, sid, pw, (r, M), T)$ from $\mathcal{A}$ where $(r, M) \in \mathcal{D}, T \in \mathbb{G}$:

If a record $(sid, pw, (r, M), c) \in \text{List}_{\text{HIC}}$ exists, output $(\text{CIPHERTEXT}, sid, c)$ to $\mathcal{A}$.

Else:

Do:

$s \leftarrow\!\!\$\ \mathcal{R}$.

While: A record $(sid, pw, *, (s, T)) \in \text{List}_{\text{HIC}}$ exists.

Create a record $(sid, pw, (r, m), (s, T))$ in $\text{List}_{\text{HIC}}$.

Output $(\text{CIPHERTEXT}, sid, (s, T))$ to $\mathcal{A}$.

**Decryption**. On query $(\text{AdvDec}, sid, pw, c)$ from $\mathcal{A}$ where $c \in \mathcal{D}$:

If a record $(sid, pw, m, c) \in \text{List}_{\text{HIC}}$ exists, output $(\text{PLAINTEXT}, sid, m)$ to $\mathcal{A}$.

Else:

Do:

$m \leftarrow\!\!\$\ \mathcal{D}$.

While: A record $(sid, pw, m, *) \in \text{List}_{\text{HIC}}$ exists.

Create a record $(sid, pw, m, c)$ in $\text{List}_{\text{HIC}}$.

Output $(\text{PLAINTEXT}, sid, m)$ to $\mathcal{A}$.

**Figure 8.12:** Half-Ideal Cipher (HIC) ideal functionality $\mathcal{F}_{\text{HIC}}$. The functionality is parametrized by the domain $\mathcal{D} = \mathcal{R} \times \mathbb{G}$.

| M2F[H, G, IC].Enc($pw, r, M$) | M2F[H, G, IC].Dec($pw, s, T$) |
|---|---|
| $R \leftarrow \mathsf{H}(pw, r)$ | $t \leftarrow \mathsf{G}(pw, T)$ |
| $T \leftarrow M \odot R$ | $r \leftarrow \mathsf{IC.Dec}(t, s)$ |
| $t \leftarrow \mathsf{G}(pw, T)$ | $R \leftarrow \mathsf{H}(pw, r)$ |
| $s \leftarrow \mathsf{IC.Enc}(t, r)$ | $M \leftarrow T \odot R^{-1}$ |
| **return** $(s, T)$ | **return** $(r, M)$ |

**Figure 8.13:** The Modified 2-Feistel M2F construction where H is a hash-to-group function, G is a hash function and IC is an ideal cipher, $\odot$ is the group operation in $\mathbb{G}$ and $(\cdot)^{-1}$ is the inverse in $\mathbb{G}$.

| PAPKE-IC.KGen($\lambda$, $pw$) | PAPKE-IC.Enc($apk$, $pw, m$) | PAPKE-IC.Dec($sk, c$) |
|---|---|---|
| $(sk, pk) \leftarrow \mathsf{PKE.KGen}(\lambda)$ | $pk \leftarrow \mathsf{IC.Dec}(pw, apk)$ | $m \leftarrow \mathsf{PKE.Dec}(sk, c)$ |
| $apk \leftarrow \mathsf{IC.Enc}(pw, pk)$ | $c \leftarrow \mathsf{PKE.Enc}(pk, m)$ | **return** $m$ |
| **return** $(sk, apk)$ | **return** $c$ | |

**Figure 8.14:** The PAPKE-IC construction where PKE is a public-key encryption scheme and IC is an ideal cipher over the public key space of PKE.

decryption. See Figure 8.14 for a detailed description of PAPKE-IC built from the public-key encryption scheme PKE and the ideal cipher IC.

The authors of PAPKE [Bra+19b] prove that PAPKE-IC UC-realizes $\mathcal{F}_{\mathsf{PAPKE}}$ in the $\mathcal{F}_{\mathsf{IC}}$-hybrid model (Section 8.2.3.2) if the public-key encryption scheme has perfectly uniform public keys and is AI-CCA and SROB-CCA secure.

"Perfect uniformity of public keys" refers to a special case of pseudo-randomness of public keys (UNI-PK), which we defined in Figure 2.4, where the adversary's advantage is 0. Recall that SROB-CCA refers to *strong robustness* (Figure 2.4). The AI-CCA property is *anonymity and indistinguishability under chosen-ciphertext attack*, which we present in Figure 8.15. The property is a combination of anonymity (ANO-CCA) and indistinguishability (IND-CCA), which we have already defined in Figure 2.4, but the authors chose to use this combined property (which is equivalent) since it simplifies the proof.

**Issues with the PAPKE-IC proof in [Bra+19a].** Because our PAPKE-HIC construction (Section 8.3) builds upon PAPKE-IC, we have attempted to adapt the PAPKE-IC proof to PAPKE-HIC. However, we have discovered that there are several issues with the original PAPKE-IC proof [Bra+19a, Theorem 3].

The main issue concerns the treatment of the event $\mathsf{bad}_5$ in $\mathsf{Game}_5$ (using numbering from the original proof). The original proof claims that this event cannot occur because there always exists a row in $\mathbf{L}$ for the honest $apk$ such that $sk \neq \perp$. However, under the simulator's decryption procedure in $\mathsf{Game}_5$, the corresponding row in $\mathbf{L}$ is generated via a call to $\mathsf{IC.Enc}(pw, pk)$ and therefore contains $sk = \perp$. One could alter the simulation of the key generation process to avoid

$\underline{\mathsf{Exp}^{\text{AI-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda)}$

$b \leftarrow\!\!\$\ \{0,1\}$
$(sk_0, pk_0) \leftarrow \mathsf{KGen}(\lambda)$
$(sk_1, pk_1) \leftarrow \mathsf{KGen}(\lambda)$
$n \leftarrow 0$
$c^* \leftarrow \emptyset$
$b' \leftarrow \mathcal{A}^{\mathsf{LoR}_b(\cdot,\cdot),\mathsf{Dec}'(\cdot,\cdot)}(\lambda)$
**return** $b = b'$

$\underline{\mathsf{LoR}_b(m_0, m_1)}$

**if** $n > 0 \vee |m_0| \neq |m_1|$ **then**
  **abort**
$c \leftarrow \mathsf{PKE.Enc}(pk_b, m_b)$
$c^* \leftarrow c$
$n \leftarrow 1$
**return** $c$

$\underline{\mathsf{Dec}'(d, c)}$

**if** $c = c^*$ **then**
  **abort**
$m \leftarrow \mathsf{PKE.Dec}(sk_d, c)$
**return** $m$

**Figure 8.15:** AI-CCA security experiment for a public-key encryption scheme PKE = (KGen, Enc, Dec).

this issue, for example, by sampling $apk$, querying $\mathsf{IC.Dec}(pw, apk)$, and retrieving $(pk, sk)$ from $\mathbf{L}$ to ensure $sk \neq \perp$. While this would prevent $\mathsf{bad}_5$ in $\mathsf{Game}_5$, it does not extend to later games such as $\mathsf{Game}_8$, where the generated $(pk, sk)$ pair becomes independent of $apk$, and the simulator cannot rely on knowing the password. In such settings, the environment could potentially craft a valid ciphertext without knowing the public key, which reintroduces $\mathsf{bad}_5$.

This motivates our decryption robustness property (Section 8.1), which formalizes the requirement that an adversary cannot forge a valid ciphertext under a public key it does not know, even with access to a decryption oracle.

We also observed that the "badkeys" abort (3rd line in the key generation of $\mathcal{F}_{\mathsf{PAPKE}}$ in Figure 8.10) is not mentioned in the original proof. Although this omission is minor, it is relevant because the event occurs with negligible probability, assuming a large $apk$ key space.

We fix these issues in our PAPKE-HIC proof presented in Section 8.3; we also generalize the proof to consider non-perfectly correct public-key encryption schemes with non-perfectly uniform public keys (UNI-PK). This is important for post-quantum PAPKE-HIC, because the majority of post-quantum PKE schemes are neither perfectly correct nor have perfectly uniform public keys. Finally, our proof of PAPKE-HIC can be adapted into a PAPKE-IC proof by modifying our $\mathsf{Game}_2$ to accommodate for an IC instead of a HIC.

#### 8.2.4.2 PAPKE-FO

The PAPKE-FO construction is built by applying the Fujisaki-Okamoto (FO) transformation [FO13] to the ElGamal PKE [Elg85].

Compared to the generic PAPKE-IC, this construction exploits Diffie-Hellman group properties, making it difficult to adapt to other public-key encryption schemes—especially post-quantum PKEs, since there is no *efficient* post-quantum Diffie-Hellman analog.[5] On the other hand, in the classical setting, this construction does not require an ideal cipher over the public key space.

We do not present this construction in detail since we do not work with this construction further.

[5] There exist post-quantum Diffie-Hellman analogues, such as CSIDH [Cas+18; Cam+24] or Swoosh [Gaj+23], but they are not very efficient.

PAPKE-HIC.KGen($\lambda$, $pw$)

$(sk, pk) \leftarrow$ PKE.KGen($\lambda$)

$apk \leftarrow$ HIC.Enc($pw, pk$)

**return** $(sk, apk)$

PAPKE-HIC.Enc($apk$, $pw$, $m$)

$pk \leftarrow$ HIC.Dec($pw, apk$)

$c \leftarrow$ PKE.Enc($pk, m$)

**return** $c$

PAPKE-HIC.Dec($sk, c$)

$m \leftarrow$ PKE.Dec($sk, c$)

**return** $m$

**Figure 8.16:** The PAPKE-HIC construction where PKE is a public-key encryption scheme and HIC is a half-ideal cipher over the public key space of PKE.

## 8.3 PAPKE-HIC: Post-Quantum PAPKE

In this section, we present a novel PAPKE construction named PAPKE-HIC. As the name suggests, PAPKE-HIC is based on the PAPKE-IC construction (Section 8.2.4.1) with the difference that it uses a half-ideal cipher (Section 8.2.3.3) instead of an ideal cipher (Section 8.2.3.2). See Figure 8.16 for the exact description of the PAPKE-HIC construction, PAPKE-HIC, which combines a public-key encryption scheme PKE with a half-ideal cipher HIC.

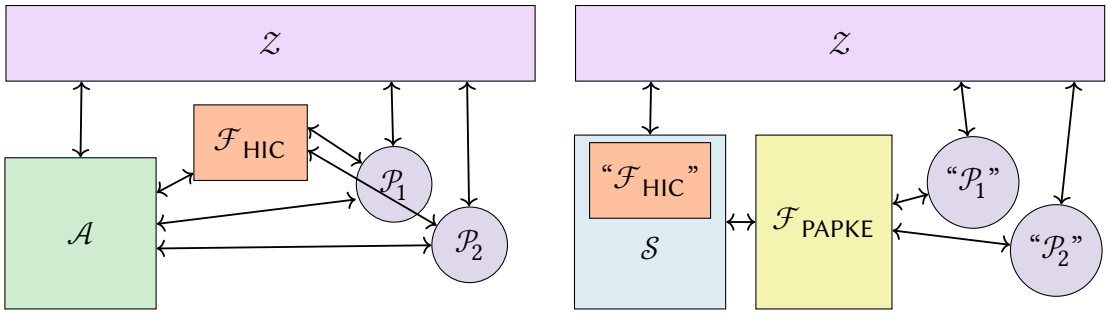
Recall from Section 8.2.3.3 that a HIC is superior to an IC since its instantiation M2F over the public key space is simpler. This also leads to better performance as shown in Section 8.4. Constructing a post-quantum PAPKE using the PAPKE-IC construction (Section 8.2.4.1) requires a post-quantum PKE, and such schemes typically involve large public keys. As a result, instantiating a post-quantum PAPKE-IC in practice is challenging due to its reliance on an IC, making the HIC-based approach a more practical alternative.

Recall that the HIC construction M2F uses an ideal cipher internally, but this cipher operates over bitstrings of length 256 (assuming 128-bit security), which can be instantiated using a regular block cipher. Even though some post-quantum KEMs, such as Classic McEliece [Alb+22], have public keys that are indistinguishable from random bitstrings, and there exist methods for encoding public keys as random bitstrings (e.g., Kemeleon for ML-KEM [GSV24]), the main challenge remains their size. For instance, the size of a Kemeleon-encoded ML-KEM public key is 781 *bytes*, and for Classic McEliece it is 261120 *bytes* at NIST security level 1. Recall from Section 8.2.3.2 that there are techniques to instantiate an ideal cipher over arbitrary bitstrings [Cor+10; DS16]; however, these are practically inefficient for large domains.

While M2F requires a *hash-to-group* function, hashing into the group, which in this case is the public key space, is straightforward for most post-quantum KEMs because they internally derive their public keys from a seed. This derivation mechanism can be used to implement the hash-to-group function.

In Theorem 9, we prove that PAPKE-HIC (Figure 8.16) UC-realizes the PAPKE functionality $\mathcal{F}_{\mathsf{PAPKE}}$ (Figure 8.10) by leveraging the HIC ideal functionality (Figure 8.12) and several security properties of the underlying PKE scheme.

**Theorem 9.** *The scheme* PAPKE-HIC *(Figure 8.16) UC-realizes* $\mathcal{F}_{\mathsf{PAPKE}}$ *(Figure 8.10) in the* $\mathcal{F}_{\mathsf{HIC}}$*-hybrid model (Figure 8.12), if the public-key encryption scheme* PKE *is* AI-CCA, SROB-CCA, UNI-PK, DROB-CCA *secure and* $\delta$*-correct. Specifically, there exists a PPT simulator* $\mathcal{S}$*, such that for all PPT environments* $\mathcal{Z}$ *controlling a dummy real-world adversary* $\mathcal{A}$*, its distinguishing advantage,* Adv,




(a) Real World (b) Ideal World

**Figure 8.17:** The $\mathcal{F}_{\mathsf{HIC}}$-hybrid model used in the PAPKE-HIC proof (Theorem 9).

*between the real and the ideal world is*

$$\begin{aligned}\mathsf{Adv}(\lambda) \leq\ & \frac{q_{\mathsf{HIC}}^2}{2\cdot|\mathcal{R}|} + \frac{q_{\mathsf{HIC}}^2}{2\cdot|\mathcal{PK}|} + \frac{q_{\mathsf{PAPKE}}}{|\mathcal{R}\times\mathcal{PK}|} + q_{\mathsf{PAPKE}}\cdot\delta(\lambda) \\ & + 2\cdot q_{\mathsf{HIC}}\cdot(q_{\mathsf{HIC}}+1)\cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) \\ & + 2\cdot(q_{\mathsf{HIC}}+1)\cdot q_{\mathsf{PAPKE,Enc}}\cdot \mathsf{Adv}^{\text{AI-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) \\ & + q_{\mathsf{HIC}}\cdot \mathsf{Adv}^{\text{UNI-PK}}_{\mathcal{B},\mathsf{PKE}}(\lambda) + \mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).\end{aligned}$$

*Here $q_{\mathsf{HIC}}$ and $q_{\mathsf{PAPKE}}$ represent upper bounds on the number of queries to $\mathcal{F}_{\mathsf{HIC}}$ and $\mathcal{F}_{\mathsf{PAPKE}}$, respectively. $\mathcal{R}$ is the* HIC *randomness space and $\mathcal{PK}$ is the public key space of* PKE.

### 8.3.1 Proof of Theorem 9

We provide a proof of Theorem 9, demonstrating that the PAPKE-HIC scheme from Figure 8.16 UC-realizes the $\mathcal{F}_{\mathsf{PAPKE}}$ functionality (Figure 8.10) in the $\mathcal{F}_{\mathsf{HIC}}$-hybrid model (Section 8.2.3.3), given that the underlying public-key encryption scheme PKE is AI-CCA, SROB-CCA, UNI-PK, DROB-CCA secure and $\delta$-correct. We assume that the half-ideal cipher (HIC) is accessible to all parties via the ideal functionality $\mathcal{F}_{\mathsf{HIC}}$ from Figure 8.12. See Figure 8.17 for visual description of the $\mathcal{F}_{\mathsf{HIC}}$-hybrid model.

Recall from Section 8.2.1 that we must show that for any efficient environment $\mathcal{Z}$ and a dummy real-world adversary $\mathcal{A}$, there exists an efficient ideal-world adversary, referred to as the simulator $\mathcal{S}$, such that the view of $\mathcal{Z}$ during its interaction with the real-world scheme and the real-world adversary $\mathcal{A}$ is indistinguishable from its view when interacting with the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$ and the simulator $\mathcal{S}$.

Note that the proof is in the $\mathcal{F}_{\mathsf{HIC}}$-hybrid model. Therefore, we assume that in the real world the $\mathcal{F}_{\mathsf{HIC}}$ ideal functionality is present, which can be queried by the protocol parties and by the adversary, whereas in the ideal-world, the $\mathcal{F}_{\mathsf{HIC}}$ ideal functionality is entirely emulated by the simulator (indicated as "$\mathcal{F}_{\mathsf{HIC}}$" in Figure 8.17).

We first describe the behavior of the ideal-world simulator $\mathcal{S}$, and then proceed to prove that the real-world interaction involving $\mathcal{A}$ and the PAPKE-HIC scheme is indistinguishable from the ideal-world interaction involving $\mathcal{F}_{\mathsf{PAPKE}}$ and $\mathcal{S}$.

**Simulator** $\mathcal{S}$. The simulator $\mathcal{S}$ acts as the ideal-world adversary, having access to the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$. For our proof to work, $\mathcal{S}$ must simulate network messages between honest and corrupted parties for $\mathcal{Z}$ without knowledge of the passwords chosen by $\mathcal{Z}$, while being consistent in its answers to HIC queries. In cases in which the simulator is not able to simulate consistently, we abort—in the proof we show that these events, referred to as "bad," occur with only negligible probability. These probabilities together are accounted for in the overall probability of $\mathcal{Z}$ distinguishing between the real and the ideal worlds. The description of the ideal-world simulator is depicted in Figures 8.23 and 8.24. Specifically, Figure 8.23 describes how the simulator responds to HIC queries, and Figure 8.24 describes how the simulator responds to queries from $\mathcal{F}_{\mathsf{PAPKE}}$.

*Proof.* We prove Theorem 9 via a series of game hops starting with $\mathsf{Game}_0$, which corresponds to the real world, and ending with $\mathsf{Game}_{13}$, which corresponds to the ideal world, where the ideal-world simulator $\mathcal{S}$ (Figures 8.23 and 8.24) interacts with the environment $\mathcal{Z}$ and $\mathcal{F}_{\mathsf{PAPKE}}$.

We follow the proof strategy visualized in Figure 8.9. At the beginning, in $\mathsf{Game}_0$, there is an unrestricted simulator which follows the real-world scheme from Figure 8.16 and honestly simulates HIC. Throughout the proof, we modify the simulator in each game and explain why such modification is indistinguishable for the environment. At the end, in $\mathsf{Game}_{13}$, we explain how the simulator at that point can be decomposed into the functionality $\mathcal{F}_{\mathsf{PAPKE}}$ (which knows the password) and the ideal-world simulator (which does not know the password) as described in Figures 8.23 and 8.24. The decomposition is visualized in Figures 8.21 and 8.22.

The modifications to the simulator in each game are detailed in Figures 8.18 to 8.20. In the figures, lines in gray boxes denote modifications based on the "simulator version." Specifically, when a line is annotated with $\mathsf{G}_i/\mathsf{G}_{\geq i}$, it denotes that the line is present in the simulator(s) for $\mathsf{Game}_i/(\mathsf{Game}_i, \mathsf{Game}_{i+1}, \dots)$. Blue text denotes the "lookup" indices in the instruction, e.g., $(a, b, c) \in \mathsf{L}$ means "search for an entry in $\mathsf{L}$ where the third entry is equal to $c$ and load the other entries into variables $a$ and $b$."

Denote by $\mathsf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}HIC}}(\lambda)$ the event that the environment $\mathcal{Z}$ outputs 1 in the $\mathcal{F}_{\mathsf{HIC}}$-hybrid model with the adversary $\mathcal{A}$ and the scheme PAPKE-HIC. Denote by $\mathsf{Ideal}_{\mathcal{Z},\mathcal{S},\mathcal{F}_{\mathsf{PAPKE}}}(\lambda)$ the event that $\mathcal{Z}$ outputs 1 when interacting in the ideal world with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{PAPKE}}$. $\mathsf{G}_i$ is the event where $\mathcal{Z}$ outputs 1 when interacting with $\mathsf{Game}_i$. Our goal is to show that

$$\left|\Pr\left[\mathsf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}HIC}}(\lambda)\right] - \Pr\left[\mathsf{Ideal}_{\mathcal{Z},\mathcal{S},\mathcal{F}_{\mathsf{PAPKE}}}(\lambda)\right]\right| \leq \mathsf{negl}(\lambda).$$

$\mathsf{Game}_0$ **(Real world):** This game corresponds to the real world. The simulator in this game runs the PAPKE-HIC (Figure 8.16) scheme honestly emulating the real-world parties. The simulator implements the half-ideal cipher (HIC.Enc, HIC.Dec, HIC.AdvEnc, HIC.AdvDec) via lazy sampling, while ensuring that $\forall(sid, pw)$ : $\mathsf{HIC.Enc}(sid, pw, \cdot)$, $\mathsf{HIC.Dec}(sid, pw, \cdot)$, $\mathsf{HIC.AdvEnc}(sid, pw, \cdot, \cdot)$, and $\mathsf{HIC.AdvDec}(sid, pw, \cdot)$ are permutations. See Figure 8.18 for detailed description (while ignoring gray lines annotated by $\mathsf{G}_i$ where $i > 1$). Note that this simulation corresponds to the ideal functionality $\mathcal{F}_{\mathsf{HIC}}$ from Figure 8.12.

Without loss of generality, we assume that the environment always queries $\mathsf{HIC.AdvDec}(sid, pw', apk')$ before making a query $(\mathsf{ENCRYPT}, sid, apk', pw', m)$.

Additionally, the simulator keeps a record of the PKE private key $sk$, the authenticated public key $apk$ corresponding to $sk$, and the password $pw$. See Figures 8.19 and 8.20 for detailed description of the simulator in this game (ignoring gray lines annotated by $\mathsf{G}_i$ where $i > 1$).

The simulator mimics the real-world scheme and HIC identically while keeping additional internal records, so the view of the environment in this game is identical to the real world.

$$\Pr[\mathrm{G}_0] = \Pr\left[\mathrm{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}HIC}}(\lambda)\right].$$

$\mathrm{Game}_1$ **(PKE correctness):** In this game, we assume that during this game no public-key encryption correctness errors occur. Note that a correctness error may be detected during ENCRYPT or DECRYPT queries; therefore,

$$|\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_0]| \leq q_{\mathsf{PAPKE}} \cdot \delta(\lambda).$$

$\mathrm{Game}_2$ **(Abort on HIC collisions):** In this game, the simulator aborts in cases where:

1) The sampled ciphertext $apk = (s, T) \in \mathcal{R} \times \mathcal{PK}$ during the invocation of HIC.Enc($sid$, $pw$, $\cdot$) or HIC.AdvEnc($sid$, $pw$, $\cdot$, $T$) has already been sampled during a previous invocation with a different password $pw' \neq pw$. We denote these events as $\mathrm{bad}_1$ (during Enc) and $\mathrm{bad}_2$ (during AdvEnc).

2) The sampled public key $pk \in \mathcal{PK}$ during the invocation of HIC.Dec/HIC.AdvDec($sid$, $pw$, $\cdot$) has already been sampled during a previous invocation with a different password $pw' \neq pw$. Note that the simulator also aborts in the case that the sampled $pk$ was previously input to HIC.Enc/HIC.AdvEnc as those calls also create a record in $\mathsf{List}_{\mathsf{HIC}}$. We denote this event as $\mathrm{bad}_3$.

Recall that $\mathcal{D} = \mathcal{R} \times \mathcal{PK}$, then

$$\begin{aligned}
|\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1]| &\leq \Pr[\mathrm{bad}_1] + \Pr[\mathrm{bad}_2] + \Pr[\mathrm{bad}_3], \text{where} \\
\Pr[\mathrm{bad}_1] &\leq \frac{q^2_{\mathsf{HIC,Enc}}}{2 \cdot |\mathcal{D}|} \leq \frac{q^2_{\mathsf{HIC,Enc}}}{2 \cdot |\mathcal{R}|}, \\
\Pr[\mathrm{bad}_2] &\leq \frac{q^2_{\mathsf{HIC,AdvEnc}}}{2 \cdot |\mathcal{R}|}, \\
\Pr[\mathrm{bad}_3] &\leq \frac{q^2_{\mathsf{HIC,Dec+AdvDec}}}{2 \cdot |\mathcal{PK}|}, \\
&\Longrightarrow \\
|\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1]| &\leq \frac{q^2_{\mathsf{HIC}}}{2 \cdot |\mathcal{R}|} + \frac{q^2_{\mathsf{HIC}}}{2 \cdot |\mathcal{PK}|}.
\end{aligned}$$

Note that this modification means that no double password guess is possible using a single $apk$. The fact that HIC (with a fixed password) is a permutation already prevents collisions for the same password on $apk$s (when HIC.Enc/HIC.AdvEnc is invoked) and on $(r, pk)$s (when HIC.Dec/HIC.AdvDec is invoked).

$\mathsf{Game}_3$ **(Removal of lazy sampling while loop during decryption):** Internally, the HIC.AdvDec simulation contains a while loop which resamples a public key if it has already been sampled for a fixed $(pw, r)$. This ensures that HIC with a fixed password is a permutation which corresponds to the ideal functionality $\mathcal{F}_{\mathsf{HIC}}$.

With the introduction of the abort $\mathsf{bad}_3$ in $\mathsf{Game}_2$, this loop is redundant as the simulator aborts in every case that would trigger another loop. Consequently, from the environment's point of view, these games are identical:

$$\Pr[\mathrm{G}_3] = \Pr[\mathrm{G}_2].$$

$\mathsf{Game}_4$ **(Sampling private keys during HIC decryption):** In this game, the simulator now samples and records keypairs, instead of only public keys, when unknown ciphertexts are submitted to HIC.Dec/HIC.AdvDec. This game is indistinguishable from the previous one due to the pseudo-randomness of public keys (UNI-PK) of PKE using a hybrid argument over all HIC decryption queries, i.e.,

$$|\Pr[\mathrm{G}_4] - \Pr[\mathrm{G}_3]| \leq q_{\mathsf{HIC,Dec+AdvDec}} \cdot \mathsf{Adv}^{\text{UNI-PK}}_{\mathcal{B},\mathsf{PKE}}(\lambda) \leq q_{\mathsf{HIC}} \cdot \mathsf{Adv}^{\text{UNI-PK}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

$\mathsf{Game}_5$ **(Record keeping for honestly generated ciphertexts):** In this game, the simulator starts keeping records of honestly generated PAPKE ciphertexts.

Specifically, upon receiving $(\mathsf{ENCRYPT}, sid, apk', pw', m)$, the simulator verifies if $apk'$ belongs to an honest party ($apk = apk'$) and if so, it creates a record $(sid, m^*, c)$ in $\mathsf{List}_{\mathsf{encrec}}$ where $m^* \leftarrow m$ if $pw = pw'$, and $m^* \leftarrow \perp$ otherwise.

Note that we consider any List to be realized as a set, i.e., there cannot exist duplicate records. Without loss of generality, when the simulator adds a record $(sid, m, c)$, and there already exists an identical record, nothing happens.

This is an internal change to the simulator. Therefore,

$$\Pr[\mathrm{G}_5] = \Pr[\mathrm{G}_4].$$

$\mathsf{Game}_6$ **(Aborts on double records):** In this game, the simulator aborts if it attempts to create a record $(sid, m^*, c)$ in $\mathsf{List}_{\mathsf{encrec}}$, but there already exists a record $(sid, m', c)$ with the same $c$ (and $sid$), but $m^* \neq m'$.[6] In other words, from now on, for any $(sid, c)$ pair there exists at most one record in $\mathsf{List}_{\mathsf{encrec}}$.

To argue indistinguishability from the previous game, we must prove that the probability that the simulator aborts is negligible. Let $(sid, m^*, c)$ be the new "to-be-added" record and $(sid, m', c)$ be the existing record. We distinguish between two events, $\mathsf{bad}_4$ and $\mathsf{bad}_5$, that cause the simulator to abort:

$\mathsf{bad}_4$: $m^* \neq \perp \ \wedge m' \neq \perp \ \wedge m^* \neq m'$: This event contradicts the correctness of PKE since it implies that $c$ would be the result of two queries $(\mathsf{ENCRYPT}, sid, apk, pw, m')$ and $(\mathsf{ENCRYPT}, sid, apk, pw, m^*)$, i.e.,

[6] If $m^* = m'$, no identical duplicate records can exist since List is realized as a set. See discussion in Game 5.

$$pk \leftarrow \mathsf{HIC.Dec}(pw, apk)$$
$$\mathsf{PKE.Enc}(pk, m') \rightarrow c \leftarrow \mathsf{PKE.Enc}(pk, m^*).$$

Therefore, $\Pr[\mathrm{bad}_4] = 0$ since PKE is perfectly correct since $\mathrm{Game}_1$.

$\mathrm{bad}_5$: Without loss of generality $m^* = \perp \wedge m' \neq \perp$: This implies that $c$ would be the result of two queries $(\mathsf{ENCRYPT}, sid, apk, pw, m')$ and $(\mathsf{ENCRYPT}, sid, apk, pw', m)$ where $pw' \neq pw$, i.e.,

$$pk \leftarrow \mathsf{HIC.Dec}(pw, apk)$$
$$c \leftarrow \mathsf{PKE.Enc}(pk, m')$$
$$\text{and}$$
$$pk' \leftarrow \mathsf{HIC.Dec}(pw', apk)$$
$$c \leftarrow \mathsf{PKE.Enc}(pk', m).$$

Note that we can assume $pk \neq pk'$ since $\mathrm{Game}_2$. Therefore, an environment that causes this event can be used to build an adversary $\mathcal{B}$ against strong robustness of PKE. $\mathcal{B}$ acts as follows: it receives $pk_0, pk_1$ from the SROB-CCA challenger and runs $\mathcal{Z}$ and $\mathcal{A}$ in a modified $\mathrm{Game}_5$. The simulator in the game is modified as follows:

- Upon receiving $(\mathsf{KEYGEN}, sid, pwd)$, the simulator does not run $\mathsf{PKE.KGen}(\lambda)$ to get $(sk, pk)$, but it encrypts $pk_0$, i.e., $apk \leftarrow \mathsf{HIC.Enc}(pwd, pk_0)$. In cases where the simulator would invoke $\mathsf{PKE.Dec}(sk, \cdot)$ (responding to $(\mathsf{DECRYPT}, sk, c)$ queries) the simulator uses $\mathrm{Dec}(0, \cdot)$ oracle to decrypt.
- It randomly samples $i \leftarrow\!\!\$ \{1, \ldots, q_{\mathsf{HIC,Dec}}\}$ and responds to the $i$-th HIC.Dec query with $pk_1$.

If the event $\mathrm{bad}_5$ occurs during the modified game, there is a $q^{-1}_{\mathsf{HIC,Dec}}$ chance that the "colliding" $c$ is the result of an encryption using $pk_1$. In that case, $\mathcal{B}$ wins the SROB-CCA game with such $c$ since it must decrypt under the secret keys corresponding to $pk_0$ and $pk_1$ because of PKE correctness. In other words

$$\Pr[\mathrm{bad}_5] \leq q_{\mathsf{HIC,Dec}} \cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) \leq q_{\mathsf{HIC}} \cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

In total, the difference between this game and the previous game is that the simulator aborts when $\mathrm{bad}_4$ or $\mathrm{bad}_5$ occurs. Therefore,

$$|\Pr[\mathrm{G}_6] - \Pr[\mathrm{G}_5]| \leq \Pr[\mathrm{bad}_4] + \Pr[\mathrm{bad}_5] \leq q_{\mathsf{HIC}} \cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

$\mathsf{Game}_7$ **(Decryption of known ciphertext using records):** In this game, we modify how the simulator responds to decryption queries for ciphertexts that were created honestly. Specifically, upon receiving $(\mathsf{DECRYPT}, sid, sk, c)$ the simulator checks if there exists a record $(sid, m, c)$ in $\mathsf{List}_{\mathsf{encrec}}$. If such record exists, the simulator responds with that $m$. Note that since $\mathsf{Game}_6$ there can be at most one record with such $c$ (and $sid$). If no record is found, the simulator proceeds as in the previous games, i.e., responds with $\mathsf{PKE.Dec}(sk, c)$.

The only way the environment could distinguish between this game and the previous one is in case there exists a record $(sid, \perp, c)$, but $c$ still decrypts under $sk$, i.e., $\perp \neq m' \leftarrow \mathsf{PKE.Dec}(sk, c)$. We call this event $\mathsf{bad}_6$, and we show that an environment that causes $\mathsf{bad}_6$ to occur can be used to build an adversary $\mathcal{B}$ against strong robustness of PKE (SROB-CCA). The record $(sid, \perp, c)$ was created during the response to the query $(\mathsf{ENCRYPT}, sid, apk, pw', m)$ where $apk$ is honest, but $pw'$ is incorrect ($pw' \neq pw$), i.e., $c$ was created as follows:

$$pk' \leftarrow \mathsf{HIC.Dec}(pw', apk)$$
$$c \leftarrow \mathsf{PKE.Enc}(pk', m).$$

Since $\mathsf{Game}_2$, we know that $pk' \neq pk$ and $c$ decrypts under $sk'$, which was sampled during the HIC.Dec call, due to correctness of PKE, and it decrypts under $sk$ assuming $\mathsf{bad}_6$.

The reduction is identical as in the previous game. $\mathcal{B}$ acts as follows: it receives $pk_0, pk_1$ from the SROB-CCA challenger and runs $\mathcal{Z}$ and $\mathcal{A}$ in a modified $\mathsf{Game}_7$. The simulator in the game is modified as follows:

- Upon receiving $(\mathsf{KEYGEN}, sid, pwd)$, the simulator does not run $\mathsf{PKE.KGen}(\lambda)$ to get $(sk, pk)$, but it encrypts $pk_0$, i.e., $apk \leftarrow \mathsf{HIC.Enc}(pwd, pk_0)$. In cases where the simulator runs $\mathsf{PKE.Dec}(sk, \cdot)$ (responding to $(\mathsf{DECRYPT}, sk, c)$ queries) the simulator uses the $\mathsf{Dec}(0, \cdot)$ oracle to decrypt.

- It randomly samples $i \leftarrow\!\!\$\, \{1, \ldots, q_{\mathsf{HIC,Dec}}\}$ and responds to the $i$-th HIC.Dec query with $pk_1$.

If the event $\mathsf{bad}_6$ occurs during the modified game, there is a $q^{-1}_{\mathsf{HIC,Dec}}$ chance that the "colliding" $c$ is the result of an encryption using $pk_1$. In that case, $\mathcal{B}$ wins the SROB-CCA game with such $c$ since it decrypts under the secret keys corresponding to $pk_0$ and $pk_1$ because of correctness of PKE.

$$\Pr[\mathsf{bad}_6] = q_{\mathsf{HIC,Dec}} \cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) \leq q_{\mathsf{HIC}} \cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda),$$
$$|\Pr[\mathrm{G}_7] - \Pr[\mathrm{G}_6]| \leq \Pr[\mathsf{bad}_6] \leq q_{\mathsf{HIC}} \cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

$\mathsf{Game}_8$ **(Record keeping of decrypted adversarial ciphertexts):** In this game, the simulator keeps a record of how it decrypted an adversarial ciphertext. Specifically, upon receiving $(\mathsf{DECRYPT}, sid, sk, c)$, if there is no record for such $(sid, c)$ in $\mathsf{List}_{\mathsf{encrec}}$, the simulator decrypts $c$ with $sk$, i.e., it returns $m' \leftarrow \mathsf{PKE.Dec}(sk, c)$. This process so far is identical to the previous games. However, the simulator now creates a new record $(sid, m', c)$ to $\mathsf{List}_{\mathsf{encrec}}$ after it decrypts $c$.

Even though this change means that the next time $(\mathsf{DECRYPT}, sid, sk, c)$ is queried, the simulator will find a record for such $(sid, c)$ in $\mathsf{List}_{\mathsf{encrec}}$ and return $m'$ from that record, the decryption result is still consistent. In the previous game, the simulator instead would decrypt (again)

$m'' \leftarrow \mathsf{PKE.Dec}(sk, c)$, but $m''$ must be equal to $m'$ from the record since decryption is deterministic. In other words,

$$\Pr[\mathrm{G}_8] = \Pr[\mathrm{G}_7].$$

$\mathrm{Game}_9$ **(Dummy plaintexts for real** $apk$**):** In this game, we modify how the encryption procedure outputs dummy ciphertexts if the submitted $apk$ is honest. Specifically, upon receiving $(\mathsf{ENCRYPT}, sid, apk', pw', m)$ where $apk' = apk$, the simulator no longer decrypts $apk'$ with $pw'$, but it generates a random $(pk', \_) \leftarrow \mathsf{PKE.KGen}(\lambda)$, a random message $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$ and encrypts $m'$ using $pk'$ to produce $c \leftarrow \mathsf{PKE.Enc}(pk', m')$.

The change means that the environment now sees dummy ciphertexts as the results of the queries $(\mathsf{ENCRYPT}, sid, apk, pw', m)$ as opposed to "honest" encryptions of messages $m$ under $pk' \leftarrow \mathsf{HIC.Dec}(pw', apk)$. We show that this game is indistinguishable from the previous game due to AI-CCA of PKE using a hybrid argument over the number of ENCRYPT queries denoted as $q_{\mathsf{PAPKE,Enc}}$.

For each $i \in \{1, \ldots, q_{\mathsf{PAPKE,Enc}} + 1\}$, the $i$-th hybrid game is defined as follows:

- The simulator responds with dummy ciphertexts to the first $i - 1$ ENCRYPT queries with $apk$.
- The simulator responds with "honest" ciphertexts to the $i$-th to the $q_{\mathsf{PAPKE,Enc}}$-th ENCRYPT queries with $apk$.

The 1st hybrid corresponds to $\mathrm{Game}_8$ and the $(q_{\mathsf{PAPKE,Enc}}+1)$-th hybrid corresponds to $\mathrm{Game}_9$. An environment able to distinguish between the $i$-th and the $(i+1)$-th hybrid can be used to build an adversary $\mathcal{B}$ against the anonymity and indistinguishability of PKE (AI-CCA). $\mathcal{B}$ acts as follows: it receives $pk_0, pk_1$ from the AI-CCA challenger and runs $\mathcal{Z}$ and $\mathcal{A}$ in a modified $i$-th hybrid. $\mathcal{B}$ samples $j \leftarrow\!\!\$\ \{0, \ldots, q_{\mathsf{HIC}}\}$ and modifies the $i$-th hybrid based on the value of $j$:

- $j = 0$: $pk_0$ is used instead of the honest $pk$, i.e., $apk \leftarrow \mathsf{HIC.Enc}(pw, pk_0)$. The simulator uses the $\mathrm{Dec}'(0, \cdot)$ oracle from the AI-CCA game to emulate $\mathsf{PKE.Dec}(sk, \cdot)$. If in the $i$-th $(\mathsf{ENCRYPT}, sid, apk', pw', m)$ query $apk' \neq apk$ or $pw' \neq pw$, then $\mathcal{B}$ aborts; otherwise, the simulator samples dummy $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$, sends $(m, m')$ to the $\mathsf{LoR}(\cdot,\cdot)$ AI-CCA oracle and responds with $c$ that it received from the oracle.
- $j \neq 0$: If the $j$-th HIC query is not a decryption query with inputs $(pw^j, apk^j)$ where $apk^j = apk$ and $pw^j \neq pw$, then $\mathcal{B}$ aborts; otherwise, the simulator responds with $pk_0$ and records the input pair $(pw^j, apk^j)$. Let $(\mathsf{ENCRYPT}, sid, apk', pw', m)$ be the $i$-th ENCRYPT query. If $apk' \neq apk^j$ or $pw' \neq pw^j$, then $\mathcal{B}$ aborts; otherwise, the simulator samples dummy $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$, sends $(m, m')$ to the $\mathsf{LoR}(\cdot,\cdot)$ AI-CCA oracle and responds with $c$ that it received from the oracle.

If $\mathcal{Z}$ indicates that it was running in the $i$-th hybrid, $\mathcal{B}$ returns 0, otherwise it returns 1. Consequently,

$$\mathrm{Adv}^{\text{AI-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) \geq \frac{1}{q_{\mathsf{HIC}} + 1} \cdot |\Pr[\mathrm{H}_i] - \Pr[\mathrm{H}_{i+1}]|$$

where $\mathsf{H}_i$ is the event that $\mathcal{Z}$ indicates it was running in the $i$-th hybrid while running in the $i$-th hybrid and $\mathsf{H}_{i+1}$ is the event that $\mathcal{Z}$ indicates it was running in the $i$-th hybrid while running in the $i+1$-th hybrid. In total,

$$|\Pr[\mathsf{G}_9] - \Pr[\mathsf{G}_8]| \leq q_{\mathsf{PAPKE,Enc}} \cdot (q_{\mathsf{HIC}} + 1) \cdot \mathsf{Adv}^{\text{AI-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

$\mathsf{Game}_{10}$ **(Dummy plaintexts for dishonest $apk'$):** In this game, we additionally modify the encryption procedure to respond with a dummy ciphertext also in the case when $apk' \neq apk$.

However, we need to consider the case when $apk'$ is created by the environment from its own keypair $(pk', sk')$ by calling $apk' \leftarrow \mathsf{HIC.AdvEnc}(sid, pw^*, (r, pk'), T)$ for some $pw^* \in \mathcal{PW}, r \in \mathcal{R}$ and $T \in \mathcal{PK}$. In this case, the simulator needs to actually encrypt the message "honestly", if the password $pw'$ supplied in the ENCRYPT query matches the one used to create the supplied $apk'$, which is $pw^*$.

The only way the environment can create an $apk'$ corresponding to a keypair known to the environment is by calling HIC.AdvEnc. So during encryption, the simulator looks for a record in $(sid, pw^*, pk', apk', \bot) \in \mathsf{List}_{\mathsf{HIC}}$. If such a record is found, the simulator checks if $pw^* = pw'$, where $pw'$ is the password from the query $(\mathsf{ENCRYPT}, sid, pw', apk', m)$.[7] If the passwords match, the simulator encrypts the real message $m$ under $pk'$, otherwise, it creates a dummy ciphertext.

The simulator additionally keeps a list $(sid, apk', pw^*) \in \mathsf{List}_{\mathsf{badkeys}}$ of already looked up results from $\mathsf{List}_{\mathsf{HIC}}$ for each $apk$. This change does not affect the outcome since the row in $\mathsf{List}_{\mathsf{HIC}}$ cannot change once it exists.

Therefore, the change in this game is that the simulator additionally outputs dummy ciphertexts in cases when $apk' \neq apk$ and $pw' \neq pw^*$. We show that this game is indistinguishable from the previous game using a hybrid argument over the number of ENCRYPT queries denoted as $q_{\mathsf{PAPKE,Enc}}$. For each $i \in \{1, \ldots, q_{\mathsf{PAPKE,Enc}} + 1\}$, the $i$-th hybrid game is defined as follows:

- The simulator responds with dummy ciphertexts to the first $i-1$ ENCRYPT queries where $apk' \neq apk$ and $pw' \neq pw^*$.
- The simulator responds with "honest" ciphertexts to the $i$-th to the $q_{\mathsf{PAPKE,Enc}}$-th ENCRYPT queries where $apk' \neq apk$ and $pw' \neq pw^*$.

The 1st hybrid corresponds to $\mathsf{Game}_9$ and the $(q_{\mathsf{PAPKE,Enc}}+1)$-th hybrid corresponds to $\mathsf{Game}_{10}$. An environment that is able to distinguish between the $i$-th and the $i+1$-th hybrid can be used to build an adversary $\mathcal{B}$ against the anonymity and indistinguishability of PKE (AI-CCA). $\mathcal{B}$ acts as follows: it receives $pk_0, pk_1$ from the AI-CCA challenger and runs $\mathcal{Z}$ and $\mathcal{A}$ in a modified $i$-th hybrid. $\mathcal{B}$ samples $j \leftarrow\!\!\$\ \{1, \ldots, q_{\mathsf{HIC}}\}$. If the $j$-th HIC query is not a decryption query with input $(pw^j, apk^j)$ where $apk^j \neq apk$, then $\mathcal{B}$ aborts; otherwise, the simulator responds with $pk_0$ and records the input pair $(pw^j, apk^j)$. Let $(\mathsf{ENCRYPT}, sid, apk', pw', m)$ be the $i$-th ENCRYPT query. If $apk' \neq apk^j$ or $pw' \neq pw^j$, then $\mathcal{B}$ aborts; otherwise, the simulator samples dummy $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$, sends $(m, m')$ to the $\mathsf{LoR}(\cdot,\cdot)$ AI-CCA oracle and responds with $c$ that it received from the oracle.

If $\mathcal{Z}$ indicates that it was running in the $i$-th hybrid, $\mathcal{B}$ returns 0, otherwise it returns 1. Consequently,

$$|\Pr[\mathsf{G}_{10}] - \Pr[\mathsf{G}_9]| \leq q_{\mathsf{PAPKE,Enc}} \cdot q_{\mathsf{HIC}} \cdot \mathsf{Adv}^{\text{AI-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

[7] Note that there can be at most one such record in $\mathsf{List}_{\mathsf{HIC}}$ since $\mathsf{Game}_2$.

$\mathsf{Game}_{11}$ **(Dummy $apk$ and decryption of unknown ciphertexts):** In this game, the simulator during key generation samples a random $apk \leftarrow\!\!\$\ \mathcal{D}$ rather than encrypting the sampled public key using HIC.Enc. This change requires a modification of how unknown ciphertexts, i.e., those that are not in $\mathsf{List}_{\mathsf{encrec}}$, are decrypted because the simulator was responding with $\mathsf{PKE.Dec}(sk, c)$ until now, and now such $sk$ is independent of the $apk$. In this game, the simulator decrypts and extracts the password by iterating over all records in $\mathsf{List}_{\mathsf{HIC}}$ and testing if any $sk'$ from $\mathsf{List}_{\mathsf{HIC}}$ decrypts $c$.

Specifically, upon receiving $(\mathsf{DECRYPT}, sid, \bot, c)$[8] s.t. there is no record for such $(sid, c)$ in $\mathsf{List}_{\mathsf{encrec}}$, the simulator checks if there exists a record $(sid, pw', pk', apk, sk')$ in $\mathsf{List}_{\mathsf{HIC}}$ where $sk' \neq \bot$ and $\bot \neq m \leftarrow \mathsf{PKE.Dec}(sk', c)$.

- If there exists more than one such record, then the simulator aborts; we call this event $\mathsf{bad}_8$.
- If there exists only one record, then the simulator returns $(m^*, pw')$ where $m^* \leftarrow m$ if $pw' = pw$, and $m^* \leftarrow \bot$ otherwise.
- If there is no such record, then the simulator uses the "dummy" keypair $(pk, sk)$, that was sampled during KEYGEN along with $apk$, and tries to decrypt $m^* \leftarrow \mathsf{PKE.Dec}(sk, c)$. If $m^* \neq \bot$, then the simulator aborts; we call this event $\mathsf{bad}_9$. Otherwise, it returns $(\bot, \bot)$. Note that $(pk, sk)$ is independent of $apk$ in this game.

In summary, if $c$ was created using a $pk$ that was the result of the query $\mathsf{HIC.AdvDec}(sid, pw, apk)$ for an honest $pw$ and an honest $apk$, we decrypt $c$ to a message.

This game is indistinguishable from the previous game unless aborts occur. First, we argue that $\mathsf{bad}_8$ reduces to strong robustness of PKE (SROB-CCA) and then we show that $\mathsf{bad}_9$ reduces to decryption robustness of PKE (DROB-CCA).

If $\mathsf{bad}_8$ occurs, it means that there exist two records $(sid, pw_0, pk_0, apk, sk_0)$ and $(sid, pw_1, pk_1, apk, sk_1)$ in $\mathsf{List}_{\mathsf{HIC}}$ where $pw_0 \neq pw_1$ (otherwise HIC would return the existing row with such $(sid, pw, apk)$) and where $sk_0$ and $sk_1$ both decrypt $c$ to a value that is not $\bot$. An environment causing $\mathsf{bad}_8$ can be used to build an adversary $\mathcal{B}$ against strong robustness of PKE. $\mathcal{B}$ receives $pk_0, pk_1$ from the SROB-CCA challenger, samples two values $i_0, i_1 \leftarrow\!\!\$\ \{1, \ldots, q_{\mathsf{HIC,AdvDec}}\}$ s.t. $i_0 \neq i_1$ and runs $\mathcal{Z}$ and $\mathcal{A}$ in a modified $\mathsf{Game}_{11}$. The simulator in the game is modified as follows:

- For $b \in \{0, 1\}$: it responds to the $i_b$-th HIC decryption query with $pk_b$ and uses the SROB-CCA oracle $\mathsf{Dec}(b, \cdot)$ to calculate $\mathsf{PKE.Dec}(sk_b, c)$.[9]

If $\mathsf{bad}_8$ occurs during the modified game during decryption of $c$, there is a $\frac{2}{(q_{\mathsf{HIC,AdvDec}}-1)\cdot q_{\mathsf{HIC,AdvDec}}}$ probability that those two records correspond to $pk_0$ and $pk_1$. Therefore, if $\mathcal{B}$ uses that $c$, then it wins the strong robustness game. In other words,

$$\begin{aligned}\Pr[\mathsf{bad}_8] &\leq \frac{(q_{\mathsf{HIC,AdvDec}} - 1) \cdot q_{\mathsf{HIC,AdvDec}}}{2} \cdot \mathsf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) \\ &\leq \frac{(q_{\mathsf{HIC}} - 1) \cdot q_{\mathsf{HIC}}}{2} \cdot \mathsf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).\end{aligned}$$

[8] Technically, the query contains an $sk^{\dagger}$, but the simulator ignores it from the beginning and selects $sk$ based on $sid$.

[9] More precisely, the simulator remembers which records in $\mathsf{List}_{\mathsf{HIC}}$ correspond to $pk_b$ and acts as if there was $sk_b$ in the record and anytime that $sk_b$ would be needed (during decryption during iteration) it uses the oracle.

Now we show that the probability of $\mathsf{bad}_9$ is negligible if PKE satisfies decryption robustness (DROB-CCA). This event occurs if the submitted ciphertext $c$ in a DECRYPT query does not decrypt under any secret key from $\mathsf{List}_{\mathsf{HIC}}$, but it decrypts under the keypair $(pk, sk)$ (generated by the KEYGEN query). In other words, a valid ciphertext was produced without utilizing $apk$ (and the only way to use $apk$ is to try to decrypt since it is a HIC ciphertext), given only access to the $\mathsf{PKE.Dec}(sk, \cdot)$ oracle by querying DECRYPT.

We show that we can build an adversary $\mathcal{B}$ against DROB-CCA by running a modified $\mathsf{Game}_{11}$ where anytime $\mathsf{PKE.Dec}(sk, \cdot)$ is called, the adversary uses its decryption oracle from the DROB-CCA challenger. If $\mathsf{bad}_9$ occurs, $\mathcal{B}$ wins the DROB-CCA game. Clearly,

$$\Pr[\mathsf{bad}_9] \leq \mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda).$$

In total, we have

$$\begin{aligned} |\Pr[\mathsf{G}_{11}] - \Pr[\mathsf{G}_{10}]| &\leq \Pr[\mathsf{bad}_8] + \Pr[\mathsf{bad}_9] \\ &\leq \frac{(q_{\mathsf{HIC}} - 1) \cdot q_{\mathsf{HIC}}}{2} \cdot \mathsf{Adv}^{\text{SROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda) + \mathsf{Adv}^{\text{DROB-CCA}}_{\mathcal{B},\mathsf{PKE}}(\lambda). \end{aligned}$$

$\mathsf{Game}_{12}$ **(Sampling a bad $apk'$):** In this game, the simulator aborts if during KEYGEN the sampled $apk \leftarrow\!\!\$\ \mathcal{D}$ matches one of the $apk'$'s in $\mathsf{List}_{\mathsf{badkeys}}$. Denote this event by $\mathsf{bad}_{10}$. Clearly,

$$\begin{aligned} \Pr[\mathsf{bad}_{10}] &\leq \frac{q_{\mathsf{PAPKE}}}{|\mathcal{D}|} = \frac{q_{\mathsf{PAPKE}}}{|\mathcal{R} \times \mathcal{PK}|}, \\ |\Pr[\mathsf{G}_{12}] - \Pr[\mathsf{G}_{11}]| &\leq \frac{q_{\mathsf{PAPKE}}}{|\mathcal{R} \times \mathcal{PK}|}. \end{aligned}$$

$\mathsf{Game}_{13}$ **(Ideal world):** The pre-ideal-world "super-simulator" from $\mathsf{Game}_{12}$ can be decomposed into the ideal-world simulator (Figures 8.23 and 8.24) and $\mathcal{F}_{\mathsf{PAPKE}}$ (Figure 8.10). The decomposition is visualized in Figures 8.21 and 8.22 which depict the $\mathsf{Game}_{12}$ simulator with gray boxes highlighting parts handled by the ideal-world simulator.

Note that the final game simulator is technically not exactly the same because in $\mathsf{Game}_{12}$ we still consider that the simulator shares memory with $\mathcal{F}_{\mathsf{PAPKE}}$, but in the ideal world, the simulator needs to keep track of things on its own. Specifically, the ideal-world simulator (Figure 8.24) keeps a list $\mathsf{List}_{\mathsf{simrec}}$ which is essentially its equivalent of $\mathcal{F}_{\mathsf{PAPKE}}$'s $\mathsf{List}_{\mathsf{keyrec}}$ except the password.

In other words,

$$\Pr\left[\mathsf{Ideal}_{\mathcal{Z},\mathcal{S},\mathcal{F}_{\mathsf{PAPKE}}}(\lambda)\right] = \Pr[\mathsf{G}_{12}].$$

By summing up all the bad events we get the bound from Theorem 9. □

## HIC part of the simulator

### *Interface for honest parties:.*

On query HIC.Enc($sid$, $pw$, $pk$) from party $\mathcal{P}$ where $pk \in \mathcal{PK}$:
- $r \leftarrow\$ \mathcal{R}$
- if $\exists(sid, pw, (r, pk), apk, *) \in \text{List}_{\text{HIC}}$ then:
  - return $apk$ to $\mathcal{P}$
- else:
  - do:
    - $apk \leftarrow\$ \mathcal{D}$
  - while: $(sid, pw, *, apk, *) \in \text{List}_{\text{HIC}}$
  - if $\exists(sid, *, *, apk, \bot) \in \text{List}_{\text{HIC}}$ then: abort $\text{bad}_1$ — $G_{\geq 2}$
  - add $(sid, pw, (r, pk), apk, \bot)$ to $\text{List}_{\text{HIC}}$
  - return $apk$ to $\mathcal{P}$

On query HIC.Dec($sid$, $pw$, $apk$) from party $\mathcal{P}$ where $apk \in \mathcal{D}$:
- $(r, pk) \leftarrow$ HIC.AdvDec($sid$, $pw$, $apk$)
- return $pk$ to $\mathcal{P}$

### *Interface for the ideal-world adversary and corrupt parties:.*

On query HIC.AdvEnc($sid$, $pw$, $(r, pk)$, $T$) from adversary $\mathcal{A}$ where $(r, pk) \in \mathcal{D}, T \in \mathcal{PK}$:
- if $\exists(sid, pw, (r, pk), apk, *) \in \text{List}_{\text{HIC}}$:
  - return $apk$ to $\mathcal{A}$
- else:
  - do:
    - $s \leftarrow\$ \mathcal{R}$
  - while: $(sid, pw, *, (s, T), *) \in \text{List}_{\text{HIC}}$
  - if $\exists(sid, *, *, (s, T), \bot) \in \text{List}_{\text{HIC}}$ then: abort $\text{bad}_2$ — $G_{\geq 2}$
  - $apk \leftarrow (s, T)$
  - add $(sid, pw, (r, pk), apk, \bot)$ to $\text{List}_{\text{HIC}}$
  - return $apk$ to $\mathcal{A}$

On query HIC.AdvDec($sid$, $pw$, $apk$) from adversary $\mathcal{A}$ where $apk \in \mathcal{D}$:
- if $\exists(sid, pw, (r, pk), apk, *) \in \text{List}_{\text{HIC}}$:
  - return $(r, pk)$ to $\mathcal{A}$
- else:
  - $r \leftarrow\$ \mathcal{R}$
  - do: — $G_{\leq 2}$
    - $pk \leftarrow\$ \mathcal{PK}$ — $G_{\leq 3}$
    - $(pk, sk) \leftarrow \text{PKE.KGen}(\lambda)$ — $G_{\geq 4}$
    - if $\exists(sid, *, (*, pk), *, *) \in \text{List}_{\text{HIC}}$ then: abort $\text{bad}_3$ — $G_{\geq 2}$
  - while $(sid, pw, (r, pk), *, *) \in \text{List}_{\text{HIC}}$ — $G_{\leq 2}$
  - add $(sid, pw, (r, pk), apk, \bot)$ to $\text{List}_{\text{HIC}}$ — $G_{\leq 3}$
  - add $(sid, pw, (r, pk), apk, sk)$ to $\text{List}_{\text{HIC}}$ — $G_{\geq 4}$
  - return $(r, pk)$ to $\mathcal{A}$

**Figure 8.18:** HIC part of the simulator. This description captures the changes made during game hops. For the ideal-world description see Figure 8.23.

## PAPKE-HIC part of the simulator, part 1

On query (KEYGEN, $sid$, $pw$) from $\mathcal{P}$:
  if $sid \neq (\mathcal{P}, sid') \vee \exists(sid, *, *, *) \in \text{List}_{\text{keyrec}}$:
    abort
  $(pk, sk) \leftarrow \text{PKE.KGen}(\lambda)$
  $apk \leftarrow \text{HIC.Enc}(sid, pw, pk)$ — $G_{\leq 10}$
  $apk \leftarrow\!\!\$\ \mathcal{D}$ — $G_{\geq 11}$
  if $\exists(sid, apk, *) \in \text{List}_{\text{badkeys}}$: — $G_{\geq 12}$
    abort $\text{bad}_{10}$
  add $(sid, sk, apk, pw)$ to $\text{List}_{\text{keyrec}}$
  return (KEYCONF, $sid, apk$) to $\mathcal{P}$

On query (ENCRYPT, $sid, apk', pw', m$) from $\mathcal{Q}$:

  $pk \leftarrow \text{HIC.Dec}(sid, pw', apk')$
  $c \leftarrow \text{PKE.Enc}(pk, m)$ — $G_{\leq 8}$
  lookup $(sid, *, apk, pw) \in \text{List}_{\text{keyrec}}$ — $G_{\geq 5}$
  if $apk = apk'$:
    $(pk', \_) \leftarrow \text{PKE.KGen}(\lambda)$
    $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$
    $c \leftarrow \text{PKE.Enc}(pk', m')$ — $G_{\geq 9}$
    if $pw = pw'$:
      $m^* \leftarrow m$
    else:
      $m^* \leftarrow\!\perp$
    if $\exists(sid, *, c) \in \text{List}_{\text{encrec}}$ then: — $G_{\geq 6}$
      abort $\text{bad}_4 + \text{bad}_5$
    add $(sid, m^*, c)$ to $\text{List}_{\text{encrec}}$
  else:
    $pk \leftarrow \text{HIC.Dec}(sid, pw', apk')$
    $c \leftarrow \text{PKE.Enc}(pk, m)$ — $G_9$
    if $\exists(sid, apk', pw^\dagger) \in \text{List}_{\text{badkeys}}$: — $G_{\geq 10}$
      $pw^* \leftarrow pw^\dagger$
    else:
      if $\exists(sid, pw^\dagger, (r, pk'), apk', \perp) \in \text{List}_{\text{HIC}}$:
        $pw^* \leftarrow pw^\dagger$
      else:
        $pw^* \leftarrow\!\perp$
      add $(sid, apk', pw^*)$ to $\text{List}_{\text{badkeys}}$
    if $pw^* = pw'$:
      lookup $(sid, *, (r, pk'), apk', \perp) \in \text{List}_{\text{HIC}}$
      $c \leftarrow \text{PKE.Enc}(pk', m)$
    else:
      $(pk^*, \_) \leftarrow \text{PKE.KGen}(\lambda)$:
      $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$
      $c \leftarrow \text{PKE.Enc}(pk^*, m')$
  return (CIPHERTEXT, $sid, c$) to $\mathcal{Q}$

**Figure 8.19:** PAPKE-HIC part of the simulator. Part one: KEYGEN + ENCRYPT. This description captures the changes made during game hops. Note that the simulator is merged with $\mathcal{F}_{\text{PAPKE}}$. For the separate ideal-world simulator see Figure 8.24.

## PAPKE-HIC part of the simulator, part 2

On query (DECRYPT, $sid, sk, c$) from $\mathcal{P}$:

if $sid \neq (\mathcal{P}, sid') \vee \nexists(sid, sk, apk, pw) \in \mathsf{List}_{\mathsf{keyrec}}$:
  abort

if $\exists(sid, m, c) \in \mathsf{List}_{\mathsf{encrec}}$ then:
  if $m = \perp \neq \mathsf{PKE.Dec}(sk, c)$ then:
    abort $\mathsf{bad}_6$
  return (PLAINTEXT, $sid, m$) to $\mathcal{P}$

$G_{\geq 7}$

$m^* \leftarrow \mathsf{PKE.Dec}(sk, c)$

$G_{\leq 10}$

$\mathsf{List}_{\mathsf{can}} \leftarrow \{(pw', m') : (sid, pw', *, apk, sk') \in \mathsf{List}_{\mathsf{HIC}} \wedge sk' \neq \perp \neq m' \leftarrow \mathsf{PKE.Dec}(sk', c)\}$
if $|\mathsf{List}_{\mathsf{can}}| > 1$ then:
  abort $\mathsf{bad}_8$
if $|\mathsf{List}_{\mathsf{can}}| = 1$ then:
  $(pw^*, m^*) \leftarrow \mathsf{getFirst}(\mathsf{List}_{\mathsf{can}})$
  if $pw \neq pw^*$ then:
    $m^* \leftarrow \perp$
else:
  $pw^* \leftarrow \perp$
  $m^* \leftarrow \mathsf{PKE.Dec}(sk, c)$
  if $m^* \neq \perp$ then:
    abort $\mathsf{bad}_9$

$G_{\geq 11}$

add $(sid, m^*, c)$ to $\mathsf{List}_{\mathsf{encrec}}$

$G_{\geq 8}$

return (PLAINTEXT, $sid, m^*$) to $\mathcal{P}$

**Figure 8.20:** PAPKE-HIC part of the simulator. Part two: DECRYPT. This description captures the changes made during game hops. Note that the simulator is merged with $\mathcal{F}_{\mathsf{PAPKE}}$. For the separate ideal-world simulator see Figure 8.24.

## PAPKE-HIC part of the simulator in $\mathsf{Game}_{12}$, part 1

On query $(\mathsf{KEYGEN}, sid, pw)$ from $\mathcal{P}$:
- if $sid \neq (\mathcal{P}, sid') \vee \exists(sid, *, *, *) \in \mathsf{List}_{\mathsf{keyrec}}$:
  - abort
- $(\_, sk) \leftarrow \mathsf{PKE.KGen}(\lambda)$
- $apk \leftarrow\!\!\$\ \mathcal{D}$ — $(\mathsf{KEYGEN}, sid) \rightarrow \mathcal{S} \rightarrow (\mathsf{KEYCONF}, sid, apk)$
- if $\exists(sid, apk, *) \in \mathsf{List}_{\mathsf{badkeys}}$:
  - abort $\mathsf{bad}_{10}$
- add $(sid, sk, apk, pw)$ to $\mathsf{List}_{\mathsf{keyrec}}$
- return $(\mathsf{KEYCONF}, sid, apk)$ to $\mathcal{P}$

On query $(\mathsf{ENCRYPT}, sid, apk', pw', m)$ from $\mathcal{Q}$:
- lookup $(sid, *, apk, pw) \in \mathsf{List}_{\mathsf{keyrec}}$
- if $apk = apk'$:
  - $(pk', \_) \leftarrow \mathsf{PKE.KGen}(\lambda)$; $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$; $c \leftarrow \mathsf{PKE.Enc}(pk', m')$ — $(\mathsf{ENC\text{-}L}, sid, apk', |m|) \rightarrow \mathcal{S} \rightarrow (\mathsf{CIPHERTEXT}, sid, c)$
  - if $pw = pw'$:
    - $m^* \leftarrow m$
  - else:
    - $m^* \leftarrow \perp$
  - if $\exists(sid, *, c) \in \mathsf{List}_{\mathsf{encrec}}$:
    - abort $\mathsf{bad}_4$ + $\mathsf{bad}_5$
  - add $(sid, m^*, c)$ to $\mathsf{List}_{\mathsf{encrec}}$
- else:
  - if $\exists(sid, apk', pw^\dagger) \in \mathsf{List}_{\mathsf{badkeys}}$:
    - $pw^* \leftarrow pw^\dagger$
  - else:
    - if $\exists(sid, pw^\dagger, (r, pk'), apk', \perp) \in \mathsf{List}_{\mathsf{HIC}}$: $pw^* \leftarrow pw^\dagger$; else: $pw^* \leftarrow \perp$ — $(\mathsf{GUESS}, sid, apk') \rightarrow \mathcal{S} \rightarrow (\mathsf{GUESS}, sid, pw^*)$
    - add $(sid, apk', pw^*)$ to $\mathsf{List}_{\mathsf{badkeys}}$
  - if $pw^* = pw'$:
    - lookup $(sid, *, (r, pk'), apk', \perp) \in \mathsf{List}_{\mathsf{HIC}}$; $c \leftarrow \mathsf{PKE.Enc}(pk', m)$ — $(\mathsf{ENC\text{-}M}, sid, apk', m) \rightarrow \mathcal{S} \rightarrow (\mathsf{CIPHERTEXT}, sid, c)$
  - else:
    - $(pk^*, \_) \leftarrow \mathsf{PKE.KGen}(\lambda)$:; $m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$; $c \leftarrow \mathsf{PKE.Enc}(pk^*, m')$ — $(\mathsf{ENC\text{-}L}, sid, apk', |m|) \rightarrow \mathcal{S} \rightarrow (\mathsf{CIPHERTEXT}, sid, c)$
- return $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{Q}$

**Figure 8.21:** PAPKE-HIC part of the simulator in $\mathsf{Game}_{12}$ before decomposition into the ideal-world simulator and $\mathcal{F}_{\mathsf{PAPKE}}$. Part one: KEYGEN + ENCRYPT. We visualize the decomposition in gray.

## PAPKE-HIC part of the simulator in $\mathsf{Game}_{12}$, part 2

On query (DECRYPT, $sid, sk, c$) from $\mathcal{P}$:
  if $sid \neq (\mathcal{P}, sid') \vee \nexists(sid, sk, apk, pw) \in \mathsf{List}_{\mathsf{keyrec}}$:
    abort
  if $\exists(sid, m, c) \in \mathsf{List}_{\mathsf{encrec}}$ then:
    if $m = \perp \neq \mathsf{PKE.Dec}(sk, c)$ then:
      abort $\mathsf{bad}_6$
    return (PLAINTEXT, $sid, m$) to $\mathcal{P}$

$\mathsf{List}_{\mathsf{can}} \leftarrow \{(pw', m') : (sid, pw', *, apk, sk') \in \mathsf{List}_{\mathsf{HIC}} \wedge sk' \neq \perp \neq m' \leftarrow \mathsf{PKE.Dec}(sk', c)\}$
if $|\mathsf{List}_{\mathsf{can}}| > 1$ then:
  abort $\mathsf{bad}_8$
if $|\mathsf{List}_{\mathsf{can}}| = 1$ then:
  $(pw^*, m^*) \leftarrow \mathsf{getFirst}(\mathsf{List}_{\mathsf{can}})$
else:
  $pw^* \leftarrow \perp$
  $m^* \leftarrow \mathsf{PKE.Dec}(sk, c)$
  if $m^* \neq \perp$ then:
    abort $\mathsf{bad}_9$

$(\mathsf{DECRYPT}, sid, c) \rightarrow \mathcal{S} \rightarrow (\mathsf{PLAINTEXT}, sid, m^*, pw^*)$

  if $pw \neq pw^*$ then:  / moved from case where 1 record exists for clear separation
    $m^* \leftarrow \perp$
  add $(sid, m^*, c)$ to $\mathsf{List}_{\mathsf{encrec}}$
  return (PLAINTEXT, $sid, m^*$) to $\mathcal{P}$

**Figure 8.22:** PAPKE-HIC part of the simulator in $\mathsf{Game}_{12}$ before decomposition into the ideal-world simulator and $\mathcal{F}_{\mathsf{PAPKE}}$. Part two: DECRYPT. We visualize the decomposition in gray.

## HIC part of the ideal-world simulator $\mathcal{S}$

### *Interface for honest parties:.*

On query $(\mathsf{HIC.Enc}, sid, pw, pk)$ from party $\mathcal{P}$ where $pk \in \mathcal{PK}$:

$r \leftarrow\!\!\$\ \mathcal{R}$
if $\exists(sid, pw, (r, pk), apk, *) \in \mathsf{List}_{\mathsf{HIC}}$:
  return $apk$ to $\mathcal{P}$
else:
  do:
    $apk \leftarrow\!\!\$\ \mathcal{D}$
  while: $(sid, pw, *, apk, *) \in \mathsf{List}_{\mathsf{HIC}}$
  if $\exists(sid, *, *, apk, *) \in \mathsf{List}_{\mathsf{HIC}}$:
    abort $\mathsf{bad}_1$
  add $(sid, pw, (r, pk), apk, \bot)$ to $\mathsf{List}_{\mathsf{HIC}}$
  return $apk$ to $\mathcal{P}$

On query $(\mathsf{HIC.Dec}, sid, pw, apk)$ from party $\mathcal{P}$ where $apk \in \mathcal{D}$:

$(r, pk) \leftarrow \mathsf{HIC.AdvDec}(sid, pw, apk)$
return $pk$ to $\mathcal{P}$

### *Interface for the ideal-world adversary and corrupt parties:.*

On query $(\mathsf{HIC.AdvEnc}, sid, pw, (r, pk), T)$ from adversary $\mathcal{A}$ where $(r, pk) \in \mathcal{D}, T \in \mathcal{PK}$:

if $\exists(sid, pw, (r, pk), apk, *) \in \mathsf{List}_{\mathsf{HIC}}$:
  return $apk$ to $\mathcal{A}$
else:
  do:
    $s \leftarrow\!\!\$\ \mathcal{R}$
  while: $(sid, pw, *, (s, T), *) \in \mathsf{List}_{\mathsf{HIC}}$
  if $\exists(sid, *, *, (s, T), *) \in \mathsf{List}_{\mathsf{HIC}}$:
    abort $\mathsf{bad}_2$
  $apk \leftarrow (s, T)$
  add $(sid, pw, (r, pk), apk, \bot)$ to $\mathsf{List}_{\mathsf{HIC}}$
  return $apk$ to $\mathcal{A}$

On query $(\mathsf{HIC.AdvDec}, sid, pw, apk)$ from adversary $\mathcal{A}$ where $apk \in \mathcal{D}$:

if $\exists(sid, pw, (r, pk), apk, *) \in \mathsf{List}_{\mathsf{HIC}}$:
  return $(r, pk)$ to $\mathcal{A}$
else:
  $r \leftarrow\!\!\$\ \mathcal{R}$
  $(pk, sk) \leftarrow \mathsf{PKE.KGen}(\lambda)$
  if $\exists(sid, *, (*, pk), *, *) \in \mathsf{List}_{\mathsf{HIC}}$:
    abort $\mathsf{bad}_3$
  add $(sid, pw, (r, pk), apk, sk)$ to $\mathsf{List}_{\mathsf{HIC}}$
  return $(r, pk)$ to $\mathcal{A}$

**Figure 8.23:** HIC part of the ideal-world simulator $\mathcal{S}$.

## PAPKE-HIC part of the ideal-world simulator $\mathcal{S}$

On query (KEYGEN, $sid$) from $\mathcal{F}_{\text{PAPKE}}$:

$apk \leftarrow\!\!\$\ \mathcal{D}$
/ The following 2 lines are extra since now the simulator is independent of the functionality,
/ so it cannot use $\text{List}_{\text{keyrec}}$.
/ It also needs to sample $sk$ for $\text{bad}_9$ since that abort does not occur in the functionality.
$(sk, \_) \leftarrow \text{PKE.KGen}(\lambda)$
add $(sid, apk, sk)$ to $\text{List}_{\text{simrec}}$
return $(\text{KEYCONF}, sid, apk)$ to $\mathcal{F}_{\text{PAPKE}}$

On query (GUESS, $sid, apk$) from $\mathcal{F}_{\text{PAPKE}}$:

if $\exists (sid, pw, (r, pk), apk, \bot) \in \text{List}_{\text{HIC}}$:
return $(\text{GUESS}, sid, pw)$ to $\mathcal{F}_{\text{PAPKE}}$
else:
return $(\text{GUESS}, sid, \bot)$ to $\mathcal{F}_{\text{PAPKE}}$

On query (ENC-L, $sid, apk, |m|$) from $\mathcal{F}_{\text{PAPKE}}$:

$m' \leftarrow\!\!\$\ \{0,1\}^{|m|}$
$(pk, \_) \leftarrow \text{PKE.KGen}(\lambda)$
$c \leftarrow \text{PKE.Enc}(pk, m')$
return $(\text{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\text{PAPKE}}$

On query (ENC-M, $sid, apk, m$) from $\mathcal{F}_{\text{PAPKE}}$:

lookup $(sid, pw, (r, pk), apk, \bot) \in \text{List}_{\text{HIC}}$
$c \leftarrow \text{PKE.Enc}(pk, m)$
return $(\text{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\text{PAPKE}}$

On query (DECRYPT, $sid, c$) from $\mathcal{F}_{\text{PAPKE}}$:

lookup $(sid, apk, sk) \in \text{List}_{\text{simrec}}$
$\text{List}_{\text{can}} \leftarrow \{(pw, m) : (sid, pw, *, apk, sk') \in \text{List}_{\text{HIC}} \ \wedge sk' \neq \bot \neq m \leftarrow \text{PKE.Dec}(sk', c)\}$
if $|\text{List}_{\text{can}}| > 1$ then:
abort $\text{bad}_8$
if $|\text{List}_{\text{can}}| = 1$ then:
$(pw, m) \leftarrow \text{getFirst}(\text{List}_{\text{can}})$
else:
$m^* \leftarrow \text{PKE.Dec}(sk, c)$ / $sk$ is independent of $apk$
if $m^* \neq \bot$ then:
abort $\text{bad}_9$
$(pw, m) \leftarrow (\bot, \bot)$
return $(\text{PLAINTEXT}, sid, m, pw)$ to $\mathcal{F}_{\text{PAPKE}}$

**Figure 8.24:** PAPKE-HIC part of the ideal-world simulator $\mathcal{S}$ (interactions with $\mathcal{F}_{\text{PAPKE}}$).

## 8.4 PAPKE-HIC Performance

In this section, we compare the performance of PAPKE-IC and PAPKE-HIC when instantiated with post-quantum PKEs (built from post-quantum KEMs). Recall that the main advantage of PAPKE-HIC compared to PAPKE-IC is the fact that it does not require an ideal cipher over (large) groups, which generally hinders performance. To demonstrate the concrete advantage of our HIC-based approach, we selected the following post-quantum KEMs: ML-KEM [NIS24b], FrodoKEM [Fro25] and Classic McEliece [Alb+22].

First, in Section 8.4.1, we analyze whether these KEMs satisfy the requirements of Theorem 9. After that, in Section 8.4.2, we discuss specific implementation caveats for each KEM. Finally, in

Section 8.4.3, we present and analyze the results of our benchmarks.

### 8.4.1 PAPKE-Eligible KEMs

We consider three post-quantum KEMs, ML-KEM [NIS24b], FrodoKEM [Fro25], and Classic McEliece [Alb+22], as candidates for PAPKE instantiation.

Recall that PAPKE-IC and PAPKE-HIC require a PKE scheme that is IND-CCA, ANO-CCA, SROB-CCA, UNI-PK, and DROB-CCA secure (Theorem 9). We construct a post-quantum PKE using the standard KEM+DEM construction presented in Definition 12.

IND-CCA-security of the PKE is easily achievable since each mentioned KEM is IND-CCA-secure (Theorem 1).

As shown in Section 8.1, deriving DROB-CCA from SROB-CCA incurs a quadratic security degradation, which then becomes the dominant term in the security bound of Theorem 9. Considering concrete security, this may require choosing larger security parameters for PAPKE instantiations. However, as shown in Section 8.1.2, this is unnecessary for KEMs with the Prefix-Hashing FO transform (pFO) described in Figure 8.3. We now discuss why each PKE satisfies the other properties (apart from IND-CCA).

**ML-KEM.** The hybrid PKE with ML-KEM is ANO-CCA-secure [MX23] and SROB-CCA-secure [GMP22]. Moreover, in ML-KEM, the public key is computationally indistinguishable from uniform under the MLWE assumption [Beg+23], thus supporting the UNI-PK property in practice. ML-KEM uses the pFO transform, and thus directly satisfies DROB-CCA with the bound from Theorem 5.

**FrodoKEM.** The hybrid PKE with FrodoKEM is ANO-CCA-secure and SROB-CCA-secure as shown in [Xag21]. FrodoKEM uses the pFO transform, so it satisfies DROB-CCA with the bound from Theorem 5. FrodoKEM satisfies UNI-PK under the LWE assumption by an analogous argument as for ML-KEM in [Beg+23].

**Classic McEliece.** It was shown in [Xag21] that Classic McEliece satisfies ANO-CCA under additional assumptions. UNI-PK is considered a standard assumption for Classic McEliece [Alb+22]. However, [GMP22] demonstrated that Classic McEliece does not satisfy SCFR-CCA, which means that the hybrid PKE with Classic McEliece is not SROB-CCA-secure. A modified variant of Classic McEliece, proposed in [Xag21], yields a hybrid PKE that satisfies SROB-CCA. This modified variant essentially integrates the pFO transform internally into Classic McEliece, thus directly satisfying DROB-CCA per Theorem 5.

Still, modifying the internals of an existing scheme can be impractical. That is why we introduced the *hashed-public-key* (HPK) KEM transform in Section 8.1.3, a simpler black-box modification resulting in a DROB-CCA-secure scheme (via Theorem 6). We show that HPK preserves anonymity and that any HPK-transformed KEM is SCFR-CCA-secure and consequently the hybrid PKE is SROB-CCA-secure.

### 8.4.2 PAPKE Implementations

We discuss caveats of implementing PAPKE-IC and PAPKE-HIC with ML-KEM, FrodoKEM and Classic McEliece.

Our C++ implementation [Oup25b] uses libraries Botan [The26a] and Catch2 [The26b]. Specifically, we use Botan for cryptographic primitives, such as ML-KEM and AES-256-GCM, and Catch2 for benchmarking.

#### 8.4.2.1 ML-KEM-based PAPKE

The crux of implementing PAPKE-IC is the instantiation of the ideal cipher over the ML-KEM public key space. Recall from Section 8.2.3.2 that there exist two approaches to implement an ideal cipher over the ML-KEM public key space.

The first approach is to combine an ideal cipher over bitstrings with a quasi-bijective encoding of the public keys to bitstrings [GJK21]. This requires an ideal cipher over a large bitstring domain, which can be realized either via domain extenders [Cor+10] or via an 8-round Feistel network [DS16].

The second approach is to use the 8-round Feistel construction [DS16] to build an ideal cipher over the public key space directly, which in turn requires four hash-to-group operations.[10]

Regarding the PAPKE-HIC implementation, there currently exists only one HIC construction, M2F (Figure 8.13), which requires one hash-to-group operation, i.e., hashing to the ML-KEM public key space. To realize this, we implemented two hash-to-group approaches: the first one employs the SampleNTT subroutine from [NIS24b, Algorithm 7], the same rejection sampling routine used by ML-KEM key generation; the second one is to use Kemeleon [GSV24], specifically Kemeleon decoding (combined with an XOF), which maps bitstrings to ML-KEM public keys.

To summarize, we have implemented and benchmarked the following implementations of post-quantum PAPKE with ML-KEM:

1) PAPKE-IC is implemented by combining Kemeleon—used to encode ML-KEM as a pseudo-uniform bitstring—with an ideal cipher instantiated over bitstrings, either via a domain extender or an 8-round Feistel network.

2) PAPKE-IC is also implemented by directly employing an 8-round Feistel, in which four of the hashing operations are hash-to-group operations realized using either SampleNTT or Kemeleon decoding.

3) PAPKE-HIC is implemented with a 2-round modified Feistel M2F, in which the single hash-to-group operation is implemented either via SampleNTT or Kemeleon decoding.

Note that the underlying public-key encryption scheme is the KEM+DEM hybrid construction $\mathsf{PKE}^{hy}$ (Definition 12) with ML-KEM serving as the KEM. Recall from Section 8.3 that we require the PKE to be SROB-CCA-secure, which in this case requires the DEM to be FROB-secure [GMP22, Theorem 2]. Standard AEAD constructions, such as AES-GCM, are typically not FROB-secure [Dod+18]. Therefore, we use the CTX construction introduced in [CR22, Figure 2]

[10] One wire carries the group element while the other carries a random bitstring.

which builds an FROB-secure[11] DEM from an AEAD and a collision-resistant hash function. We use AES-256-GCM as the underlying AEAD and SHA-256 as the hash function.

#### 8.4.2.2 FrodoKEM-based and Classic McEliece-based PAPKE

Compared to ML-KEM, FrodoKEM's public keys are indistinguishable from random bitstrings under the LWE assumption [GSV24] since it uses a power-of-two modulus. Consequently, hashing to the FrodoKEM public key space can be achieved using an eXtendable-Output Function (XOF) defined in Definition 8, such as SHAKE-128. We implemented three variants of FrodoKEM-based PAPKE: PAPKE-IC with domain-extender IC [Cor+10], PAPKE-IC with 8-round Feistel IC [DS16] and PAPKE-HIC.

Similar to FrodoKEM, the indistinguishability of Classic McEliece public keys from random bitstrings is an established assumption [Alb+22]. Therefore, the PAPKE implementations are essentially identical. We implemented the same PAPKE variants as with FrodoKEM. Regarding parameter selection, we focus only on the security level 1 Classic McEliece parameter set: 348864. Note that the implementation also uses the HPK transform presented in Section 8.1.3.

### 8.4.3 PAPKE Performance Results

We executed the benchmarks on a machine with AMD EPYC 7543 (2,8 GHz) as the CPU, running Ubuntu 24.04 and compiled using GCC 13. We used the `--benchmark-samples 1000` Catch2 parameter during benchmarking.

#### 8.4.3.1 ML-KEM-based PAPKE Results

Table 8.1 provides performance results of our PAPKE-IC and PAPKE-HIC implementations with ML-KEM. See Figure 8.25 for visualization of the data.

We found that the Kemeleon-based hash-to-group is the bottleneck in variants that use it. This is because Kemeleon decoding requires division of large integers (around 6000 bits for ML-KEM-512). Note that PAPKE-IC with bitstring ideal cipher constructions use Kemeleon *encoding* during key generation and *decoding* during encryption. This explains why those instantiations perform better than PAPKE-HIC (Kemeleon variant) which relies on Kemeleon decoding for the hash-to-group, at key generation, but perform essentially the same at encryption.

Overall, the best performing constructions rely on the SampleNTT-based hashing to group, and of those, PAPKE-HIC performs better than PAPKE-IC. This is to be expected when we trade four hash-to-group operations for one hash-to-group operation and a block cipher.

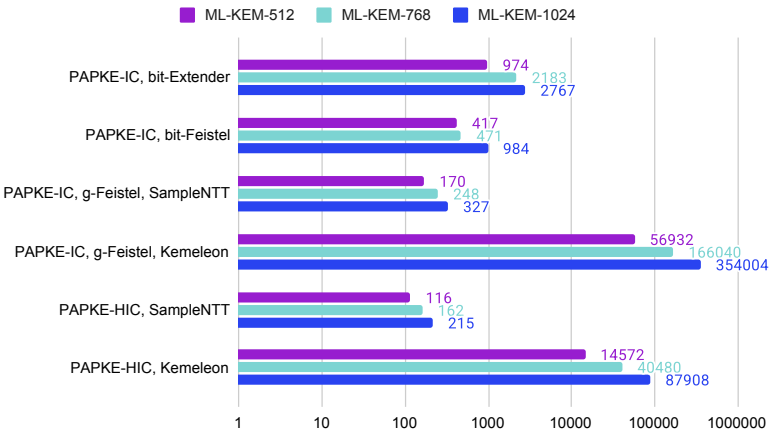
It's also worth noting that the (relative) high variance of PAPKE-IC variants with bitstring ideal cipher is caused by rejection of keypairs—only certain ML-KEM public keys can be Kemeleon-encoded. For ML-KEM-512 the success probability is $\approx 56\%$ [GSV24], but for ML-KEM-768 it is $\approx 83\%$, which explains the smaller relative variance for ML-KEM-768.

[11] The paper in question [CR22] uses a different security notion ($\mathrm{CAE}_{\mathrm{XX}}$) which implies FROB from [GMP22].

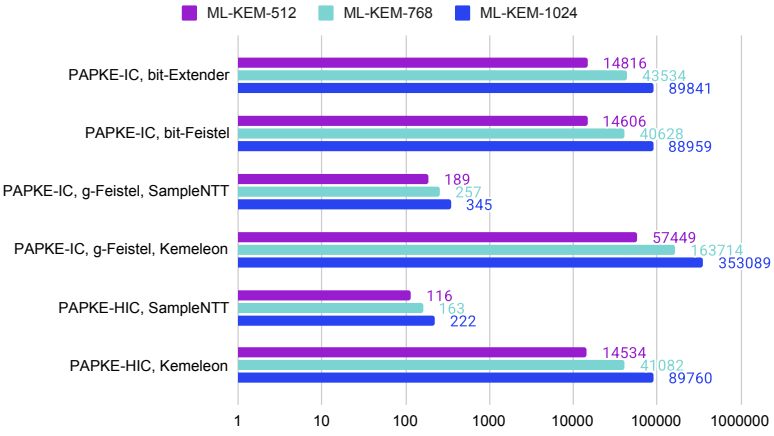
**Table 8.1:** ML-KEM-based PAPKE Performance

| Sec. Level | Variant | Key Gen. | Encryption | Decryption |
|---|---|---|---|---|
| 1 | PAPKE-HIC, SampleNTT | 116 ± 8 | 116 ± 11 | 98 ± 7 |
| | PAPKE-IC, g-Feistel, SampleNTT | 170 ± 6 | 189 ± 15 | 98 ± 10 |
| | PAPKE-IC, bit-Feistel | 417 ± 258 | 14 606 ± 126 | 103 ± 13 |
| | PAPKE-IC, bit-Extender | 974 ± 227 | 14 816 ± 114 | 99 ± 10 |
| | PAPKE-HIC, Kemeleon | 14 572 ± 222 | 14 534 ± 185 | 100 ± 10 |
| | PAPKE-IC, g-Feistel, Kemeleon | 56 932 ± 503 | 57 449 ± 864 | 98 ± 9 |
| 3 | PAPKE-HIC, SampleNTT | 162 ± 9 | 163 ± 10 | 171 ± 13 |
| | PAPKE-IC, g-Feistel, SampleNTT | 248 ± 10 | 257 ± 11 | 170 ± 9 |
| | PAPKE-IC, bit-Feistel | 471 ± 174 | 40 628 ± 308 | 179 ± 10 |
| | PAPKE-IC, bit-Extender | 2 183 ± 181 | 43 534 ± 431 | 178 ± 13 |
| | PAPKE-HIC, Kemeleon | 40 480 ± 268 | 41 082 ± 632 | 183 ± 15 |
| | PAPKE-IC, g-Feistel, Kemeleon | 166 040 ± 1 810 | 163 714 ± 2 374 | 197 ± 19 |
| 5 | PAPKE-HIC, SampleNTT | 215 ± 9 | 222 ± 14 | 269 ± 18 |
| | PAPKE-IC, g-Feistel, SampleNTT | 327 ± 10 | 345 ± 13 | 266 ± 14 |
| | PAPKE-IC, bit-Feistel | 984 ± 577 | 88 959 ± 1 228 | 280 ± 21 |
| | PAPKE-IC, bit-Extender | 2 767 ± 657 | 89 841 ± 972 | 282 ± 20 |
| | PAPKE-HIC, Kemeleon | 87 908 ± 372 | 89 760 ± 1 157 | 284 ± 21 |
| | PAPKE-IC, g-Feistel, Kemeleon | 354 004 ± 4 455 | 353 089 ± 3 801 | 290 ± 16 |

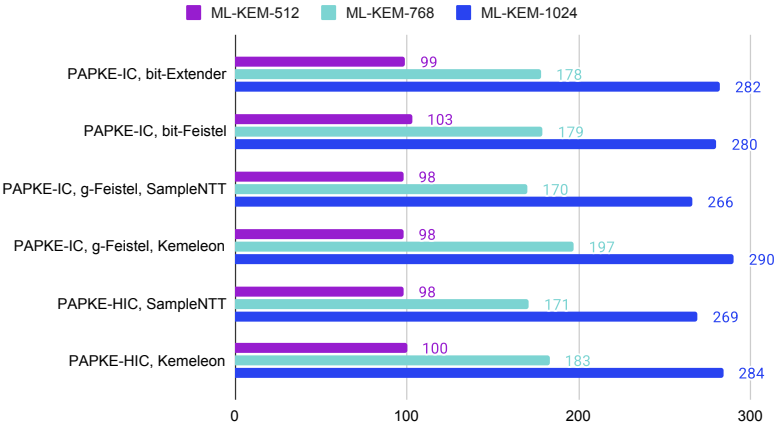
Values are in microseconds in the format: mean ± std. dev. Security levels: 1, 3 and 5 correspond to parameter sets ML-KEM-512, ML-KEM-768 and ML-KEM-1024, respectively. SampleNTT/Kemeleon refers to the hash-to-group implementation, "g-Feistel" refers to the 8-round Feistel with hash-to-group, and "bit-Feistel/Extender" refers to combining bitstring IC with Kemeleon encoding; see Section 8.4.2.1 for details.




(a) Key generation




(b) Encryption




(c) Decryption

**Figure 8.25:** ML-KEM-based PAPKE performance. Values are in microseconds.

Table 8.2: FrodoKEM-based PAPKE Performance

| Sec. Level | Variant | Key Gen. | Encryption | Decryption |
|---|---|---|---|---|
| 1 | PAPKE-HIC | 1 734 ± 16 | 1 730 ± 38 | 1 723 ± 23 |
| | PAPKE-IC, bit-Feistel | 1 855 ± 23 | 1 877 ± 28 | 1 758 ± 24 |
| | PAPKE-IC, bit-Extender | 48 257 ± 311 | 18 667 ± 614 | 1 742 ± 34 |
| 3 | PAPKE-HIC | 3 782 ± 65 | 3 702 ± 53 | 3 709 ± 53 |
| | PAPKE-IC, bit-Feistel | 3 830 ± 28 | 3 894 ± 45 | 3 708 ± 50 |
| | PAPKE-IC, bit-Extender | 50 031 ± 222 | 50 563 ± 623 | 3 706 ± 78 |
| 5 | PAPKE-HIC | 6 554 ± 93 | 6 513 ± 96 | 6 516 ± 95 |
| | PAPKE-IC, bit-Feistel | 6 780 ± 111 | 6 750 ± 76 | 6 515 ± 125 |
| | PAPKE-IC, bit-Extender | 147 078 ± 1 819 | 148 049 ± 1 856 | 6 567 ± 132 |

Values are in microseconds in the format: mean ± std. dev. Security levels 1, 3 and 5 correspond to parameter sets FrodoKEM-640, FrodoKEM-976 and FrodoKEM-1344, respectively. "bit-Feistel/Extender" refers to bitstring IC implementation; see Section 8.4.2.2 for details.

#### 8.4.3.2 FrodoKEM-based PAPKE Results

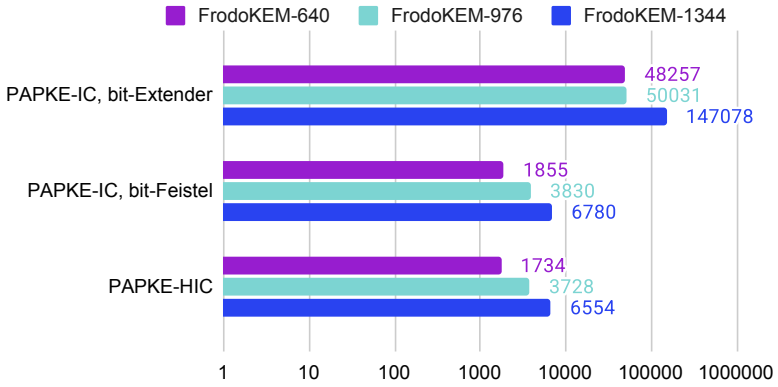
We provide results for FrodoKEM in Table 8.2. See Figure 8.26 for visualization of the data. Observe that the domain-extender IC variant is no longer competitive due to the size of FrodoKEM's public keys. However, since constructing an IC over FrodoKEM's public keys does not require additional public-key encoding and decoding (like Kemeleon for ML-KEM), we see that the performance gain of PAPKE-HIC is not as significant when compared to the ML-KEM-based PAPKE-HIC. For example, for key generation and encryption with FrodoKEM-640, PAPKE-HIC takes 0.92× the time of (8-round Feistel) PAPKE-IC. This shows that although the PAPKE-IC construction requires three additional hash-to-group calls using SHAKE-128, the hash-to-group overhead is not as significant as for the ML-KEM-based construction.

#### 8.4.3.3 Classic McEliece-based PAPKE Results

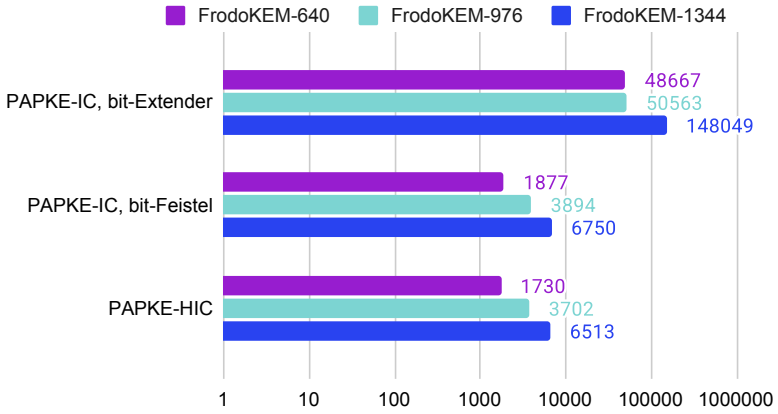
We provide results for the Classic McEliece-based PAPKE in Table 8.3. See Figure 8.27 for visualization of the data. As with FrodoKEM, we observe that the domain-extender-based IC is not competitive at all. Classic McEliece key generation is slow; therefore, there is no significant difference between PAPKE-HIC and PAPKE-IC with 8-round Feistel during key generation. However, for encryption, PAPKE-HIC takes 0.26× the time of PAPKE-IC. This illustrates that PAPKE-IC requires four hash-to-group operations compared to the one required by PAPKE-HIC. In this case, the hashing to group (XOF) is the bottleneck since Classic McEliece encapsulation is very fast.

## 8.5 Conclusion

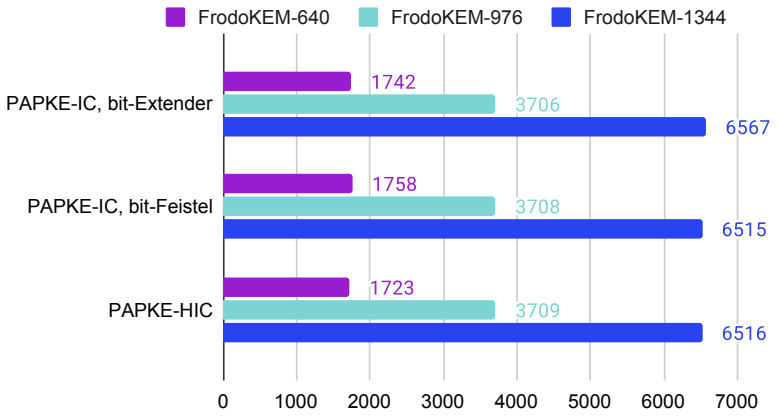
This chapter was a bit different from other chapters because we focused on the migration of a cryptographic primitive, PAPKE, to post-quantum cryptography instead of a protocol or a standard. We defined a new public-key encryption property and proved its relation to other




(a) Key generation




(b) Encryption




(c) Decryption

**Figure 8.26:** FrodoKEM-based PAPKE performance. Values are in microseconds.

**Table 8.3:** Classic McEliece-based PAPKE Performance

| Variant | Key Gen. | Encryption | Decryption |
|---|---|---|---|
| PAPKE-HIC | 225 977 ± 125 216 | 1 080 ± 44 | 68 289 ± 692 |
| PAPKE-IC, bit-Feistel | 239 348 ± 136 300 | 4 176 ± 58 | 67 787 ± 442 |
| PAPKE-IC, bit-Extender | 3 260 260 ± 123 669 | 3 408 270 ± 410 071 | 68 055 ± 1 486 |

Classic McEliece parameter set 348864. Values are in microseconds in the format: mean ± std. dev. "bit-Feistel/Extender" refers to bitstring IC implementation; see Section 8.4.2.2 for details.

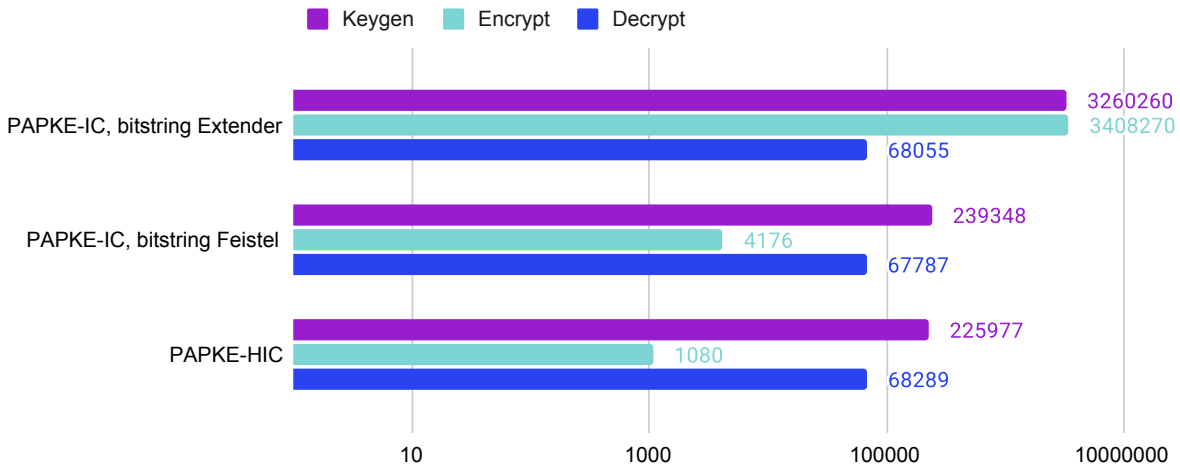



**Figure 8.27:** Classic McEliece (parameter set 348864) PAPKE performance. Values are in microseconds.

already established properties. Building this theoretical foundation was necessary for the second part of the chapter, where we introduced a new PAPKE construction called PAPKE-HIC. The benefit of PAPKE-HIC compared to existing constructions is that it does not require an ideal cipher over a group, which makes post-quantum implementations simpler and faster. In order to show that our construction is secure, we provided a detailed proof in the Universal Composability framework. Finally, in order to demonstrate the performance benefits of PAPKE-HIC, we implemented post-quantum variants of the existing construction PAPKE-IC and our new construction PAPKE-HIC, and conducted a performance evaluation. Our results showed that PAPKE-HIC's advantage is most noticeable either when instantiated with KEMs that require public-key quasi-bijective encodings, such as ML-KEM, or KEMs with large public keys, such as Classic McEliece.

# Chapter 9

# Conclusion and Outlook

9.1 Conclusion . . . . . 271
9.2 Outlook . . . . . 272

In this thesis, we have concerned ourselves with migration to post-quantum cryptography. This chapter summarizes our results in Section 9.1 and provides an outlook in Section 9.2 with respect to the current state of post-quantum cryptography and its deployment in practice.

## 9.1 Conclusion

This thesis covered several topics in cryptography; namely: the public-key infrastructure (Chapter 3), the TLS protocol (Chapter 4), the XML standard and the SAML single sign-on protocol (Chapter 5), the eIDAS-compliant AdES signature format (Chapter 6), and the PAPKE primitive (Chapter 8). For each topic, we provided an introduction covering the background necessary for the subsequent analysis of the impact of quantum computers.

Although the introductory sections of each chapter contain no novel results—apart from the, at times trivial, security analyses—we believe they still provide value for readers wishing to familiarize themselves with these topics. For well-established topics such as PKI and TLS, a plethora of introductory material exists. For more niche topics such as SAML Single Sign-On, XML security standards, or AdES, this is not the case; understanding them typically requires sifting through numerous specifications and standards. For instance, our overview of AdES signatures (Section 6.1) introduces the PDF standard, the Cryptographic Message Syntax (CMS) standard, and the JSON Web Signature (JWS) standard—the AdES framework itself is composed of multiple ETSI standards. Furthermore, the PAPKE chapter contains a succinct introduction to the Universal Composability (UC) framework, for which, in our opinion, beginner-friendly study material is difficult to find. Our UC proof of PAPKE-HIC in Section 8.3 (which is a novel result) is also deliberately detailed with the aim of serving as a non-trivial yet rigorous example of a UC proof.

The introductory sections are followed by descriptions of post-quantum solutions addressing the identified quantum-related vulnerabilities. For the majority of topics, we present two variants

of post-quantum solutions distinguished by their security guarantees: the purely post-quantum variant and the hybrid post-quantum variant.

Since post-quantum TLS has been an active research area for over a decade, the post-quantum TLS chapter (Chapter 4) takes the form of a survey of the numerous proposals in the literature. Regarding post-quantum PKI (Chapter 3), while this work is not the first to discuss the concept of post-quantum certificates, we provide a comprehensive treatment of post-quantum PKI as a whole, covering revocation, timestamping, and more.

Turning to the more niche topics of post-quantum XML, SAML and AdES—motivated by our collaboration with LuxTrust—the proposed purely post-quantum solutions are typically fairly straightforward; however, requiring hybrid post-quantum security makes the design of an adequate solution more challenging, particularly with respect to security, ease of implementation and performance. We have also conducted numerous performance evaluations and provide proof-of-concept implementations for post-quantum AdES, SAML, XML and PKI [Oup25a; Oup24c]. The final chapter, on post-quantum PAPKE (Chapter 8), is the only one not motivated by our LuxTrust collaboration. It represents a somewhat different category of post-quantum migration, focusing on modifying a specific primitive to be more "post-quantum-friendly." As part of this work, we also provide what are, to the best of our knowledge, the first implementations of the PAPKE primitive; these implementations inherently contain multiple implementations of an ideal cipher over a group, which may prove useful beyond the PAPKE context.

We believe this work may serve a purpose beyond its immediate scope. The methodology adopted throughout—identifying and analyzing vulnerable parts of cryptographic systems, and proposing both purely post-quantum and hybrid post-quantum alternatives—can be readily applied to other protocols or primitives undergoing migration to post-quantum cryptography. Furthermore, some organizations may find this work particularly valuable as a practical reference when planning their own post-quantum migration, as further supported by our example migration plan presented in Chapter 7. In this sense, the contributions of this thesis extend beyond the specific standards, protocols and primitives examined.

## 9.2 Outlook

We began our work on this thesis—on migration to quantum-safe cryptography—in the middle of 2022. A lot has happened over the past four years (not only in the cryptography world).

In the summer of 2022, one of the most promising post-quantum KEMs, the isogeny-based SIKE, was completely broken [CD23]. We say *completely*, because it was not a theoretical attack that merely caused the scheme to lose a few bits of security falling below the specified security level, but an attack that could be executed on a laptop, recovering the key in minutes. This attack dealt a significant blow not only to the perceived security of isogeny-based cryptography, but also to the general trust in the security of post-quantum algorithms—since SIKE had been seemingly secure for over ten years.[1] Although hybrid approaches have been proposed long before SIKE's demise, the incident reinforced the idea that hybrid post-quantum cryptography should be used before moving to purely post-quantum cryptography.

[1] Technically, SIKE is the KEM-variant of the key exchange SIDH, which was proposed in 2011. However, they share the underlying "hard" problem, which was shown to be easy.

A few weeks later, NIST announced that ML-KEM, ML-DSA, FN-DSA and SLH-DSA would be standardized, and that it planned to select an additional code-based KEM, BIKE, HQC, or Classic McEliece, after another round of evaluation [Ala+22]. A few months later, NIST also announced a new standardization process with the aim of selecting additional post-quantum signatures [NIS22].

In mid-2024, two years after the announcement, NIST published the standards for ML-KEM [NIS24b], ML-DSA [NIS24a] and SLH-DSA [NIS24c], enabling developers to begin implementing these algorithms without concerns about future compatibility issues. At the time of writing, the standard for FN-DSA is still not even in its draft phase.

We witnessed the first large deployment of post-quantum algorithms in 2023, when Google made hybrid post-quantum key exchange in TLS the default [Goo23b]; Firefox deployed hybrid post-quantum key exchange a year later [Moz24]. Recently, Google has announced the next step, deploying post-quantum authentication in TLS, which completes TLS's migration to post-quantum cryptography [Goo26b]. Of course, the client (the web browser) is only one of the two protocol participants—the server must migrate as well. There has also been development on the TLS server side migration: OpenSSL, the most prominent TLS library, has supported hybrid post-quantum key exchange since mid-2025 [Ope25]; the same goes for other TLS libraries such as Google's BoringSSL [Goo26a] or Amazon's s2n [AWS26].

Another significant post-quantum migration concerned messaging apps, namely, Signal and iMessage. Since 2024, both of these messaging apps use hybrid post-quantum key exchange to protect against "store now, decrypt later" attacks. Similar to TLS, the migration has been seamless. The last step to achieving full post-quantum security is migrating the authentication mechanisms as well—this step is still pending.

However, these examples are only the tip of the iceberg—big tech companies have always been quick to adopt new cryptographic primitives. Other sectors such as banking, healthcare, critical infrastructure, governmental organizations, militaries—all handling sensitive data—are known to be slow to adopt new approaches (for various reasons). For these sectors, the post-quantum migration is just beginning. Notably, most of these organizations rely on some PKI. As discussed in Chapter 3, migrating a PKI is not difficult from a technical point of view, but the difficulties emerge due to external factors such as hardware dependencies or regulatory compliance.

SAML and XML are another topic we have investigated in this work. The continued use of SAML and XML is, in a sense, a reflection of how slowly the sectors deploying them evolve. These technologies are not insecure, but they are inferior: the standards are under-specified, which has led to numerous security vulnerabilities [Som+12; Som14; JS11; SS12]. More modern and better-designed alternatives exist: OpenID Connect (OIDC) for SAML SSO and JSON-based signatures/encryption for XML signatures/encryption. Although we have proposed post-quantum solutions for SAML and XML, we believe that an inter-protocol migration deserves equal consideration—for instance, migrating classical SAML SSO to post-quantum OIDC [Sch+22a].

Meanwhile, as touched upon in Section 1.4, quantum computer development has also progressed. Seemingly, there have not been any significant hurdles over the last few years, which suggests that the optimistic prediction of cryptographically relevant quantum computers (CRQCs) being developed in the 2030s may come to fruition. Both private and public funding for quantum-related research has been increasing significantly, which suggests that the rate of progress may even increase in the coming years [McK25].

In principle, some fundamental barrier may prevent the construction of a CRQC. Even in that case, the post-quantum cryptography effort would still have been beneficial for several reasons.

First, although post-quantum algorithms have not yet demonstrated superior performance to their classical counterparts, their development has driven the investigation of novel approaches to cryptography—most notably lattice-based cryptography, which finds applications in other areas such as fully homomorphic encryption. Second, the NIST standardization process constituted an unprecedented open and collaborative cryptographic effort in modern cryptography: researchers were invited both to submit their proposals and to analyze their security; the surviving standards have consequently been subjected to greater scrutiny than, arguably, elliptic curve cryptography received at the time of its deployment. Third, the migration effort highlighted the importance of a sometimes neglected topic in cryptographic engineering—cryptographic agility: the algorithm-agnostic design of cryptographic systems. The notion of cryptographic agility remains valuable regardless of whether CRQCs ultimately materialize, because no cryptographic algorithm should be assumed to last *forever*.

*Fin*

# Bibliography


[Aar+24a] Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, et al. *PERK*. National Institute of Standards and Technology, 2024 (cit. on p. 14).

[Aar+24b] Marius A. Aardal, Gora Adj, Diego F. Aranha, et al. *SQIsign*. National Institute of Standards and Technology, 2024 (cit. on pp. 36, 39).

[ABN10] Michel Abdalla, Mihir Bellare, and Gregory Neven. "Robust Encryption". In: *Theory of Cryptography*. Ed. by Daniele Micciancio. Springer, 2010, pp. 480–497. ISBN: 978-3-642-11799-2. DOI: 10.1007/978-3-642-11799-2_28 (cit. on pp. 28, 215, 228).

[ABR99] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. *DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem*. 1999. URL: https://eprint.iacr.org/1999/007 (visited on 2025-04-11). Pre-published (cit. on pp. 215, 228).

[Adl79] Leonard Adleman. "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography". In: *20th Annual Symposium on Foundations of Computer Science (Sfcs 1979)*. 20th Annual Symposium on Foundations of Computer Science (Sfcs 1979). 1979-10, pp. 55–60. DOI: 10.1109/SFCS.1979.2 (cit. on p. 6).

[AH99] Leonard M. Adleman and Ming-Deh A. Huang. "Function Field Sieve Method for Discrete Logarithms over Finite Fields". In: *Information and Computation* 151.1 (1999-05-25), pp. 5–16. ISSN: 0890-5401. DOI: 10.1006/inco.1998.2761 (cit. on p. 6).

[Ado08] Adobe. *Portable Document Format — Part 1: PDF 1.7*. PDF 32000-1:2008. 2008-07-01. URL: https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/PDF32000_2008.pdf (visited on 2024-07-03) (cit. on p. 170).

[Agu+24] Carlos Aguilar Melchor, Slim Bettaieb, Loïc Bidoux, et al. *SDitH — Syndrome Decoding in the Head*. National Institute of Standards and Technology, 2024 (cit. on pp. 35, 39).

[Agu+25] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, et al. *Hamming Quasi-Cyclic (HQC)*. 4th round version. 2025. URL: https://pqc-hqc.org/doc/hqc_specifications_2025_08_22.pdf. Archived at https://web.archive.org/web/20260420030552/https://pqc-hqc.org/doc/hqc_specifications_2025_08_22.pdf (cit. on pp. 34, 36).

[Agu+22] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, et al. *HQC*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 14, 110).

[Aki+24] Rika Akiyama, Hiroki Furue, Yasuhiko Ikematsu, et al. *QR-UOV*. National Institute of Standards and Technology, 2024 (cit. on pp. 35, 39).

[Ala+19] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, et al. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST Internal or Interagency Report (NISTIR) 8240. National Institute of Standards and Technology, 2019-01-31. DOI: 10.6028/NIST.IR.8240 (cit. on p. 13).

[Ala+22] Gorjan Alagic, Daniel Apon, David Cooper, et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST Internal or Interagency Report (NISTIR) 8413. National Institute of Standards and Technology, 2022-09-29. DOI: 10.6028/NIST.IR.8413-upd1 (cit. on pp. 13, 103, 104, 108, 109, 273).

[Ala+24] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, et al. *Status Report on the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process*. NIST Internal or Interagency Report (NISTIR) 8528. National Institute of Standards and Technology, 2024-10-24. DOI: 10.6028/NIST.IR.8528 (cit. on p. 14).

[Ala+25a] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, et al. *Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST Internal or Interagency Report (NISTIR) 8545. National Institute of Standards and Technology, 2025-03-11. DOI: 10.6028/NIST.IR.8545 (cit. on pp. 13, 110).

[Ala+25b] Gorjan Alagic, Joseph Carolan, Christian Majenz, et al. *The Sponge Is Quantum Indifferentiable*. 2025. URL: https://eprint.iacr.org/2025/731 (visited on 2026-04-15). Pre-published (cit. on p. 11).

[Alb+22] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, et al. *Classic McEliece*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 14, 36, 110, 115, 244, 261, 262, 264).

[Alk+16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, et al. "Post-Quantum Key Exchange - A New Hope". In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, 2016, pp. 327–343. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim (cit. on pp. 109, 111, 112, 117).

[Aln+25] Nouri Alnahawi, David Haas, Erik Mauß, et al. *SoK: PQC PAKEs - Design, Security and Performance*. 2025. URL: https://eprint.iacr.org/2025/119 (visited on 2026-02-09). Pre-published (cit. on p. 214).

[Aln+24] Nouri Alnahawi, Johannes Müller, Jan Oupický, et al. "A Comprehensive Survey on Post-Quantum TLS". In: *IACR Communications in Cryptology* 1.2 (2024-07-08). ISSN: 3006-5496. DOI: 10.62056/ahee0iuc (cit. on pp. 17, 19, 20, 98).

[Ama+24] Alessandro Amadori, Thomas Attema, Maxime Bombar, et al. *The PQC Migration Handbook: Guidelines for Migrating to Post-Quantum Cryptography*. AIVD / CWI / TNO, 2024-12. URL: https://publications.tno.nl/publication/34643386/fXcPVHsX/TNO-2024-pqc-en.pdf. Archived at https://web.archive.org/web/20260419170345/https://publications.tno.nl/publication/34643386/fXcPVHsX/TNO-2024-pqc-en.pdf (cit. on p. 201).

[ANS23] ANSSI. *ANSSI Views on the Post-Quantum Cryptography Transition (2023 Follow Up)*. 2023-12-21. URL: https://messervices.cyber.gouv.fr/documents-guides/follow_up_position_paper_on_post_quantum_cryptography.pdf. Archived at https://web.archive.org/web/20260423143033/https://messervices.cyber.gouv.fr/documents-guides/follow_up_position_paper_on_post_quantum_cryptography.pdf (cit. on p. 14).

[ANS+24] ANSSI, BSI, NLNCSA, et al. *Position Paper on Quantum Key Distribution*. 2024-01-26. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Quantum_Positionspapier.pdf?__blob=publicationFile&v=4. Archived at https://web.archive.org/web/20260415111222/https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Quantum_Positionspapier.pdf?__blob=publicationFile&v=4 (cit. on p. 15).

[Apa25] Apache Santuario Community. *Apache Santuario*. 2025. URL: https://santuario.apache.org/ (cit. on pp. 19, 136, 208).

[App24] Apple. *SC-081: Introduce Schedule of Reducing Validity and Data Reuse Periods by Clintwilson · Pull Request #553 · Cabforum/Servercert*. GitHub. 2024. URL: https://github.com/cabforum/servercert/pull/553 (visited on 2025-06-24) (cit. on p. 64).

[Aqu+25] Nick Aquina, Bruno Cimoli, Soumya Das, et al. "A Critical Analysis of Deployed Use Cases for Quantum Key Distribution and Comparison with Post-Quantum Cryptography". In: *EPJ Quantum Technology* 12.1 (2025-05-06), p. 51. ISSN: 2196-0763. DOI: 10.1140/epjqt/s40507-025-00350-5 (cit. on p. 15).

[Ara+24] Nicolas Aragon, Magali Bardet, Loïc Bidoux, et al. *RYDE*. National Institute of Standards and Technology, 2024 (cit. on p. 35).

[Ara+22] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, et al. *BIKE*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 110, 113).

[Arr+25] Afonso Arriaga, Manuel Barbosa, Stanislaw Jarecki, et al. "C'est Très CHIC: A Compact Password-Authenticated Key Exchange from Lattice-Based KEM". In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Springer Nature, 2025, pp. 3–33. ISBN: 978-981-96-0935-2. DOI: 10.1007/978-981-96-0935-2_1 (cit. on p. 28).

[Arr+26] Afonso Arriaga, David Mestel, Jan Oupický, et al. "HIC Is All You Need: Practical Post-Quantum Password-Authenticated Public-Key Encryption". In: *Topics in Cryptology – CT-RSAC 2026*. Ed. by Feng-Hao Liu. Springer Nature Switzerland, 2026, pp. 127–154. ISBN: 978-3-032-22931-1. DOI: 10.1007/978-3-032-22931-1_5 (cit. on pp. 18, 20, 215).

[AWS26] AWS. *S2n-Tls*. 2026. URL: https://github.com/aws/s2n-tls (visited on 2026-03-05) (cit. on p. 273).

[Bab+26] Ryan Babbush, Adam Zalcman, Craig Gidney, et al. *Securing Elliptic Curve Cryptocurrencies against Quantum Vulnerabilities: Resource Estimates and Mitigations*. 2026-04-15. DOI: 10.48550/arXiv.2603.28846. arXiv: 2603.28846 [quant-ph]. Preprinted (cit. on pp. 9, 10, 201).

[Bal+24] Marco Baldi, Alessandro Barenghi, Michele Battagliola, et al. *CROSS — Codes and Restricted Objects Signature Scheme*. National Institute of Standards and Technology, 2024 (cit. on pp. 35, 39).

[BC20] Utsav Banerjee and Anantha P. Chandrakasan. “Efficient Post-Quantum TLS Handshakes Using Identity-Based Key Exchange from Lattices”. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. ICC 2020 - 2020 IEEE International Conference on Communications (ICC). 2020-06, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148829 (cit. on p. 117).

[Bar+24] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, et al. “X-Wing”. In: *IACR Communications in Cryptology* 1.1 (2024-04-09). ISSN: 3006-5496. DOI: 10.62056/a3qj89n4e (cit. on pp. 47, 112).

[Bar20] Elaine Barker. *Recommendation for Key Management Part 1: General*. NIST SP 800-57pt1r5. National Institute of Standards and Technology, 2020-05, NIST SP 800–57pt1r5. DOI: 10.6028/NIST.SP.800-57pt1r5 (cit. on pp. 11, 117).

[Bar+19] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, et al. *Automatic Certificate Management Environment (ACME)*. Request for Comments RFC 8555. Internet Engineering Task Force, 2019-03. 95 pp. DOI: 10.17487/RFC8555 (cit. on pp. 52, 64).

[BGJ25] Alison Becker, Rebecca Guthrie, and Michael J. Jenkins. *Related Certificates for Use in Multiple Authentications within a Protocol*. Request for Comments RFC 9763. Internet Engineering Task Force, 2025-06. 13 pp. DOI: 10.17487/RFC9763 (cit. on p. 90).

[Beg+23] Hugo Beguinet, Céline Chevalier, David Pointcheval, et al. “GeT a CAKE: Generic Transformations from Key Encaspulation Mechanisms to Password Authenticated Key Exchanges”. In: *Applied Cryptography and Network Security*. Ed. by Mehdi Tibouchi and XiaoFeng Wang. Springer Nature Switzerland, 2023, pp. 516–538. ISBN: 978-3-031-33491-7. DOI: 10.1007/978-3-031-33491-7_19 (cit. on pp. 28, 262).

[Bel+01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, et al. “Key-Privacy in Public-Key Encryption”. In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by Colin Boyd. Springer, 2001, pp. 566–582. ISBN: 978-3-540-45682-7. DOI: 10.1007/3-540-45682-1_33 (cit. on p. 28).

[BR93] Mihir Bellare and Phillip Rogaway. “Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols”. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS ’93. Association for Computing Machinery, 1993-12-01, pp. 62–73. ISBN: 978-0-89791-629-5. DOI: 10.1145/168588.168596 (cit. on p. 237).

[Ben+26] David Benjamin, Devon O’Brien, Bas Westerbaan, et al. *Merkle Tree Certificates*. Internet Draft draft-ietf-plants-merkle-tree-certs-02. Internet Engineering Task Force, 2026-03-02. 83 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-plants-merkle-tree-certs (visited on 2026-03-09) (cit. on p. 106).

[BB14] Charles H. Bennett and Gilles Brassard. “Quantum Cryptography: Public Key Distribution and Coin Tossing”. In: *Theoretical Computer Science*. Theoretical Aspects of Quantum Cryptography – Celebrating 30 Years of BB84 560 (2014-12-04), pp. 7–11. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2014.05.025 (cit. on p. 15).

[Ber09] Daniel J Bernstein. *Cost Analysis of Hash Collisions: Will Quantum Computers Make SHARCS Obsolete?* 2009. URL: https://cr.yp.to/hash/collisioncost-20090823.pdf. Archived at https://web.archive.org/web/20260306141302/https://cr.yp.to/hash/collisioncost-20090823.pdf (cit. on pp. 11, 118).

[Ber+20] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, et al. *NTRU Prime*. National Institute of Standards and Technology, 2020 (cit. on pp. 36, 107, 108, 110).

[Ber+22] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, et al. "OpenSSLNTRU: Faster Post-Quantum TLS Key Exchange". In: 31st USENIX Security Symposium (USENIX Security 22). 2022, pp. 845–862. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/bernstein (cit. on pp. 107, 108, 110).

[BBD09] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, eds. *Post-Quantum Cryptography*. Springer, 2009. ISBN: 978-3-540-88701-0. DOI: 10.1007/978-3-540-88702-7 (cit. on p. 13).

[Ber+19] Daniel J. Bernstein, Tung Chou, Tanja Lange, et al. *Classic McEliece*. National Institute of Standards and Technology, 2019 (cit. on p. 34).

[Ber+13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, et al. "Elligator: Elliptic-Curve Points Indistinguishable from Uniform Random Strings". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. Association for Computing Machinery, 2013-11-04, pp. 967–980. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516734 (cit. on p. 239).

[Ber+17] Daniel J. Bernstein, Nadia Heninger, Paul Lou, et al. "Post-Quantum RSA". In: *Post-Quantum Cryptography*. Ed. by Tanja Lange and Tsuyoshi Takagi. Springer International Publishing, 2017, pp. 311–329. ISBN: 978-3-319-59879-6. DOI: 10.1007/978-3-319-59879-6_18 (cit. on p. 8).

[BL25] Daniel J. Bernstein and Tanja Lange. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. 2025. URL: https://bench.cr.yp.to (visited on 2025-12-10) (cit. on pp. 37–39).

[Ber+07] Guido Bertoni, Joan Daemen, Michaël Peeters, et al. "Sponge Functions". In: *ECRYPT Hash Workshop*. Vol. 2007. 2007. URL: https://csrc.nist.rip/groups/ST/hash/documents/JoanDaemen.pdf. Archived at https://web.archive.org/web/20260423150848/https://csrc.nist.rip/groups/ST/hash/documents/JoanDaemen.pdf (cit. on p. 11).

[Beu22] Ward Beullens. "Breaking Rainbow Takes a Weekend on a Laptop". In: *Advances in Cryptology – CRYPTO 2022*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Springer Nature Switzerland, 2022, pp. 464–479. ISBN: 978-3-031-15979-4. DOI: 10.1007/978-3-031-15979-4_16 (cit. on pp. 35, 104).

[Beu+24a] Ward Beullens, Fabio Campos, Sofía Celi, et al. *MAYO*. National Institute of Standards and Technology, 2024 (cit. on pp. 35, 39).

[Beu+24b] Ward Beullens, Ming-Shing Chen, Jintai Ding, et al. *UOV — Unbalanced Oil and Vinegar*. National Institute of Standards and Technology, 2024 (cit. on p. 35).

[Beu+19] Ward Beullens, Bart Preneel, Alan Szepieniec, et al. *LUOV*. National Institute of Standards and Technology, 2019 (cit. on p. 35).

[Bin+19a] Nina Bindel, Sedat Akleylek, Erdem Alkim, et al. *qTESLA*. National Institute of Standards and Technology, 2019 (cit. on pp. 103, 104).

[Bin+19b] Nina Bindel, Jacqueline Brendel, Marc Fischlin, et al. "Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange". In: *Post-Quantum Cryptography*. Ed. by Jintai Ding and Rainer Steinwandt. Springer International Publishing, 2019, pp. 206–226. ISBN: 978-3-030-25510-7. DOI: 10.1007/978-3-030-25510-7_12 (cit. on p. 46).

[Bin+17a] Nina Bindel, Udyani Herath, Matthew McKague, et al. "Transitioning to a Quantum-Resistant Public Key Infrastructure". In: *Post-Quantum Cryptography: 8th International Workshop, PQCrypto 2017, Utrecht, the Netherlands, June 26-28, 2017, Proceedings 8*. Springer, 2017, pp. 384–405. DOI: 10.1007/978-3-319-59879-6_22 (cit. on pp. 43, 44, 46).

[Bin+17b] Nina Bindel, Udyani Herath, Matthew McKague, et al. *Transitioning to a Quantum-Resistant Public Key Infrastructure*. 2017. URL: https://eprint.iacr.org/2017/460 (visited on 2026-04-23). Pre-published (cit. on p. 45).

[BR02] John Black and Phillip Rogaway. "Ciphers with Arbitrary Finite Domains". In: *Topics in Cryptology — CT-RSA 2002*. Ed. by Bart Preneel. Springer, 2002, pp. 114–130. ISBN: 978-3-540-45760-2. DOI: 10.1007/3-540-45760-7_9 (cit. on p. 239).

[Blo25] Blockchain.com. *Blockchain.Com, Total Hash Rate (TH/s)*. 2025-11-21. URL: https://www.blockchain.com/explorer/charts/hash-rate (visited on 2025-11-21). Archived at https://web.archive.org/web/20260423152644/https://www.blockchain.com/explorer/charts/hash-rate (cit. on p. 11).

[Böc23] Hanno Böck. *Fermat Factorization in the Wild*. 2023. URL: https://eprint.iacr.org/2023/026 (visited on 2025-11-19). Pre-published (cit. on p. 5).

[Boe+08] Sharon Boeyen, Stefan Santesson, Tim Polk, et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Request for Comments RFC 5280. Internet Engineering Task Force, 2008-05. 151 pp. DOI: 10.17487/RFC5280 (cit. on pp. 15, 52–54, 56, 57, 60, 78, 82, 165).

[Bon+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, et al. "Random Oracles in a Quantum World". In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Springer, 2011, pp. 41–69. ISBN: 978-3-642-25385-0. DOI: 10.1007/978-3-642-25385-0_3 (cit. on p. 219).

[Bon+25] Corey Bonnell, John Gray, D. Hook, et al. *A Mechanism for Encoding Differences in Paired Certificates*. Internet Draft draft-bonnell-lamps-chameleon-certs-06. Internet Engineering Task Force, 2025-04-16. 52 pp. URL: https://datatracker.ietf.org/doc/draft-bonnell-lamps-chameleon-certs-06 (visited on 2025-09-26) (cit. on pp. 90, 92).

[Bos+16] Joppe Bos, Craig Costello, Leo Ducas, et al. "Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Association for Computing Machinery, 2016-10-24, pp. 1006–1018. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978425 (cit. on pp. 107–109, 111).

[Bos+15] Joppe W Bos, Craig Costello, Michael Naehrig, et al. "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem". In: *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 553–570. DOI: 10.1109/SP.2015.40 (cit. on pp. 107, 109, 111, 116).

[Bos+24] Joppe W. Bos, Olivier Bronchain, Léo Ducas, et al. *HAWK*. National Institute of Standards and Technology, 2024 (cit. on pp. 34, 37).

[Bra+19a] Tatiana Bradley, Jan Camenisch, Stanislaw Jarecki, et al. *Password-Authenticated Public-Key Encryption*. 2019. URL: https://eprint.iacr.org/2019/199 (visited on 2025-01-10). Pre-published (cit. on pp. 18, 228, 234, 242).

[Bra+19b] Tatiana Bradley, Jan Camenisch, Stanislaw Jarecki, et al. "Password-Authenticated Public-Key Encryption". In: *Applied Cryptography and Network Security*. Ed. by Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, et al. Springer International Publishing, 2019, pp. 442–462. ISBN: 978-3-030-21568-2. DOI: 10.1007/978-3-030-21568-2_22 (cit. on pp. 214, 215, 229, 231, 234, 235, 240, 242).

[BHT98] Gilles Brassard, Peter Hoyer, and Alain Tapp. "Quantum Algorithm for the Collision Problem". In: vol. 1380. 1998, pp. 163–169. DOI: 10.1007/BFb0054319. arXiv: quant-ph/9705002 (cit. on pp. 11, 12, 32, 118).

[Bra17] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. Request for Comments RFC 8259. Internet Engineering Task Force, 2017-12. 16 pp. DOI: 10.17487/RFC8259 (cit. on p. 175).

[BI20] Anne Broadbent and Rabib Islam. "Quantum Encryption with Certified Deletion". In: *Theory of Cryptography*. Ed. by Rafael Pass and Krzysztof Pietrzak. Springer International Publishing, 2020, pp. 92–122. ISBN: 978-3-030-64381-2. DOI: 10.1007/978-3-030-64381-2_4 (cit. on p. 15).

[BL20] Anne Broadbent and Sébastien Lord. "Uncloneable Quantum Encryption via Oracles". In: *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*. Ed. by Steven T. Flammia. Vol. 158. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 4:1–4:22. ISBN: 978-3-95977-146-7. DOI: 10.4230/LIPIcs.TQC.2020.4 (cit. on p. 15).

[BSI26] BSI. *Cryptographic Mechanisms: Recommendations and Key Lengths*. Federal Office for Information Security, Germany, 2026-01-23. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=6 (visited on 2026-04-20). Archived at https://web.archive.org/web/20260423154014/https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=6 (cit. on pp. 6, 11, 12, 14, 43, 109, 117).

[BSI25] BSI. *Status of Quantum Computer Development*. 2.2. Federal Office for Information Security, Germany, 2025-12-19. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/Entwicklungstand_QC_V_2_2.pdf?__blob=publicationFile&v=5. Archived at https://web.archive.org/web/20251219224332/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/St

udien/Quantencomputer/Entwicklungstand_QC_V_2_2.pdf?__blob=publicationFile&v=5 (cit. on p. 8).

[BKW13] Johannes A. Buchmann, Evangelos Karatsiolis, and Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. Springer, 2013. ISBN: 978-3-642-40656-0. DOI: 10.1007/978-3-642-40657-7 (cit. on p. 56).

[Bür+20] Kevin Bürstinghaus-Steinbach, Christoph Krauß, Ruben Niederhagen, et al. "Post-Quantum TLS on Embedded Systems: Integrating and Evaluating Kyber and SPHINCS+ with Mbed TLS". In: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. ASIA CCS '20. Association for Computing Machinery, 2020-10, pp. 841–852. ISBN: 978-1-4503-6750-9. DOI: 10.1145/3320269.3384725 (cit. on pp. 102, 103, 107, 109).

[Cai+26] Madelyn Cain, Qian Xu, Robbie King, et al. *Shor's Algorithm Is Possible with as Few as 10,000 Reconfigurable Atomic Qubits*. 2026-03-30. DOI: 10.48550/arXiv.2603.28627. arXiv: 2603.28627 [quant-ph]. Pre-published (cit. on pp. 7, 9).

[CC21] Matt Campagna and Eric Crockett. *Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS)*. Internet-Draft draft-campagna-tls-bike-sike-hybrid-07. Internet Engineering Task Force / Internet Engineering Task Force, 2021-09-02. 17 pp. URL: https://datatracker.ietf.org/doc/draft-campagna-tls-bike-sike-hybrid/07/ (cit. on p. 112).

[Cam+24] Fabio Campos, Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, et al. "Optimizations and Practicality of High-Security CSIDH". In: *IACR Communications in Cryptology* 1.1 (1 2024-04-09). ISSN: 3006-5496. DOI: 10.62056/anjbksdja (cit. on pp. 36, 116, 243).

[Can01] R. Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. 42nd IEEE Symposium on Foundations of Computer Science. 2001-10, pp. 136–145. DOI: 10.1109/SFCS.2001.959888 (cit. on pp. 231, 233).

[Can05] R. Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. 2005-12-14. URL: https://eprint.iacr.org/archive/2000/067/20051214:064128 (visited on 2026-02-09). Pre-published (cit. on p. 232).

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. "The Random Oracle Methodology, Revisited". In: *J. ACM* 51.4 (2004-07-01), pp. 557–594. ISSN: 0004-5411. DOI: 10.1145/1008731.1008734 (cit. on p. 237).

[Cas+20] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, et al. *GeMSS*. National Institute of Standards and Technology, 2020 (cit. on p. 104).

[CD23] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH". In: *Advances in Cryptology – EUROCRYPT 2023*. Ed. by Carmit Hazay and Martijn Stam. Springer Nature Switzerland, 2023, pp. 423–447. ISBN: 978-3-031-30589-4. DOI: 10.1007/978-3-031-30589-4_15 (cit. on pp. 13, 36, 111, 272).

[Cas+18] Wouter Castryck, Tanja Lange, Chloe Martindale, et al. “CSIDH: An Efficient Post-Quantum Commutative Group Action”. In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven Galbraith. Springer International Publishing, 2018, pp. 395–427. ISBN: 978-3-030-03332-3. DOI: 10.1007/978-3-030-03332-3_15 (cit. on pp. 36, 116, 243).

[Cel+21] Sofía Celi, Armando Faz-Hernández, Nick Sullivan, et al. “Implementing and Measuring KEMTLS”. In: *Progress in Cryptology – LATINCRYPT 2021*. Lecture Notes in Computer Science. Springer International Publishing, 2021, pp. 88–107. ISBN: 978-3-030-88238-9. DOI: 10.1007/978-3-030-88238-9_5 (cit. on p. 115).

[Cel+22] Sofía Celi, Jonathan Hoyland, Douglas Stebila, et al. “A Tale of Two Models: Formal Verification of KEMTLS via Tamarin”. In: *Computer Security – ESORICS 2022*. Lecture Notes in Computer Science. Springer Nature Switzerland, 2022, pp. 63–83. ISBN: 978-3-031-17143-7. DOI: 10.1007/978-3-031-17143-7_4 (cit. on p. 115).

[CR22] John Chan and Phillip Rogaway. “On Committing Authenticated-Encryption”. In: *Computer Security – ESORICS 2022*. Ed. by Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, et al. Springer Nature Switzerland, 2022, pp. 275–294. ISBN: 978-3-031-17146-8. DOI: 10.1007/978-3-031-17146-8_14 (cit. on pp. 263, 264).

[Che+20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, et al. *NTRU*. National Institute of Standards and Technology, 2020 (cit. on pp. 107, 110, 112).

[Che+22] Vincent Cheval, Charlie Jacomme, Steve Kremer, et al. “SAPIC+: Protocol Verifiers of the World, Unite!” In: *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022-08, pp. 3935–3952. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/cheval (cit. on p. 115).

[CFS26] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. *Reducing the Number of Qubits in Quantum Discrete Logarithms on Elliptic Curves*. 2026. URL: https://eprint.iacr.org/2026/280 (visited on 2026-04-20). Pre-published (cit. on p. 10).

[Cla23] Classic McEliece team. *Draft: Information Security — Encryption Algorithms — Part 1978: Classic McEliece*. 2023-04-19. URL: https://classic.mceliece.org/iso-mceliece-20230419.pdf (visited on 2025-11-25). Archived at https://web.archive.org/web/20250704053737/https://classic.mceliece.org/iso-mceliece-20230419.pdf (cit. on pp. 14, 34).

[Clo26a] Cloudflare. *Adoption & Usage Worldwide | Cloudflare Radar*. 2026-01-22. URL: https://radar.cloudflare.com/adoption-and-usage (visited on 2026-01-22) (cit. on p. 98).

[Clo26b] Cloudflare. *Cloudflare Targets 2029 for Full Post-Quantum Security*. The Cloudflare Blog. 2026-04-07. URL: https://blog.cloudflare.com/post-quantum-roadmap/ (visited on 2026-04-23). Archived at https://web.archive.org/web/20260408105614/https://blog.cloudflare.com/post-quantum-roadmap/ (cit. on p. 201).

[Con25] Deire Connolly. *ML-KEM Post-Quantum Key Agreement for TLS 1.3*. Internet Draft draft-ietf-tls-mlkem-04. Internet Engineering Task Force, 2025-07-22. 10 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-mlkem (visited on 2025-10-31) (cit. on p. 203).

[Coo+20] David A. Cooper, Daniel C. Apon, Quynh H. Dang, et al. *Recommendation for Stateful Hash-Based Signature Schemes*. National Institute of Standards and Technology, 2020-10-29. DOI: 10.6028/NIST.SP.800-208 (cit. on pp. 14, 35, 102).

[Cor+10] Jean-Sébastien Coron, Yevgeniy Dodis, Avradip Mandal, et al. "A Domain Extender for the Ideal Cipher". In: *Theory of Cryptography*. Ed. by Daniele Micciancio. Springer, 2010, pp. 273–289. ISBN: 978-3-642-11799-2. DOI: 10.1007/978-3-642-11799-2_17 (cit. on pp. 238, 244, 263, 264).

[CPS19] Eric Crockett, Christian Paquin, and Douglas Stebila. *Prototyping Post-Quantum and Hybrid Key Exchange and Authentication in TLS and SSH*. 858. 2019. URL: https://eprint.iacr.org/2019/858 (visited on 2021-11-15) (cit. on pp. 105, 107, 111).

[DPH25] Flo D, Michael P, and Britta Hale. *Terminology for Post-Quantum Traditional Hybrid Schemes*. Request for Comments RFC 9794. Internet Engineering Task Force, 2025-06. 15 pp. DOI: 10.17487/RFC9794 (cit. on pp. 42, 78).

[DAn+20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, et al. *SABER*. National Institute of Standards and Technology, 2020 (cit. on pp. 107, 109, 111).

[DS16] Yuanxi Dai and John Steinberger. "Indifferentiability of 8-Round Feistel Networks". In: *Advances in Cryptology – CRYPTO 2016*. Ed. by Matthew Robshaw and Jonathan Katz. Springer, 2016, pp. 95–120. ISBN: 978-3-662-53018-4. DOI: 10.1007/978-3-662-53018-4_4 (cit. on pp. 238, 244, 263, 264).

[Dar+25] Pierrick Dartois, Jonathan Komada Eriksen, Tako Boris Fouotsa, et al. "PEGASIS: Practical Effective Class Group Action Using 4-Dimensional Isogenies". In: *Advances in Cryptology – CRYPTO 2025*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Springer Nature Switzerland, 2025, pp. 67–99. ISBN: 978-3-032-01855-7. DOI: 10.1007/978-3-032-01855-7_3 (cit. on p. 36).

[De +20] Alfonso Francisco De Abiega-L'Eglisse, Kevin Andrae Delgado-Vargas, Fernando Quetzalcoatl Valencia-Rodriguez, et al. "Performance of New Hope and CRYSTALS-Dilithium Postquantum Schemes in the Transport Layer Security Protocol". In: *IEEE Access* 8 (2020), pp. 213968–213980. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3040324 (cit. on pp. 103, 107).

[Del25] Deloitte. *Digital Usage & Online Behaviours*. 2025-05-20. URL: https://www.deloitte.com/ie/en/Industries/tmt/research/digital-consumer-trends/digital-usage-online-behaviours.html (visited on 2025-11-17). Archived at https://web.archive.org/web/20260217025316/https://www.deloitte.com/ie/en/Industries/tmt/research/digital-consumer-trends/digital-usage-online-behaviours.html (cit. on p. 3).

[DH76] W. Diffie and M. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976-11), pp. 644–654. ISSN: 1557-9654. DOI: 10.1109/TIT.1976.1055638 (cit. on p. 3).

[Din+17] Jintai Ding, Saed Alsayigh, Jean Lancrenon, et al. "Provably Secure Password Authenticated Key Exchange Based on RLWE for the Post-Quantum World". In: *Topics in Cryptology – CT-RSA 2017*. Vol. 10159. Springer International Publishing, 2017, pp. 183–204. ISBN: 978-3-319-52152-7. DOI: 10.1007/978-3-319-52153-4_11 (cit. on p. 116).

[Din+20] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, et al. *Rainbow*. National Institute of Standards and Technology, 2020 (cit. on pp. 35, 40, 104).

[Din+21] Jintai Ding, Joshua Deaton, Vishakha, et al. "The Nested Subset Differential Attack". In: *Advances in Cryptology – EUROCRYPT 2021*. Ed. by Anne Canteaut and François-Xavier Standaert. Springer International Publishing, 2021, pp. 329–347. ISBN: 978-3-030-77870-5. DOI: 10.1007/978-3-030-77870-5_12 (cit. on p. 35).

[Dod+18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, et al. "Fast Message Franking: From Invisible Salamanders to Encryptment". In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Springer International Publishing, 2018, pp. 155–186. ISBN: 978-3-319-96884-1. DOI: 10.1007/978-3-319-96884-1_6 (cit. on p. 263).

[Don17] Jason A. Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel". In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/ (cit. on p. 15).

[DG22] Ronny Döring and Marc Geitz. "Post-Quantum Cryptography in Use: Empirical Analysis of the TLS Handshake Performance". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium. 2022-04, pp. 1–5. DOI: 10.1109/NOMS54207.2022.9789913 (cit. on pp. 102–104, 107–111).

[Dum+21] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, et al. "Faster Lattice-Based KEMs via a Generic Fujisaki-Okamoto Transform Using Prefix Hashing". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Association for Computing Machinery, 2021-11-13, pp. 2722–2737. ISBN: 978-1-4503-8454-4. DOI: 10.1145/3460120.3484819 (cit. on pp. 219, 223, 224).

[Dwo07] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. NIST Special Publication (SP) 800-38D. National Institute of Standards and Technology, 2007-11-28. DOI: 10.6028/NIST.SP.800-38D (cit. on p. 134).

[Eas11] Donald E. Eastlake 3rd. *Transport Layer Security (TLS) Extensions: Extension Definitions*. Request for Comments RFC 6066. Internet Engineering Task Force, 2011-01. 25 pp. DOI: 10.17487/RFC6066 (cit. on p. 64).

[Elg85] T. Elgamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *IEEE Transactions on Information Theory* 31.4 (1985-07), pp. 469–472. ISSN: 1557-9654. DOI: 10.1109/TIT.1985.1057074 (cit. on p. 243).

[Ell70] J. H. Ellis. *The Possibility of Secure Non-Secret Digital Encryption*. GCHQ, 1970-01. URL: https://web.archive.org/web/20251107163758/https://nsarchive.gwu.edu/sites/default/files/documents/3035765/Document-02.pdf (visited on 2025-11-07) (cit. on p. 3).

[ENI25] ENISA. *Agreed Cryptographic Mechanisms*. Version 2.0. 2025-04. URL: https://certification.enisa.europa.eu/document/download/a845662b-aee0-484e-9191-890c4cfa7aaa_en?filename=ECCG%20Agreed%20Cryptographic%20Mechanisms%20version%202.pdf (visited on 2025-06-12). Archived at https://web.archive.org/web/20260420101110/https://certification.enisa.europa.eu/document/download/a845662b-aee0-484e-9191-890c4cfa7aaa_en?filename=ECCGAgreedCryptographicMechanismsversion2.pdf (cit. on p. 201).

[ETS23a] ETSI. *Electronic Signatures and Infrastructures (ESI); CAdES Digital Signatures; Part 1: Building Blocks and CAdES Baseline Signatures*. EN 319 122-1. Version 1.3.1. 2023-06. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31912201/01.03.01_60/en_31912201v010301p.pdf (cit. on pp. 59, 159).

[ETS17a] ETSI. *Electronic Signatures and Infrastructures (ESI); CAdES Digital Signatures; Part 3: Incorporation of Evidence Record Syntax (ERS) Mechanisms in CAdES*. TS 119 122-3. Version 1.1.1. 2017-01. URL: https://www.etsi.org/deliver/etsi_ts/119100_119199/11912203/01.01.01_60/ts_11912203v010101p.pdf (cit. on p. 181).

[ETS13] ETSI. *Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)*. TS 101 733. Version 2.2.1. 2013-04. URL: https://www.etsi.org/deliver/etsi_ts/101700_101799/101733/02.02.01_60/ts_101733v020201p.pdf (cit. on p. 164).

[ETS23b] ETSI. *Electronic Signatures and Infrastructures (ESI); Cryptographic Suites*. TS 119 312. Version 1.4.3. 2023-08. URL: https://www.etsi.org/deliver/etsi_ts/119300_119399/11931212/01.04.03_60/ts_119312v010403p.pdf (cit. on pp. 164, 169, 182, 187, 188).

[ETS24a] ETSI. *Electronic Signatures and Infrastructures (ESI); JAdES Digital Signatures; Part 1: Building Blocks and JAdES Baseline Signatures*. TS 119 182-1. Version 1.2.1. 2024-07. URL: https://www.etsi.org/deliver/etsi_ts/119100_119199/11918201/01.02.01_60/ts_11918201v010201p.pdf (cit. on pp. 170, 182).

[ETS24b] ETSI. *Electronic Signatures and Infrastructures (ESI); PAdES Digital Signatures; Part 1: Building Blocks and PAdES Baseline Signatures*. EN 319 142-1. Version 1.2.1. 2024-01. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.02.01_60/en_31914201v010201p.pdf (cit. on pp. 175, 179, 180, 182).

[ETS09] ETSI. *Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a Framework Document for PAdES*. TS 102 778-1. Version 1.1.1. 2009-07. URL: https://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf (cit. on pp. 170, 187, 189).

[ETS23c] ETSI. *Electronic Signatures and Infrastructures (ESI); Policy and Security Requirements for Trust Service Providers Issuing Certificates; Part 1: General Requirements*. EN 319 411-1. Version 1.4.1. 2023-10. URL: https://www.etsi.org/deliver/etsi_en/319400_319499/31941101/01.04.01_60/en_31941101v010401p.pdf (cit. on pp. 57, 64).

[ETS21] ETSI. *Electronic Signatures and Infrastructures (ESI); Policy and Security Requirements for Trust Service Providers Issuing Certificates; Part 2: Requirements for Trust Service Providers Issuing EU Qualified Certificates*. EN 319 411-2. Version 2.3.1. 2021-05. URL: https://www.etsi.org/deliver/etsi_en/319400_319499/31941102/02.03.01_60/en_31941102v020301p.pdf (cit. on p. 181).

[ETS16a] ETSI. *Electronic Signatures and Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation*. EN 319 102-1. Version 1.1.1. 2016-12. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31910201/01.01.01_60/en_31910201v010101p.pdf (cit. on pp. 15, 52, 156, 187, 190).

[ETS22] ETSI. *Electronic Signatures and Infrastructures (ESI); XAdES Digital Signatures; Part 1: Building Blocks and XAdES Baseline Signatures*. EN 319 132-1. Version 1.2.1. 2022-02. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31913201/01.02.01_60/en_31913201v010201p.pdf (cit. on pp. 162, 163, 182, 187, 188).

[ETS16b] ETSI. *Electronic Signatures and Infrastructures (ESI); XAdES Digital Signatures; Part 2: Extended XAdES Signatures*. EN 319 132-2. Version 1.1.1. 2016-04. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31913202/01.01.01_60/en_31913202v010101p.pdf (cit. on pp. 181, 182).

[ETS10] ETSI. *Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)*. TS 101 903. Version 1.4.2. 2010-12. URL: https://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.04.02_60/ts_101903v010402p.pdf (cit. on p. 162).

[ETS17b] ETSI. *Quantum-Safe Cryptography (QSC); Limits to Quantum Computing Applied to Symmetric Key Sizes*. GR QSC 006. Version 1.1.1. ETSI, 2017-02. URL: https://www.etsi.org/deliver/etsi_gr/QSC/001_099/006/01.01.01_60/gr_QSC006v010101p.pdf (cit. on pp. 12, 117).

[Eur25a] European Commission. *A Coordinated Implementation Roadmap for the Transition to Post-Quantum Cryptography*. Part 1, Version 1.1. European Commission, 2025-06-11. URL: https://ec.europa.eu/newsroom/dae/redirection/document/117507 (visited on 2025-06-30). Archived at https://web.archive.org/web/20260221095142/https://ec.europa.eu/newsroom/dae/redirection/document/117507 (cit. on p. 200).

[Eur25b] European Commission. *Digital Signature Service - DSS*. 2025. URL: https://ec.europa.eu/digital-building-blocks/sites/spaces/DIGITAL/pages/467109107/Digital+Signature+Service+-+DSS (visited on 2026-04-20) (cit. on p. 19).

[Eur25c] European Commission. *DSS Demonstration WebApp*. 2025. URL: https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/validation (visited on 2025-02-21). Archived at https://web.archive.org/web/20260418231228/https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/validation (cit. on pp. 190, 208).

[Eur15] European Commission. *Implementing Decision - 2015/1506 - EN - EUR-Lex*. Official Journal of the European Union, 2015. URL: https://eur-lex.europa.eu/eli/dec_impl/2015/1506/oj/ (cit. on p. 162).

[Eur14] European Commission. *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market and Repealing Directive 1999/93/EC*. 2014-07-23. URL: http://data.europa.eu/eli/reg/2014/910/oj/ (cit. on pp. 19, 155).

[Flu17] Scott Fluhrer. *Reassessing Grover's Algorithm*. 2017. URL: https://eprint.iacr.org/2017/811 (visited on 2025-11-21). Pre-published (cit. on p. 11).

[Fro25] FrodoKEM team. *FrodoKEM: Learning With Errors Key Encapsulation, Preliminary Standardization Proposal*. 2025-09-29. URL: https://frodokem.org/files/FrodoKEM_standard_proposal_20250929.pdf (visited on 2025-11-25). Archived at https://web.archive.org/web/20260419170344/https://frodokem.org/files/FrodoKEM_standard_proposal_20250929.pdf (cit. on pp. 14, 34, 36, 109, 201, 223, 261, 262).

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Journal of Cryptology* 26.1 (2013-01-01), pp. 80–101. ISSN: 1432-1378. DOI: 10.1007/s00145-011-9114-1 (cit. on p. 243).

[Gaj+23] Phillip Gajland, Bor de Kock, Miguel Quaresma, et al. *Swoosh: Efficient Lattice-Based Non-Interactive Key Exchange*. 2023. URL: https://eprint.iacr.org/2023/271 (visited on 2025-03-02). Pre-published (cit. on p. 243).

[Gao+18] Xinwei Gao, Jintai Ding, Lin Li, et al. "Efficient Implementation of Password-Based Authenticated Key Exchange from RLWE and Post-Quantum TLS". In: *International Journal of Network Security* 20.5 (2018-09), pp. 923–930. ISSN: 1816-353X. DOI: 10.6633/IJNS.201809_20(5).14 (cit. on p. 116).

[Gao+17] Xinwei Gao, Lin Li, Jintai Ding, et al. "Fast Discretized Gaussian Sampling and Post-quantum TLS Ciphersuite". In: *Information Security Practice and Experience*. Vol. 10701. Springer International Publishing, 2017, pp. 551–565. ISBN: 978-3-319-72358-7. DOI: 10.1007/978-3-319-72359-4_33 (cit. on p. 116).

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. "KEM Combiners". In: *Public-Key Cryptography – PKC 2018*. Ed. by Michel Abdalla and Ricardo Dahab. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 190–218. ISBN: 978-3-319-76578-5. DOI: 10.1007/978-3-319-76578-5_7 (cit. on pp. 46, 47).

[Gid25] Craig Gidney. *How to Factor 2048 Bit RSA Integers with Less than a Million Noisy Qubits*. 2025-05-21. DOI: 10.48550/arXiv.2505.15917. arXiv: 2505.15917 [quant-ph]. Pre-published (cit. on p. 9).

[Gid+23] Craig Gidney, Michael Newman, Peter Brooks, et al. *Yoked Surface Codes*. 2023-12-07. DOI: 10.48550/arXiv.2312.04522. arXiv: 2312.04522 [quant-ph]. Pre-published (cit. on p. 7).

[Gir+22] Alexandre Augusto Giron, João Pedro Adami do Nascimento, Ricardo Custódio, et al. *Post-Quantum Hybrid KEMTLS Performance in Simulated and Real Network Environments*. 2022. URL: https://eprint.iacr.org/2022/1639 (visited on 2026-01-26). Pre-published (cit. on p. 115).

[GBP07] Tobias Gondrom, Ralf Brandner, and Ulrich Pordesch. *Evidence Record Syntax (ERS)*. Request for Comments RFC 4998. Internet Engineering Task Force, 2007-08. 32 pp. DOI: 10.17487/RFC4998 (cit. on p. 181).

[GW22] Ruben Gonzalez and Thom Wiggers. "KEMTLS vs. Post-quantum TLS: Performance on Embedded Systems". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Lejla Batina, Stjepan Picek, and Mainack Mondal. Lecture Notes in Computer Science. Springer Nature Switzerland, 2022, pp. 99–117. ISBN: 978-3-031-22829-2. DOI: 10.1007/978-3-031-22829-2_6 (cit. on p. 115).

[Goo26a] Google. *BoringSSL*. 2026. URL: https://boringssl.googlesource.com/boringssl (visited on 2026-03-05). Archived at https://web.archive.org/web/20260425122206/https://boringssl.googlesource.com/boringssl (cit. on p. 273).

[Goo26b] Google. *Cultivating a Robust and Efficient Quantum-Safe HTTPS*. Google Security Blog. 2026-02-27. URL: https://security.googleblog.com/2026/02/cultivating-robust-and-efficient.html (visited on 2026-03-05). Archived at https://web.archive.org/web/20260423073409/https://security.googleblog.com/2026/02/cultivating-robust-and-efficient.html (cit. on pp. 106, 201, 273).

[Goo23a] Google. *General-Purpose Machine Family for Compute Engine*. 2023. URL: https://cloud.google.com/compute/docs/general-purpose-machines (visited on 2023-12-01). Archived at https://web.archive.org/web/20251112143048/https://docs.cloud.google.com/compute/docs/general-purpose-machines (cit. on p. 119).

[Goo24] Google. *Meet Willow, Our State-of-the-Art Quantum Chip*. Google Blog. 2024-12-09. URL: https://blog.google/technology/research/google-willow-quantum-chip/ (visited on 2025-11-19). Archived at https://web.archive.org/web/20260228062517/https://blog.google/innovation-and-ai/technology/research/google-willow-quantum-chip/ (cit. on p. 7).

[Goo23b] Google. *Protecting Chrome Traffic with Hybrid Kyber KEM*. Chromium Blog. 2023-08-10. URL: https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html (visited on 2026-04-27). Archived at https://web.archive.org/web/20260326152045/https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html (cit. on pp. 112, 273).

[Goo26c] Google. *Quantum Frontiers May Be Closer than They Appear*. Google Blog. 2026-03-25. URL: https://blog.google/innovation-and-ai/technology/safety-security/cryptography-migration-timeline/ (visited on 2026-03-31). Archived at https://web.archive.org/web/20260411185904/https://blog.google/innovation-and-ai/technology/safety-security/cryptography-migration-timeline/ (cit. on p. 201).

[Goo22] Google. *Why Google Now Uses Post-Quantum Cryptography for Internal Comms*. Google Cloud Blog. 2022-11-19. URL: https://cloud.google.com/blog/products/identity-security/why-google-now-uses-post-quantum-cryptography-for-internal-comms (visited on 2026-04-27). Archived at https://web.archive.org/web/20250316053710/https://cloud.google.com/blog/products/identity-security/why-google-now-uses-post-quantum-cryptography-for-internal-comms (cit. on p. 112).

[Gro96] Lov K. Grover. *A Fast Quantum Mechanical Algorithm for Database Search*. 1996-11-19. DOI: 10.48550/arXiv.quant-ph/9605043. arXiv: quant-ph/9605043. Pre-published (cit. on pp. 10, 12, 32, 117, 118).

[GMP22] Paul Grubbs, Varun Maram, and Kenneth G. Paterson. "Anonymous, Robust Post-quantum Public Key Encryption". In: *Advances in Cryptology – EUROCRYPT 2022*. Ed. by Orr Dunkelman and Stefan Dziembowski. Springer International Publishing, 2022, pp. 402–432. ISBN: 978-3-031-07082-2. DOI: 10.1007/978-3-031-07082-2_15 (cit. on pp. 223, 225, 226, 262–264).

[GJK21] Yanqi Gu, Stanislaw Jarecki, and Hugo Krawczyk. "KHAPE: Asymmetric PAKE from Key-Hiding Key Exchange". In: *Advances in Cryptology – CRYPTO 2021*. Ed. by Tal Malkin and Chris Peikert. Springer International Publishing, 2021, pp. 701–730. ISBN: 978-3-030-84259-8. DOI: 10.1007/978-3-030-84259-8_24 (cit. on pp. 239, 263).

[GSV26] Felix Günther, Douglas Stebila, and Shannon Veitch. *Kemeleon Encodings*. Internet Draft draft-irtf-cfrg-kemeleon-01. Internet Engineering Task Force, 2026-01-16. 13 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-kemeleon (visited on 2026-02-17) (cit. on p. 239).

[GSV24] Felix Günther, Douglas Stebila, and Shannon Veitch. "Obfuscated Key Exchange". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Association for Computing Machinery, 2024-12-09, pp. 2385–2399. ISBN: 979-8-4007-0636-3. DOI: 10.1145/3658644.3690220 (cit. on pp. 239, 244, 263, 264).

[GL15] Chun Guo and Dongdai Lin. "Improved Domain Extender for the Ideal Cipher". In: *Cryptography and Communications* 7.4 (2015-12-01), pp. 509–533. ISSN: 1936-2455. DOI: 10.1007/s12095-015-0128-7 (cit. on p. 238).

[Gut20] Peter Gutmann. *Simple Certificate Enrolment Protocol*. Request for Comments RFC 8894. Internet Engineering Task Force, 2020-09. 42 pp. DOI: 10.17487/RFC8894 (cit. on p. 71).

[HvO22] Feng Hao and Paul C. van Oorschot. "SoK: Password-Authenticated Key Exchange – Theory, Practice, Standardization and Real-World Lessons". In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '22. Association for Computing Machinery, 2022-05-30, pp. 697–711. ISBN: 978-1-4503-9140-5. DOI: 10.1145/3488932.3523256 (cit. on p. 214).

[Hes+05] Peter Hesse, Matt Cooper, Yuriy A. Dzambasow, et al. *Internet X.509 Public Key Infrastructure: Certification Path Building*. Request for Comments RFC 4158. Internet Engineering Task Force, 2005-09. 81 pp. DOI: 10.17487/RFC4158 (cit. on p. 58).

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NTRU: A Ring-Based Public Key Cryptosystem". In: *Algorithmic Number Theory*. Ed. by Joe P. Buhler. Springer, 1998, pp. 267–288. ISBN: 978-3-540-69113-6. DOI: 10.1007/BFb0054868 (cit. on pp. 13, 34, 110).

[HSW25] Tim Hollebeek, Sophie Schmieg, and Bas Westerbaan. *Use of ML-DSA in TLS 1.3*. Internet Draft draft-ietf-tls-mldsa-01. Internet Engineering Task Force, 2025-09-26. 5 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-mldsa (visited on 2025-10-31) (cit. on p. 203).

[HC19] Andrew Hopkins and Matthew Campagna. *Post-Quantum TLS Now Supported in AWS KMS - AWS Security Blog*. AWS Security Blog. 2019-11-04. URL: https://aws.amazon.com/blogs/security/post-quantum-tls-now-supported-in-aws-kms/ (visited on 2023-12-01). Archived at https://web.archive.org/web/20260415040519/https://aws.amazon.com/blogs/security/post-quantum-tls-now-supported-in-aws-kms/ (cit. on p. 113).

[Hou09] Russ Housley. *Cryptographic Message Syntax (CMS)*. Request for Comments RFC 5652. Internet Engineering Task Force, 2009-09. 56 pp. DOI: 10.17487/RFC5652 (cit. on pp. 66, 68, 70, 71, 74, 85, 161, 164, 165, 167, 168, 172).

[HW23] James Howe and Bas Westerbaan. "Benchmarking and Analysing the NIST PQC Lattice-Based Signature Schemes Standards on the ARM Cortex M7". In: *Progress in Cryptology - AFRICACRYPT 2023*. Ed. by Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne. Springer Nature Switzerland, 2023, pp. 442–462. ISBN: 978-3-031-37679-5. DOI: 10.1007/978-3-031-37679-5_19 (cit. on p. 104).

[Hue+18] Aneas Huelsing, Denis Butin, Stefan-Lukas Gazdag, et al. *XMSS: eXtended Merkle Signature Scheme*. Request for Comments RFC 8391. Internet Engineering Task Force, 2018-05. 74 pp. DOI: 10.17487/RFC8391 (cit. on pp. 14, 35, 102).

[Hül+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, et al. *SPHINCS⁺*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 35, 103, 104).

[IBM25a] IBM. *How IBM Will Build the World's First Large-Scale, Fault-Tolerant Quantum Computer*. IBM Quantum Research Blog. 2025-06-10. URL: https://www.ibm.com/quantum/blog/large-scale-ftqc (visited on 2025-11-19). Archived at https://web.archive.org/web/20251231121937/https://www.ibm.com/quantum/blog/large-scale-ftqc (cit. on p. 8).

[IBM25b] IBM. *IBM Delivers New Quantum Processors, Software, and Algorithm Breakthroughs on Path to Advantage and Fault Tolerance*. 2025-11-12. URL: https://newsroom.ibm.com/2025-11-12-ibm-delivers-new-quantum-processors,-software,-and-algorithm-breakthroughs-on-path-to-advantage-and-fault-tolerance (visited on 2025-11-19) (cit. on p. 7).

[IBM25c] IBM. *IBM Quantum Computing | Hardware and Roadmap*. 2025. URL: https://www.ibm.com/quantum/hardware (visited on 2025-11-19). Archived at https://web.archive.org/web/20251221012439/https://www.ibm.com/quantum/hardware (cit. on p. 7).

[IBM24] IBM. *IBM Quantum Delivers on 2022 100x100 Performance Challenge*. IBM Quantum Research Blog. 2024-11-13. URL: https://www.ibm.com/quantum/blog/qdc-2024 (visited on 2025-11-19). Archived at https://web.archive.org/web/20260108073433/https://www.ibm.com/quantum/blog/qdc-2024 (cit. on p. 7).

[Ins25] Institute of Commercial Cryptography Standards. *Next-Generation Commercial Cryptographic Algorithms Program (NGCC)*. 2025. URL: https://www.niccs.org.cn/en/ (visited on 2025-11-25). Archived at https://web.archive.org/web/20260219212636/https://www.niccs.org.cn/en/ (cit. on p. 14).

[ISA22] ISARA Corporation. *ISARA Dedicates Four Hybrid Certificate Patents to the Public, Easing Path to Quantum-Safe Security*. ISARA Corporation. 2022-10-26. URL: https://www.isara.com/company/newsroom/isara-dedicates-four-hybrid-certificate-patents-to-the-public.html (visited on 2025-10-06). Archived at https://web.archive.org/web/20260123072335/https://www.isara.com/company/newsroom/isara-dedicates-four-hybrid-certificate-patents-to-the-public.html (cit. on p. 91).

[ITU19] ITU-T. *Recommendation ITU-T X.509 | Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks*. 2019-10. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201910-I!!PDF-E&type=items (cit. on p. 90).

[ITU21a] ITU-T. *Recommendation ITU-T X.680 | Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. 2021-02. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.680-202102-I!!PDF-E&type=items (cit. on pp. 53, 164).

[ITU21b] ITU-T. *Recommendation ITU-T X.690 | Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. 2021-02. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.690-202102-I!!PDF-E&type=items (cit. on pp. 53, 164).

[JSS12] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky. "Bleichenbacher's Attack Strikes Again: Breaking PKCS#1 v1.5 in XML Encryption". In: *Computer Security – ESORICS 2012*. Ed. by Sara Foresti, Moti Yung, and Fabio Martinelli. Lecture Notes in Computer Science. Springer, 2012, pp. 752–769. ISBN: 978-3-642-33167-1. DOI: 10.1007/978-3-642-33167-1_43 (cit. on p. 135).

[JS11] Tibor Jager and Juraj Somorovsky. "How to Break XML Encryption". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. Association for Computing Machinery, 2011-10-17, pp. 413–422. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046756 (cit. on pp. 135, 273).

[Jao+22] David Jao, Reza Azarderakhsh, Matthew Campagna, et al. *SIKE*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 36, 40, 110–113, 115).

[Jon15] Michael B. Jones. *JSON Web Key (JWK)*. Request for Comments RFC 7517. Internet Engineering Task Force, 2015-05. 40 pp. DOI: 10.17487/RFC7517 (cit. on pp. 175, 176).

[JBS15a] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Signature (JWS)*. Request for Comments RFC 7515. Internet Engineering Task Force, 2015-05. 59 pp. DOI: 10.17487/RFC7515 (cit. on pp. 72, 175, 178–180).

[JBS15b] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. Request for Comments RFC 7519. Internet Engineering Task Force, 2015-05. 30 pp. DOI: 10.17487/RFC7519 (cit. on p. 180).

[JK03] Jakob Jonsson and Burt Kaliski. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. Request for Comments RFC 3447. Internet Engineering Task Force, 2003-02. 72 pp. DOI: 10.17487/RFC3447 (cit. on pp. 141, 142, 148, 152, 159).

[Jos06] Simon Josefsson. *The Base16, Base32, and Base64 Data Encodings*. Request for Comments RFC 4648. Internet Engineering Task Force, 2006-10. 18 pp. DOI: 10.17487/RFC4648 (cit. on p. 177).

[KZ20] Daniel Kales and Greg Zaverucha. “An Attack on Some Signature Schemes Constructed from Five-Pass Identification Schemes”. In: *Cryptology and Network Security*. Ed. by Stephan Krenn, Haya Shulman, and Serge Vaudenay. Lecture Notes in Computer Science. Springer International Publishing, 2020, pp. 3–22. ISBN: 978-3-030-65411-5. DOI: 10.1007/978-3-030-65411-5_1 (cit. on p. 104).

[KS19] Panos Kampanakis and Dimitrios Sikeridis. *Two PQ Signature Use-cases: Non-issues, Challenges and Potential Solutions*. 2019. URL: https://eprint.iacr.org/2019/1276 (cit. on pp. 102–104).

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. 3rd ed. Chapman and Hall/CRC, 2020-12-20. 648 pp. ISBN: 978-1-351-13303-6. DOI: 10.1201/9781351133036 (cit. on pp. 5, 31).

[KK18] Franziskus Kiefer and Kris Kwiatkowski. *Hybrid ECDHE-SIDH Key Exchange for TLS*. Internet-Draft. Internet Engineering Task Force, 2018-11. URL: https://datatracker.ietf.org/doc/draft-kiefer-tls-ecdhe-sidh/00/ (cit. on p. 111).

[Kpq25] KpqC. *Selected Algorithms from the KpqC Competition Round 2*. 2025-01-16. URL: https://www.kpqc.or.kr/competition_02.html (visited on 2025-11-25). Archived at https://web.archive.org/web/20260310034155/https://www.kpqc.or.kr/competition_02.html (cit. on p. 14).

[Kra10] Hugo Krawczyk. “Cryptographic Extraction and Key Derivation: The HKDF Scheme”. In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Lecture Notes in Computer Science. Springer, 2010, pp. 631–648. ISBN: 978-3-642-14623-7. DOI: 10.1007/978-3-642-14623-7_34 (cit. on p. 26).

[KW16] Hugo Krawczyk and Hoeteck Wee. “The OPTLS Protocol and TLS 1.3”. In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2016, pp. 81–96. DOI: 10.1109/EuroSP.2016.18 (cit. on pp. 114, 116).

[Kup+15] Dennis Kupser, Christian Mainka, Jörg Schwenk, et al. “How to Break XML Encryption – Automatically”. In: *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. USENIX Association, 2015-08. URL: https://www.usenix.org/conference/woot15/workshop-program/presentation/kupser (cit. on p. 135).

[Kwi+25] Kris Kwiatkowski, Panos Kampanakis, Bas Westerbaan, et al. *Post-Quantum Hybrid ECDHE-MLKEM Key Agreement for TLSv1.3*. Internet Draft draft-ietf-tls-ecdhe-mlkem-01. Internet Engineering Task Force, 2025-09-29. 10 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-ecdhe-mlkem (visited on 2025-10-31) (cit. on pp. 203, 207).

[KV19] Kris Kwiatkowski and Luke Valenta. *The TLS Post-Quantum Experiment*. The Cloudflare Blog. 2019-10-30. URL: https://blog.cloudflare.com/the-tls-post-quantum-experiment/ (visited on 2026-04-27). Archived at https://web.archive.org/web/20260416014040/https://blog.cloudflare.com/the-tls-post-quantum-experiment/ (cit. on p. 112).

[Lan16] Adam Langley. *CECPQ1 Results*. ImperialViolet. 2016-11-28. URL: https://www.imperialviolet.org/2016/11/28/cecpq1.html (visited on 2026-01-01). Archived at https://web.archive.org/web/20260404024001/https://www.imperialviolet.org/2016/11/28/cecpq1.html (cit. on p. 112).

[Lan18] Adam Langley. *CECPQ2*. ImperialViolet. 2018-12-12. URL: https://www.imperialviolet.org/2018/12/12/cecpq2.html (visited on 2026-01-01). Archived at https://web.archive.org/web/20260421022619/https://www.imperialviolet.org/2018/12/12/cecpq2.html (cit. on p. 112).

[LS15] Adeline Langlois and Damien Stehlé. "Worst-Case to Average-Case Reductions for Module Lattices". In: *Designs, Codes and Cryptography* 75.3 (2015-06-01), pp. 565–599. ISSN: 1573-7586. DOI: 10.1007/s10623-014-9938-4 (cit. on p. 34).

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. Request for Comments RFC 6962. Internet Engineering Task Force, 2013-06. 27 pp. DOI: 10.17487/RFC6962 (cit. on p. 81).

[Leg25a] Legion of the Bouncy Castle. *Bouncy Castle Releases Java 1.81 and C# .NET 2.6.1*. Bouncycastle. 2025-06-09. URL: https://www.bouncycastle.org/resources/bouncy-castle-releases-java-1-81-and-c-net-2-6-1/ (visited on 2026-03-05). Archived at https://web.archive.org/web/20250810063615/https://www.bouncycastle.org/resources/bouncy-castle-releases-java-1-81-and-c-net-2-6-1/ (cit. on p. 203).

[Leg25b] Legion of the Bouncy Castle Inc. *Bouncy Castle Cryptography APIs*. 2025. URL: https://www.bouncycastle.org/ (cit. on pp. 19, 136, 208).

[LL93] Arjen K. Lenstra and Hendrik W. Lenstra, eds. *The Development of The Number Field Sieve*. Vol. 1554. Lecture Notes in Mathematics. Springer, 1993. ISBN: 978-3-540-57013-4. DOI: 10.1007/BFb0091534 (cit. on p. 5).

[Let25] Let's Encrypt. *Announcing Six Day and IP Address Certificate Options in 2025*. 2025-01-16. URL: https://letsencrypt.org/2025/01/16/6-day-and-ip-certs/ (visited on 2025-06-24). Archived at https://web.archive.org/web/20260106090215/https://letsencrypt.org/2025/01/16/6-day-and-ip-certs/ (cit. on p. 64).

[LIM21] Fukang Liu, Takanori Isobe, and Willi Meier. "Cryptanalysis of Full LowMC and LowMC-M with Algebraic Techniques". In: *Advances in Cryptology – CRYPTO 2021*. Ed. by Tal Malkin and Chris Peikert. Springer International Publishing, 2021, pp. 368–401. ISBN: 978-3-030-84252-9. DOI: 10.1007/978-3-030-84252-9_13 (cit. on p. 103).

[Liu+22] Fukang Liu, Santanu Sarkar, Gaoli Wang, et al. "Algebraic Meet-in-the-Middle Attack on LowMC". In: *Advances in Cryptology – ASIACRYPT 2022*. Ed. by Shweta Agrawal and Dongdai Lin. Springer Nature Switzerland, 2022, pp. 225–255. ISBN: 978-3-031-22963-3. DOI: 10.1007/978-3-031-22963-3_8 (cit. on p. 103).

[LY06] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Connection Protocol*. Request for Comments RFC 4254. Internet Engineering Task Force, 2006-01. 24 pp. DOI: 10.17487/RFC4254 (cit. on p. 15).

[Lyu+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, et al. *CRYSTALS-DILITHIUM*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 34, 103).

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Springer, 2010, pp. 1–23. ISBN: 978-3-642-13190-5. DOI: 10.1007/978-3-642-13190-5_1 (cit. on p. 34).

[MX23] Varun Maram and Keita Xagawa. "Post-Quantum Anonymity of Kyber". In: *Public-Key Cryptography – PKC 2023*. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Springer Nature Switzerland, 2023, pp. 3–35. ISBN: 978-3-031-31368-4. DOI: 10.1007/978-3-031-31368-4_1 (cit. on p. 262).

[MS22] Dominik Marchsreiter and Johanna Sepúlveda. "Hybrid Post-Quantum Enhanced TLS 1.3 on Embedded Devices". In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. 2022 25th Euromicro Conference on Digital System Design (DSD). 2022-08, pp. 905–912. DOI: 10.1109/DSD57027.2022.00127 (cit. on pp. 103, 105, 107, 109, 110).

[MP16] Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. Revision 1. 2016-11-04. URL: https://signal.org/docs/specifications/x3dh/x3dh.pdf. Archived at https://web.archive.org/web/20260302150753/https://signal.org/docs/specifications/x3dh/x3dh.pdf (cit. on p. 4).

[Mas+25] Jake Massimo, Panos Kampanakis, Sean Turner, et al. *Internet X.509 Public Key Infrastructure – Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)*. Request for Comments RFC 9881. Internet Engineering Task Force, 2025-10. 91 pp. DOI: 10.17487/RFC9881 (cit. on p. 205).

[Mat26] Matrix Foundation. *Matrix.Org | An Open Network for Secure, Decentralised Communication*. 2026. URL: https://matrix.org/ (visited on 2026-04-27) (cit. on p. 15).

[MST21] John Preuß Mattsson, Ben Smeets, and Erik Thormarker. "Quantum Technology and Its Impact on Security in Mobile Networks". In: *Ericsson Technology Review* 2021.12 (2021), pp. 2–12. DOI: 10.23919/ETR.2021.9904724 (cit. on p. 117).

[McE78] R. J. McEliece. *A Public-Key Cryptosystem Based On Algebraic Coding Theory*. JPL Deep Space Network Progress Report 42–44. NASA, 1978. URL: https://ipnpr.jpl.nasa.gov/progress_report/42-44/44N.PDF (visited on 2025-11-25) (cit. on pp. 13, 34).

[MCF19] David McGrew, Michael Curcio, and Scott Fluhrer. *Leighton-Micali Hash-Based Signatures*. Request for Comments RFC 8554. Internet Engineering Task Force, 2019-04. 61 pp. DOI: 10.17487/RFC8554 (cit. on pp. 14, 35, 102).

[McK25] McKinsey & Company. *Quantum Technology Investment Hits a 'Magic Moment'*. 2025-10-27. URL: https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/tech-forward/quantum-technology-investment-hits-a-magic-moment (visited on 2026-03-05). Archived at https://web.archive.org/web/20260205053817/https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/tech-forward/quantum-technology-investment-hits-a-magic-moment (cit. on p. 273).

[Mon+05] Tero Mononen, Tomi Kause, Stephen Farrell, et al. *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. Request for Comments RFC 4210. Internet Engineering Task Force, 2005-09. 95 pp. DOI: 10.17487/RFC4210 (cit. on pp. 52, 65, 68, 69).

[Moo+20] Dustin Moody, Gorjan Alagic, Daniel C Apon, et al. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST IR 8309. National Institute of Standards and Technology, 2020-07, NIST IR 8309. DOI: 10.6028/NIST.IR.8309 (cit. on pp. 13, 103, 104, 109).

[Mos15] Michele Mosca. “Cybersecurity in a Quantum World: Will We Be Ready?” (NIST Workshop on Cybersecurity in a Post-Quantum World). 2015-04-03. URL: https://csrc.nist.gov/csrc/media/events/workshop-on-cybersecurity-in-a-post-quantum-world/documents/presentations/session8-mosca-michele.pdf (visited on 2026-04-27). Archived at https://web.archive.org/web/20260303060626/https://csrc.nist.gov/csrc/media/events/workshop-on-cybersecurity-in-a-post-quantum-world/documents/presentations/session8-mosca-michele.pdf (cit. on p. 12).

[MP26] Michele Mosca and Marco Piani. “Quantum Threat Timeline Report 2025”. In: *Global Risk Institute, Toronto, ON* (2026-09-03). URL: https://globalriskinstitute.org/mp-files/pdf-quantum-threat-timeline-report-2025.pdf/ (visited on 2026-04-27). Archived at https://web.archive.org/web/20260427133401/https://globalriskinstitute.org/mp-files/pdf-quantum-threat-timeline-report-2025.pdf/ (cit. on p. 8).

[Moz25] Mozilla Foundation. *CA/Included Certificates*. 2025. URL: https://wiki.mozilla.org/CA/Included_Certificates (visited on 2025-06-05). Archived at https://web.archive.org/web/20260305033205/https://wiki.mozilla.org/CA/Included_Certificates (cit. on p. 51).

[Moz24] Mozilla Foundation. *Firefox 132.0, See All New Features, Updates and Fixes*. 2024-10-29. URL: https://www.mozilla.org/en-US/firefox/132.0/releasenotes/ (visited on 2025-06-20). Archived at https://web.archive.org/web/20260424001032/https://www.firefox.com/en-US/firefox/132.0/releasenotes/ (cit. on p. 273).

[MO24] Johannes Müller and Jan Oupický. “Post-Quantum XML and SAML Single Sign-On”. In: *Proceedings on Privacy Enhancing Technologies* 2024.4 (2024), pp. 525–543. DOI: 10.56553/popets-2024-0128 (cit. on pp. 18–20, 128).

[MS08] Michael Myers and Jim Schaad. *Certificate Management over CMS (CMC)*. Request for Comments RFC 5272. Internet Engineering Task Force, 2008-06. 83 pp. DOI: 10.17487/RFC5272 (cit. on p. 71).

[Nae+20] Michael Naehrig, Erdem Alkim, Joppe Bos, et al. *FrodoKEM*. National Institute of Standards and Technology, 2020 (cit. on p. 109).

[Nat23] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*. Federal Information Processing Standard (FIPS) 186-5. U.S. Department of Commerce, 2023-02-03. DOI: 10.6028/NIST.FIPS.186-5 (cit. on p. 159).

[NCS25a] NCSC UK. *Quantum Networking Technologies*. 2025-08-05. URL: https://www.ncsc.gov.uk/whitepaper/quantum-networking-technologies (visited on 2025-12-04). Archived at https://web.archive.org/web/20260315192623/https://www.ncsc.gov.uk/whitepaper/quantum-networking-technologies (cit. on p. 15).

[NCS25b] NCSC UK. *Timelines for Migration to Post-Quantum Cryptography*. 2025-03-20. URL: https://www.ncsc.gov.uk/guidance/pqc-migration-timelines (visited on 2025-11-05). Archived at https://web.archive.org/web/20260414040315/https://www.ncsc.gov.uk/guidance/pqc-migration-timelines (cit. on p. 201).

[NIS22] NIST. *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*. 2022-10. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf (visited on 2025-07-17). Archived at https://web.archive.org/web/20260420162944/https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf (cit. on pp. 14, 273).

[NIS24a] NIST. *Module-Lattice-Based Digital Signature Standard*. Federal Information Processing Standard (FIPS) 204. U.S. Department of Commerce, 2024-08-13. DOI: 10.6028/NIST.FIPS.204 (cit. on pp. 13, 14, 34, 37, 103, 201, 273).

[NIS24b] NIST. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Federal Information Processing Standard (FIPS) 203. U.S. Department of Commerce, 2024-08-13. DOI: 10.6028/NIST.FIPS.203 (cit. on pp. 13, 14, 34, 36, 105, 109, 201, 219, 223, 261–263, 273).

[NIS15a] NIST. *Secure Hash Standard (SHS)*. Federal Information Processing Standard (FIPS) 180-4. U.S. Department of Commerce, 2015-08-04. DOI: 10.6028/NIST.FIPS.180-4 (cit. on pp. 11, 12).

[NIS15b] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Federal Information Processing Standard (FIPS) 202. U.S. Department of Commerce, 2015-08-04. DOI: 10.6028/NIST.FIPS.202 (cit. on pp. 12, 192).

[NIS24c] NIST. *Stateless Hash-Based Digital Signature Standard*. Federal Information Processing Standard (FIPS) 205. U.S. Department of Commerce, 2024-08-13. DOI: 10.6028/NIST.FIPS.205 (cit. on pp. 13, 14, 35, 37, 102, 201, 273).

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf (visited on 2025-02-26). Archived at https://web.archive.org/web/20260420051706/https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf (cit. on pp. 32, 117, 118).

[NSA22] NSA. *Commercial National Security Algorithm Suite 2.0*. 2022-09. URL: https://media.defense.gov/2025/May/30/2003728741/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS.PDF (visited on 2026-04-13). Archived at https://web.archive.org/web/20260323205420/https://media.defense.gov/2025/May/30/2003728741/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS.PDF (cit. on pp. 8, 12, 117, 201).

[NSA20] NSA. *NSA Cybersecurity Perspectives on Quantum Key Distribution and Quantum Cryptography*. 2020-10-26. URL: http://www.nsa.gov/Press-Room/News-Highlights/Article/Article/2394053/nsa-cybersecurity-perspectives-on-quantum-key-distribution-and-quantum-cryptogr/ (visited on 2025-12-03). Archived at https://web.archive.org/web/20251006162517/https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/2394053/nsa-cybersecurity-perspectives-on-quantum-key-distribution-and-quantum-cryptogr/ (cit. on p. 15).

[NSA24] NSA. *The Commercial National Security Algorithm Suite 2.0 and Quantum Computing FAQ*. 2024-12. URL: https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF. Archived at https://web.archive.org/web/20260416073918/https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF (cit. on pp. 8, 43).

[OAS12] OASIS. *SAML V2.0 Specification*. SAML V2.0 specification. 2012. URL: https://web.archive.org/web/20230118004327/https://wiki.oasis-open.org/security/FrontPage (visited on 2022-12-07) (cit. on pp. 15, 143).

[Ope23a] Open Quantum Safe project. *OQS Liboqs Library*. 2023. URL: https://github.com/open-quantum-safe/liboqs (cit. on pp. 119, 121).

[Ope23b] Open Quantum Safe project. *OQS OpenSSL 3 Provider*. 2023. URL: https://github.com/open-quantum-safe/oqs-provider (cit. on p. 119).

[Ope23c] Open Quantum Safe project. *OQS OpenSSL Fork*. 2023. URL: https://github.com/open-quantum-safe/openssl (cit. on pp. 119, 203).

[Ope23d] OpenSSL Foundation. *OpenSSL*. 2023. URL: https://openssl-library.org/ (cit. on p. 119).

[Ope25] OpenSSL Foundation. *The Features of 3.5: Post-quantum Cryptography*. 2025-04-22. URL: https://openssl.foundation/news/the-features-of-3-5-post-quantum-cryptography (visited on 2026-03-05). Archived at https://web.archive.org/web/20260212074658/https://openssl.foundation/news/the-features-of-3-5-post-quantum-cryptography (cit. on pp. 203, 273).

[OC25] Mike Ounsworth and Deirdre Connolly. "IETF Perspective on SP 800-227" (NIST Workshop on SP 800-227). 2025-02-25. URL: https://csrc.nist.gov/csrc/media/Presentations/2025/ietf-perspective-on-sp-800-227/images-media/ietf-perspective-on-sp800-227.pdf. Archived at https://web.archive.org/web/20260427140628/https://csrc.nist.gov/csrc/media/Presentations/2025/ietf-perspective-on-sp-800-227/images-media/ietf-perspective-on-sp800-227.pdf (cit. on p. 85).

[Oun+25a] Mike Ounsworth, John Gray, Massimiliano Pala, et al. *Composite ML-DSA for Use in X.509 Public Key Infrastructure*. Internet Draft draft-ietf-lamps-pq-composite-sigs-14. Internet Engineering Task Force, 2025-10-31. 204 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-sigs/14 (visited on 2025-10-31) (cit. on pp. 19, 44, 45, 139, 186, 205).

[Oun+25b] Mike Ounsworth, John Gray, Massimiliano Pala, et al. *Composite ML-KEM for Use in X.509 Public Key Infrastructure*. Internet Draft draft-ietf-lamps-pq-composite-kem-12. Internet Engineering Task Force, 2025-06-16. 132 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-kem/12/ (visited on 2026-01-08) (cit. on p. 47).

[OWK24] Mike Ounsworth, Aron Wussler, and Stavros Kousidis. *Combiner Function for Hybrid Key Encapsulation Mechanisms (Hybrid KEMs)*. Internet-draft draft-ounsworth-cfrg-kem-combiners-05. Internet Engineering Task Force / Internet Engineering Task Force, 2024-01-31. 14 pp. URL: https://datatracker.ietf.org/doc/draft-ounsworth-cfrg-kem-combiners/05/ (cit. on p. 47).

[Oup23] Jan Oupický. *Composite Post-Quantum Signatures Experimental Implementation, BouncyCastle*. 2023-12-10. URL: https://github.com/bcgit/bc-java/pull/1546 (visited on 2025-11-27). Archived at https://web.archive.org/web/20260427140754/https://github.com/bcgit/bc-java/pull/1546 (cit. on p. 19).

[Oup24a] Jan Oupický. *Possible Inconsistencies in the Provided Samples. · Issue #130 · EntrustCorporation/Draft-Ounsworth-Composite-Sigs*. GitHub. 2024-02-08. URL: https://github.com/EntrustCorporation/draft-ounsworth-composite-sigs/issues/130 (visited on 2025-11-27). Archived at https://web.archive.org/web/20260427141012/https://github.com/EntrustCorporation/draft-ounsworth-composite-sigs/issues/130 (cit. on p. 19).

[Oup25a] Jan Oupický. *Post-Quantum Advanced Electronic Signatures Playground*. Zenodo, 2025-10. DOI: 10.5281/zenodo.18800972 (cit. on pp. 79, 92, 191, 205, 208, 272).

[Oup25b] Jan Oupický. *Post-Quantum PAPKE Implementation and Benchmarks*. Zenodo, 2025-10. DOI: 10.5281/zenodo.18800723 (cit. on p. 263).

[Oup24b] Jan Oupický. *Post-Quantum TLS Benchmarks*. Zenodo, 2024-11. DOI: 10.5281/zenodo.18800565 (cit. on p. 119).

[Oup24c] Jan Oupický. *Post-Quantum XML and SAML Implementation and Benchmarks*. Zenodo, 2024-06. DOI: 10.5281/zenodo.18800823 (cit. on pp. 136, 204, 209, 272).

[PST20] Christian Paquin, Douglas Stebila, and Goutam Tamvada. "Benchmarking Post-quantum Cryptography in TLS". In: *Post-Quantum Cryptography*. Ed. by Jintai Ding and Jean-Pierre Tillich. Lecture Notes in Computer Science. Springer International Publishing, 2020, pp. 72–91. ISBN: 978-3-030-44223-1. DOI: 10.1007/978-3-030-44223-1_5 (cit. on pp. 103, 104, 111, 112).

[PG97] Jacques Patarin and Louis Goubin. "Trapdoor One-Way Permutations and Multivariate Polynomials". In: *Information and Communications Security*. Ed. by Yongfei Han, Tatsuaki Okamoto, and Sihan Qing. Springer, 1997, pp. 356–368. ISBN: 978-3-540-69628-5. DOI: 10.1007/BFb0028491 (cit. on p. 35).

[PPS04] Kenneth G. Paterson, Fred Piper, and Ruediger Schack. *Quantum Cryptography: A Practical Information Security Perspective*. 2004. URL: https://eprint.iacr.org/2004/156 (visited on 2025-03-07). Pre-published (cit. on p. 15).

[Pau+22] Sebastian Paul, Yulia Kuzovkova, Norman Lahr, 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. Association for Computing Machinery, 2022-05, pp. 727–740. ISBN: 978-1-4503-9140-5. DOI: 10.1145/3488932.3497755 (cit. on pp. 102–104, 107, 109).

[PSS21] Sebastian Paul, Felix Schick, and Jan Seedorf. "TPM-Based Post-Quantum Cryptography: A Case Study on Quantum-Resistant and Mutually Authenticated TLS for IoT Environments". In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES 21. Association for Computing Machinery, 2021-08, pp. 1–10. ISBN: 978-1-4503-9051-4. DOI: 10.1145/3465481.3465747 (cit. on pp. 102, 107, 109).

[Pei20] Chris Peikert. “He Gives C-Sieves on the CSIDH”. In: *Advances in Cryptology – EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Springer International Publishing, 2020, pp. 463–492. ISBN: 978-3-030-45724-2. DOI: 10.1007/978-3-030-45724-2_16 (cit. on pp. 36, 116).

[Pol75] J. M. Pollard. “A Monte Carlo Method for Factorization”. In: *BIT Numerical Mathematics* 15.3 (1975-09-01), pp. 331–334. ISSN: 1572-9125. DOI: 10.1007/BF01933667 (cit. on p. 6).

[Pöp+19] Thomas Pöppelmann, Erdem Alkim, Roberto Avanzi, et al. *NewHope*. National Institute of Standards and Technology, 2019 (cit. on pp. 107, 109).

[Por19] Thomas Pornin. *New Efficient, Constant-Time Implementations of Falcon*. 2019. URL: https://eprint.iacr.org/2019/893 (visited on 2026-01-26). Pre-published (cit. on p. 103).

[PQC06] PQCrypto. *PQCrypto 2006: International Workshop on Post-Quantum Cryptography*. 2006. URL: https://postquantum.cr.yp.to/ (visited on 2025-11-24). Archived at https://web.archive.org/web/20260308233153/http://postquantum.cr.yp.to/ (cit. on p. 13).

[Pre+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, et al. *FALCON*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 14, 37, 103).

[PYH13] Max Pritikin, Peter E. Yee, and Dan Harkins. *Enrollment over Secure Transport*. Request for Comments RFC 7030. Internet Engineering Task Force, 2013-10. 53 pp. DOI: 10.17487/RFC7030 (cit. on p. 71).

[Qua24] Quantinuum. *Quantinuum Partners with Microsoft in New Phase of Reliable Quantum Computing with Breakthrough Demonstration of Reliable Logical Qubits*. 2024-04-03. URL: https://www.quantinuum.com/press-releases/quantinuum-and-microsoft-announce-new-era-in-quantum-computing-with-breakthrough-demonstration-of-reliable-qubits (visited on 2025-11-19). Archived at https://web.archive.org/web/20260216021517/https://www.quantinuum.com/press-releases/quantinuum-and-microsoft-announce-new-era-in-quantum-computing-with-breakthrough-demonstration-of-reliable-qubits (cit. on p. 7).

[Rab79] Michael O. Rabin. *Digitalized Signatures and Public Key Functions as Intractable as Factorization*. MIT/LCS/TR-212. MIT, 1979-01. URL: https://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-212.pdf (visited on 2025-11-18). Archived at https://web.archive.org/web/20260423013152/https://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-212.pdf (cit. on p. 5).

[Ral24] Ralph Simpson. “Cracking the Enigma: The Secret Battlefield of WW2”. 2024-09-20. URL: https://ciphermachines.com/slides/enigma.pdf (visited on 2025-11-07). Archived at https://web.archive.org/web/20251107161118/https://ciphermachines.com/slides/enigma.pdf (cit. on p. 2).

[Ran25] Lars Ran. *Wedges, Oil, and Vinegar – An Analysis of UOV in Characteristic 2*. 2025. URL: https://eprint.iacr.org/2025/1143 (visited on 2025-12-08). Pre-published (cit. on p. 35).

[Reg05] Oded Regev. “On Lattices, Learning with Errors, Random Linear Codes, and Cryptography”. In: *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*. STOC ’05. Association for Computing Machinery, 2005-05-22, pp. 84–93. ISBN: 978-1-58113-960-0. DOI: 10.1145/1060590.1060603 (cit. on p. 33).

[Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Request for Comments RFC 8446. Internet Engineering Task Force, 2018-08. 160 pp. DOI: 10.17487/RFC8446 (cit. on pp. 4, 15, 50, 51, 98, 99, 110, 117, 118).

[RD08] Eric Rescorla and Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. Request for Comments RFC 5246. Internet Engineering Task Force, 2008-08. 104 pp. DOI: 10.17487/RFC5246 (cit. on pp. 98, 101, 107).

[Ric+24] Sara Ricci, Vladyslav Shapoval, Petr Dzurenda, et al. “Lattice-Based Multisignature Optimization for RAM Constrained Devices”. In: *Proceedings of the 19th International Conference on Availability, Reliability and Security*. ARES ’24. Association for Computing Machinery, 2024-07-30, pp. 1–10. ISBN: 979-8-4007-1718-5. DOI: 10.1145/3664476.3670461 (cit. on p. 20).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Commun. ACM* 21.2 (1978-02-01), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342 (cit. on p. 3).

[Saa23] Markku-Juhani O Saarinen. “Introduction to Side-Channel Security of NIST PQC Standards” (NIST PQC Seminar). 2023-04-04. URL: https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/pqc-seminars/presentations/2-side-channel-security-saarinen-04042023.pdf (visited on 2025-11-14). Archived at https://web.archive.org/web/20260312051602/https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/pqc-seminars/presentations/2-side-channel-security-saarinen-04042023.pdf (cit. on p. 34).

[SXY17] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. *Tightly-Secure Key-Encapsulation Mechanism in the Quantum Random Oracle Model*. 2017. URL: https://eprint.iacr.org/2017/1005 (visited on 2025-09-18). Pre-published (cit. on p. 219).

[SRG25] Ben Salter, Adam Raine, and Daniel Van Geest. *Use of the ML-DSA Signature Algorithm in the Cryptographic Message Syntax (CMS)*. Request for Comments RFC 9882. Internet Engineering Task Force, 2025-10. 25 pp. DOI: 10.17487/RFC9882 (cit. on p. 205).

[Sam+19] Simona Samardjiska, Ming-Shing Chen, Andreas Hulsing, et al. *MQDSS*. National Institute of Standards and Technology, 2019 (cit. on p. 104).

[San+13] Stefan Santesson, Michael Myers, Rich Ankney, et al. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. Request for Comments RFC 6960. Internet Engineering Task Force, 2013-06. 41 pp. DOI: 10.17487/RFC6960 (cit. on pp. 15, 52, 61).

[SGJ23] Bruno Freitas Dos Santos, Yanqi Gu, and Stanislaw Jarecki. “Randomized Half-Ideal Cipher on Groups with Applications to UC (a)PAKE”. In: *Advances in Cryptology – EUROCRYPT 2023*. Ed. by Carmit Hazay and Martijn Stam. Springer Nature Switzerland, 2023, pp. 128–156. ISBN: 978-3-031-30589-4. DOI: 10.1007/978-3-031-30589-4_5 (cit. on pp. 234, 240).

[Sch05] Jim Schaad. *Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)*. Request for Comments RFC 4211. Internet Engineering Task Force, 2005-09. 40 pp. DOI: 10.17487/RFC4211 (cit. on pp. 65, 66, 68).

[SS17] John M. Schanck and Douglas Stebila. *A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret*. Internet-Draft. Internet Engineering Task Force, 2017-04. URL: https://datatracker.ietf.org/doc/draft-schanck-tls-additional-keyshare/00/ (cit. on p. 112).

[SWZ16] John M. Schanck, William Whyte, and Zhenfei Zhang. *Quantum-Safe Hybrid (QSH) Ciphersuite for Transport Layer Security (TLS) Version 1.2*. Internet-Draft. Internet Engineering Task Force, 2016. URL: https://datatracker.ietf.org/doc/draft-whyte-qsh-tls12/02/ (cit. on p. 113).

[Sch+22a] Frederico Schardong, Alexandre Augusto Giron, Fernanda Larisa Müller, et al. "Post-Quantum Electronic Identity: Adapting OpenID Connect and OAuth 2.0 to the Post-Quantum Era". In: *Cryptology and Network Security*. Ed. by Alastair R. Beresford, Arpita Patra, and Emanuele Bellini. Springer International Publishing, 2022, pp. 371–390. ISBN: 978-3-031-20974-1. DOI: 10.1007/978-3-031-20974-1_20 (cit. on p. 273).

[Sch+22b] Peter Schwabe, Roberto Avanzi, Joppe Bos, et al. *CRYSTALS-KYBER*. National Institute of Standards and Technology, 2022 (cit. on pp. 13, 34, 105, 107, 109, 110).

[SSW21] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys". In: *Computer Security – ESORICS 2021*. Ed. by Elisa Bertino, Haya Shulman, and Michael Waidner. Lecture Notes in Computer Science. Springer International Publishing, 2021, pp. 3–22. ISBN: 978-3-030-88418-5. DOI: 10.1007/978-3-030-88418-5_1 (cit. on p. 115).

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Post-Quantum TLS Without Handshake Signatures". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Association for Computing Machinery, 2020-11-02, pp. 1461–1480. ISBN: 978-1-4503-7089-9. DOI: 10.1145/3372297.3423350 (cit. on pp. 102–104, 107, 109–111, 113, 115).

[She+25] Rifaat Shekh-Yusef, Hannes Tschofenig, Mike Ounsworth, et al. *Post-Quantum Traditional (PQ/T) Hybrid Authentication with Dual Certificates in TLS 1.3*. Internet Draft draft-yusef-tls-pqt-dual-certs-01. Internet Engineering Task Force, 2025-12-17. 27 pp. URL: https://datatracker.ietf.org/doc/draft-yusef-tls-pqt-dual-certs (visited on 2026-03-09) (cit. on pp. 87, 105, 203).

[Shi25] Shibboleth Project. *OpenSAML*. 2025. URL: https://shibboleth.atlassian.net/wiki/spaces/OSAML/overview. Archived at https://web.archive.org/web/20251118135310/https://shibboleth.atlassian.net/wiki/spaces/OSAML/overview (cit. on pp. 19, 136).

[Sho94] P.W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. Proceedings 35th Annual Symposium on Foundations of Computer Science. 1994-11, pp. 124–134. DOI: 10.1109/SFCS.1994.365700 (cit. on pp. 8, 10, 12).

[Sig16] Signal. *Safety Number Updates*. Signal Blog. 2016-11-17. URL: https://signal.org/blog/safety-number-updates/ (visited on 2026-02-06). Archived at https://web.archive.org/web/20260425122342/https://signal.org/blog/safety-number-updates/ (cit. on p. 214).

[Sig26] Signal. *Signal Messenger: Speak Freely*. 2026. URL: https://signal.org/ (visited on 2026-02-06) (cit. on pp. 15, 214).

[SKD20a] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. "Assessing the Overhead of Post-Quantum Cryptography in TLS 1.3 and SSH". In: *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '20. Association for Computing Machinery, 2020, pp. 149–156. ISBN: 978-1-4503-7948-9. DOI: 10.1145/3386367.3431305 (cit. on pp. 102–104, 107, 109, 110).

[SKD20b] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. "Post-Quantum Authentication in TLS 1.3: A Performance Study". In: *Proceedings 2020 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. Internet Society, 2020. ISBN: 978-1-891562-61-7. DOI: 10.14722/ndss.2020.24203 (cit. on pp. 102–104).

[Sma+18] N Smart, M Abdalla, E Bjørstad, et al. *Algorithms, Key Size and Protocols Report (2018)*. H2020-ICT-2014 – Project 645421. ECRYPT - CSA, 2018. URL: https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf. Archived at https://web.archive.org/web/20240626071626/https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf (cit. on p. 117).

[SOG23] SOG-IS Crypto Working Group. *Agreed Cryptographic Mechanisms*. Version 1.3. 2023-02. URL: https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf (visited on 2023-08-30). Archived at https://web.archive.org/web/20260420142045/https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf (cit. on p. 157).

[Som14] Juraj Somorovsky. "On the Insecurity of XML Security". In: *it-Information Technology* 56.6 (2014), pp. 313–317 (cit. on pp. 161, 273).

[Som+12] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, et al. "On Breaking SAML: Be Whoever You Want to Be". In: *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, 2012-08, pp. 397–412. ISBN: 978-931971-95-9. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/somorovsky (cit. on pp. 151, 273).

[SS12] Juraj Somorovsky and Jörg Schwenk. "Technical Analysis of Countermeasures against Attack on XML Encryption–or–Just Another Motivation for Authenticated Encryption". In: *2012 IEEE Eighth World Congress on Services*. IEEE, 2012, pp. 171–178 (cit. on p. 273).

[PQ-25] PQ-SORT Team. *PQ-SORT: Post-Quantum Signatures On-Ramp Tests*. 2025. URL: https://pqsort.tii.ae/ (visited on 2025-12-11). Archived at https://web.archive.org/web/20260427142343/https://pqsort.tii.ae/data_nist_candidates.csv (cit. on pp. 39, 41).

[Ste24] Douglas Stebila. *Security Analysis of the iMessage PQ3 Protocol*. 2024. URL: https://eprint.iacr.org/2024/357 (visited on 2025-11-17). Pre-published (cit. on p. 3).

[SFG25] Douglas Stebila, Scott Fluhrer, and Shay Gueron. *Hybrid Key Exchange in TLS 1.3*. Internet Draft draft-ietf-tls-hybrid-design-16. Internet Engineering Task Force, 2025-09-07. 23 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design (visited on 2025-10-31) (cit. on pp. 111, 203).

[SM23a] Douglas Stebila and Michele Mosca. *Open Quantum Safe Benchmarking, KEM Performance*. 2023-06. URL: https://web.archive.org/web/20230630133536/https://openquantumsafe.org/benchmarking/visualization/speed_kem.html (cit. on p. 110).

[SM23b] Douglas Stebila and Michele Mosca. *Open Quantum Safe Benchmarking, TLS Handshake Performance*. 2023-06. URL: https://web.archive.org/web/20230630104243/https://openquantumsafe.org/benchmarking/visualization/handshakes.html (cit. on p. 108).

[SM16] Douglas Stebila and Michele Mosca. "Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project". In: *Selected Areas in Cryptography – SAC 2016*. Ed. by Roberto Avanzi and Howard Heys. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 14–37. ISBN: 978-3-319-69453-5. DOI: 10.1007/978-3-319-69453-5_2 (cit. on pp. 102, 103, 105, 107, 111, 119).

[TPD21] Chengdong Tao, Albrecht Petzoldt, and Jintai Ding. "Efficient Key Recovery for All HFE Signature Variants". In: *Advances in Cryptology – CRYPTO 2021*. Lecture Notes in Computer Science. Springer International Publishing, 2021, pp. 70–93. ISBN: 978-3-030-84242-0. DOI: 10.1007/978-3-030-84242-0_4 (cit. on p. 104).

[Tas+22] George Tasopoulos, Jinhui Li, Apostolos P. Fournaris, et al. "Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems". In: *Information Security Practice and Experience*. Ed. by Chunhua Su, Dimitris Gritzalis, and Vincenzo Piuri. Lecture Notes in Computer Science. Springer International Publishing, 2022, pp. 432–451. ISBN: 978-3-031-21280-2. DOI: 10.1007/978-3-031-21280-2_24 (cit. on pp. 102, 103, 107, 109, 110).

[The26a] The Botan Authors. *Botan — A C++ Cryptography Library*. 2026. URL: https://botan.randombit.net/ (visited on 2026-04-28) (cit. on p. 263).

[The26b] The Catch2 Authors. *Catch2 — A Unit Testing Framework for C++*. Catch Org, 2026. URL: https://github.com/catchorg/Catch2 (visited on 2026-04-28) (cit. on p. 263).

[Tib14] Mehdi Tibouchi. "Elligator Squared: Uniform Points on Elliptic Curves of Prime Order as Uniform Random Strings". In: *Financial Cryptography and Data Security*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Springer, 2014, pp. 139–156. ISBN: 978-3-662-45472-5. DOI: 10.1007/978-3-662-45472-5_10 (cit. on p. 239).

[W3C21] W3C. *Web Authentication: An API for Accessing Public Key Credentials - Level 2*. 2021-04-08. URL: https://www.w3.org/TR/2021/REC-webauthn-2-20210408/ (visited on 2026-04-27) (cit. on p. 15).

[W3C03] W3C. *XML Advanced Electronic Signatures (XAdES)*. 2003-02-20. URL: https://www.w3.org/TR/XAdES/ (visited on 2024-06-27) (cit. on pp. 161–163).

[W3C13a] W3C. *XML Encryption Syntax and Processing Version 1.1*. 2013-04-11. URL: https://www.w3.org/TR/xmlenc-core1/ (visited on 2022-12-13) (cit. on pp. 128, 131, 133, 134, 140, 142).

[W3C10] W3C. *XML Path Language (XPath) 2.0 (Second Edition)*. 2010-12-14. URL: https://www.w3.org/TR/xpath20/ (visited on 2024-01-31) (cit. on p. 161).

[W3C13b] W3C. *XML Signature Best Practices*. 2013-04-11. URL: https://www.w3.org/TR/2013/NOTE-xmldsig-bestpractices-20130411/ (visited on 2024-01-24) (cit. on p. 131).

[W3C13c] W3C. *XML Signature Syntax and Processing Version 1.1*. 2013-04-11. URL: https://www.w3.org/TR/xmldsig-core1/ (visited on 2022-12-12) (cit. on pp. 128, 135, 138, 161, 162).

[W3C02] W3C. *XML-Signature XPath Filter 2.0*. 2002-11-08. URL: https://www.w3.org/TR/2002/REC-xmldsig-filter2-20021108/ (visited on 2023-09-08) (cit. on p. 150).

[W3C17] W3C. *XSL Transformations (XSLT) Version 3.0*. 2017-06-08. URL: https://www.w3.org/TR/xslt-30/ (visited on 2024-01-31) (cit. on p. 161).

[Wan+24] Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, et al. *SNOVA*. National Institute of Standards and Technology, 2024 (cit. on pp. 35, 39).

[Web+26] Paul Webster, Lucas Berent, Omprakash Chandra, et al. *The Pinnacle Architecture: Reducing the Cost of Breaking RSA-2048 to 100 000 Physical Qubits Using Quantum LDPC Codes*. 2026-02-12. DOI: 10.48550/arXiv.2602.11457. arXiv: 2602.11457 [quant-ph]. Pre-published (cit. on pp. 7, 9).

[WvdG20] D. L. Weller and R. van der Gaag. *Incorporating Post-Quantum Cryptography in a Microservice Environment*. 2020-02-09. URL: https://rp.os3.nl/2019-2020/p13/report.pdf. Archived at https://web.archive.org/web/20250629125239/https://rp.os3.nl/2019-2020/p13/report.pdf (cit. on pp. 102–105, 107, 109–111).

[Wes22] Bas Westerbaan. *NIST's Pleasant Post-Quantum Surprise*. The Cloudflare Blog. 2022-07-08. URL: http://blog.cloudflare.com/nist-post-quantum-surprise/ (visited on 2022-12-15). Archived at https://web.archive.org/web/20260307024409/https://blog.cloudflare.com/nist-post-quantum-surprise/ (cit. on pp. 104, 117).

[Wes24] Bas Westerbaan. *Re: [CFRG] [TLS] X-Wing: The Go-to PQ/T Hybrid KEM?* 2024-01-11. URL: https://mailarchive.ietf.org/arch/msg/cfrg/R6LDb44AIKSNClPHeAp9JuLBS8s/ (visited on 2026-04-27) (cit. on p. 112).

[WS23] Bas Westerbaan and Douglas Stebila. *X25519Kyber768Draft00 Hybrid Post-Quantum Key Agreement*. Internet Engineering Task Force, 2023-09. URL: https://datatracker.ietf.org/doc/draft-tls-westerbaan-xyber768d00/03/ (cit. on p. 112).

[Wha26] WhatsApp. *WhatsApp | Secure and Reliable Free Private Messaging and Calling*. 2026. URL: https://www.whatsapp.com (visited on 2026-02-06) (cit. on p. 214).

[Wha24] WhatsApp. *WhatsApp Encryption Overview*. Version 8. WhatsApp, 2024-08-19. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (visited on 2025-11-17) (cit. on p. 3).

[Why+17] W. Whyte, Z. Zhang, S. Fluhrer, et al. *Internet-Draft: Quantum-Safe Hybrid (QSH) Key Exchange for Transport Layer Security (TLS) Version 1.3*. 2017-10. URL: https://datatracker.ietf.org/doc/draft-whyte-qsh-tls13/06/ (cit. on pp. 107, 113).

[Xag21] Keita Xagawa. *Anonymity of NIST PQC Round 3 KEMs*. 2021. URL: https://eprint.iacr.org/2021/1323 (visited on 2025-01-10). Pre-published (cit. on pp. 225, 226, 262).

[Zal99] Christof Zalka. "Grover's Quantum Searching Algorithm Is Optimal". In: *Physical Review A* 60.4 (1999-10-01), pp. 2746–2751. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.60.2746. arXiv: quant-ph/9711070 (cit. on pp. 11, 117).

[Zav+20] Greg Zaverucha, Melissa Chase, David Derler, et al. *Picnic*. National Institute of Standards and Technology, 2020 (cit. on p. 102).

[Zha+15] Jiang Zhang, Zhenfeng Zhang, Jintai Ding, et al. "Authenticated Key Exchange from Ideal Lattices". In: *Advances in Cryptology - EUROCRYPT 2015*. Lecture Notes in Computer Science. Springer, 2015, pp. 719–751. ISBN: 978-3-662-46803-6. DOI: 10.1007/978-3-662-46803-6_24 (cit. on p. 116).

[Zuc+01] Robert Zuccherato, Patrick Cain, Carlisle Adams, et al. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. Request for Comments RFC 3161. Internet Engineering Task Force, 2001-08. 26 pp. DOI: 10.17487/RFC3161 (cit. on pp. 73, 75, 165).