Post-quantum XML and SAML Single Sign-On

Johannes Müller
CNRS/LORIA
Nancy, France
johannes.mueller@loria.fr

Jan Oupický
SnT/University of Luxembourg
Esch-sur-Alzette, Luxembourg
jan.oupicky@uni.lu

ABSTRACT

Extensible Markup Language (XML) is one of the most popular serialization languages. Since many security protocols are built using XML, it also provides cryptographic functionality. A central framework in this area is the Security Assertion Markup Language (SAML). This standard is one of the most widely used options for implementing Single Sign-On (SSO), which allows users to authenticate to different service providers using the credentials from a single identity provider.

Like all other security protocols currently in use, the security and privacy of XML-based frameworks such as SAML is threatened by the development of increasingly powerful quantum computers. In fact, future attackers with access to scalable quantum computers will be able to break the currently used cryptographic building blocks and thus undermine the security of the SAML SSO to illegally access sensitive private information.

Post-quantum cryptography algorithms have been developed to protect against such quantum attackers. While many security protocols have been migrated into the quantum age by using post-quantum cryptography, no such solutions for XML and the security protocols based on it have been developed, let alone tested.

We make the following contributions to fill this gap. We have designed post-quantum solutions for the cryptographic building blocks in XML and integrated them into the SAML SSO protocol. We implemented our solutions in the OpenSAML, Apache Santuario, and BouncyCastle libraries and extensively tested their performance for various post-quantum instantiations. As a result, we have created a comprehensive and solid foundation for post-quantum XML and post-quantum SAML SSO migration.

KEYWORDS

XML, SAML, post-quantum, SSO, single sign-on

1 INTRODUCTION

Extensible Markup Language (XML) is a markup language used for data serialization. XML is one of the most popular data serialization languages and has a large ecosystem built around it, such as *Extensible Stylesheet Language Transformations (XSLT)* [80], a language used to describe transformations of XML documents, *XML Schema Definition (XSD)* [76], a language used to define and validate the structure of an XML document, or *XPath* [75], a query language used to navigate through an XML document.

XML also supports cryptographic algorithms, such as digital signatures and encryption, to build security protocols. In fact, the World Wide Web Consortium (W3C) has introduced the XML Signature [79] and XML Encryption [77] standards, which are widely used to represent digital signatures and encrypted data as an XML document. Another example is the *XML Advanced Electronic Signatures (XAdES)* standard [28] (which extends the XML Signatures standard [79]) that is used throughout the EU as one of the signature formats compliant with the eIDAS regulation.

XML signatures and XML encryption are part of many XML-based protocols, e.g. *Simple Object Access Protocol (SOAP)* [74], a protocol used to communicate with Web services, *Security Assertion Markup Language (SAML)* [52], a framework for exchanging authentication and authorization information about individuals between two parties, or *Health Level 7 (HL7)* [32], a set of standards for exchanging healthcare information. What these applications have in common is that they protect sensitive personal data (e.g. health records) from the eyes and hands of unauthorized parties.

One of the primary use cases for SAML is to facilitate authentication with *Single Sign-On (SSO)* functionality. Single sign-on allows a user to authenticate to multiple *service providers* using a single set of credentials from a central authority, the *identity provider*. SAML-based SSO is supported across the Web by providers including Google [30], Microsoft [46], Cloudflare [21], or ServiceNow [62]. Since SSO can be used in the healthcare sector to efficiently and securely receive and process private medical data [59], numerous providers such as ChartRequest [19], Enterprise Health [26], or Miniorange [48] offer SAML-based solutions for such applications.

The security of currently used public key cryptography algorithms, including those in XML, is based on "classical" hardness assumptions, such as the integer factorization problem or the discrete logarithm problem. However, Peter Shor showed in 1994 [64] that these problems can be solved in polynomial time on a quantum computer. Thus, algorithms whose security is based on such assumptions, such as RSA or (EC)DSA, are vulnerable to future adversaries who can use (scalable) quantum computers.

To address the threat posed by quantum attackers, the cryptographic community has developed new public-key cryptographic algorithms whose security relies on the hardness of problems that are assumed to be "hard" even for quantum computers (e.g., the Shortest Vector Problem). These algorithms are called *post-quantum* or sometimes *quantum-safe*. Many post-quantum algorithms have been developed, especially during the NIST standardization process for post-quantum cryptography [49]. While the standardization process is ongoing, it is important to study how to efficiently and seamlessly migrate our existing cryptographic systems to these new post-quantum cryptography algorithms.

Because current XML signature and public key encryption standards support only classically secure algorithms [25, 77, 79], they

Proceedings on Privacy Enhancing Technologies 2024(4), 525–543

https://doi.org/10.56553/popets-2024-0128

are vulnerable to future quantum attackers. This threat is especially true for SAML, where a quantum adversary can forge the classical signatures to completely undermine the security features of the SAML SSO protocol. As a result, such attackers could, for example, illegally access private health data. Moreover, a current eavesdropper can record and store sensitive information encrypted using XML public key encryption, and decrypt them in the future when it gains access to a quantum computer (so-called "store now, decrypt later" attacks).

Despite its practical relevance, to the best of our knowledge there is no previous work that proposes and evaluates solutions for post-quantum XML and post-quantum SAML (SSO). However, migrating these backbone languages and frameworks of our digitized world is necessary to prevent a future security and privacy disaster: for example, a quantum attacker could forge XAdES-based eIDAS signatures used to sign real-world contracts, or break the SAML SSO protocol to illegally authenticate to service providers (e.g., banks, insurance companies, or government agencies) to steal sensitive private data.

From a practical point of view, the main challenge is to develop solutions for the upcoming migration period that are compatible with current standards and at the same time quantum-safe. Only with such conceptually planned and carefully detailed solutions will it be possible to migrate XML-based security protocols on a large scale and seamlessly into the quantum age.

1.1 Contributions

We offer the following contributions to address the open challenges mentioned above:

(1) We discuss the quantum security of the public-key cryptography in XML and of the SAML framework to illustrate the significance of this threat.
(2) We design the first post-quantum solutions for XML digital signatures, for XML public key encryption, and for SAML. In each case, we propose *purely post-quantum* solutions, which use only post-quantum cryptography, and *hybrid post-quantum* solutions, which combine classical and post-quantum cryptography.
(3) We implement our post-quantum solutions in Java using OpenSAML [72], Apache Santuario [70] and BouncyCastle [71]. We instantiate them with the (currently) most reasonable post-quantum cryptographic algorithms and extensively analyze their performance. Our implementation is available at https://github.com/PQSAML/index.
(4) Based on our findings, we discuss the features and limitations of post-quantum XML and post-quantum SAML, and give recommendations for their post-quantum migration.

Due to its practical relevance, the focus of our work is on the hybrid variants, as they provide the desired properties of being both compatible with current standards and quantum-safe (see Section 4.3 for a more detailed discussion). On a technical level, the development and testing of these hybrid variants has also attracted most of our attention, as we aimed to realize the hybrid properties in a "minimally invasive" way and with as little performance overhead as possible. To achieve this goal within the constraints set by the current XML/SAML standards, we have designed and compared various solutions. At the same time, through this approach we have identified limitations that prevent some more efficient (hybrid) post-quantum solutions within the current standards, and which we argue should be removed.

Our implementation of composite post-quantum signatures (Section 5.2.1) is available in BouncyCastle since version 1.78. Additionally, we are in the process of submitting our solutions to the maintainers of OpenSAML and Apache Santuario.

Overall, our work paves the way for the practical migration of XML-based security protocols into the post-quantum era, and provides the first such case study for a widely used application, the SAML framework.

1.2 Related work

There have been numerous papers proposing and analyzing post-quantum versions of popular protocols such as TLS [4, 68], SSH [22, 65], WireGuard [34], OpenVPN [73], or FIDO2 [12]. While there are a number of publications that analyze the security of classical XML signatures, XML encryption, and the SAML framework [36, 37, 42, 67], we have not come across a paper that studies post-quantum XML signatures and public key encryption, or post-quantum SAML.

1.3 Overview

We start with an introduction to XML signatures/encryption and a discussion of the threat of quantum computers in Section 2. In Section 3, we introduce SAML and its single sign-on (SSO) protocol and discuss its security against quantum adversaries. In Section 4, we present the current state of post-quantum cryptography. We propose our solutions for purely and hybrid post-quantum XML signatures and XML public key encryption in Sections 5 and 6. In Section 7 we propose our solutions for purely and hybrid post-quantum SAML SSO. We conclude with a summary and discuss future work in Section 8.

We assume that the reader is familiar with the notions and security properties of digital signatures, symmetric encryption, public key encryption and hash functions.

2 XML SIGNATURES AND ENCRYPTION

In this section, we briefly explain the XML signature [79] and XML encryption [77] standards. While the standards define multiple methods for achieving their goals, we focus on those ones that are relevant to XML documents in SAML [52].

Notation. We use the `typewriter font` to refer to a particular XML element, and we omit XML namespaces in our descriptions.

2.1 XML signatures

A digital signature in XML 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`.

Types. The standard [79] defines three types of XML signatures:

- Enveloping: The signed data[1] is an XML element `Object`, which is a child of `Signature`.

[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".

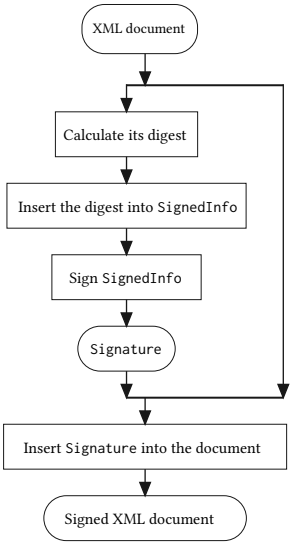

Figure 1: XML signature creation.

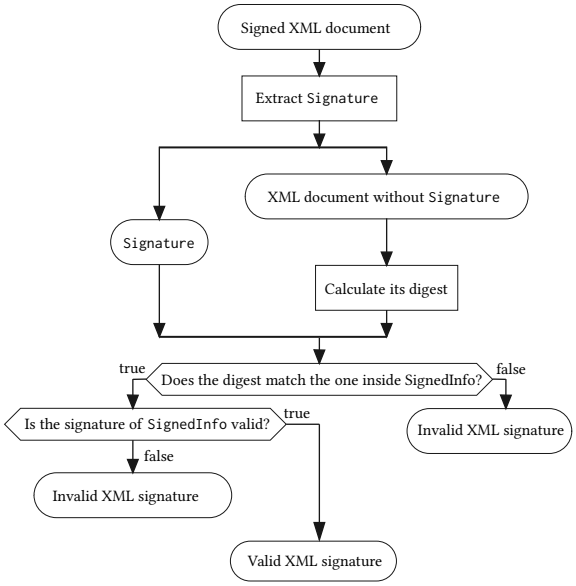

Figure 2: XML signature verification.

- Enveloped: The signed data is an XML document. `Signature` is a descendant of this document.
- Detached: The signed data is external to `Signature`, i.e., `Signature` is a separate XML document.

From now on, we will only focus on *enveloped* XML signatures as they are the only supported type in SAML [52].

Structure. A `Signature` element has the following children:

- `SignedInfo` includes the signature algorithm identifier and the references to signed data. Specifically, inside `SignedInfo` there is a `Reference` element which contains a URI reference to the signed data, a digest of the signed data and a set of transformations that are applied before the hashing.
- `SignatureValue` contains a signature over `SignedInfo`, i.e., the output of the signing operation of the signature algorithm.
- `KeyInfo` contains information about the public key which can be used to verify the signature in `SignatureValue`. Typically, `KeyInfo` contains a certificate chain.

Creation of XML signature. 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 during verification.
(3) Sign `SignedInfo` using the signature algorithm. Paste the result into `SignatureValue`.
(4) Prepare `KeyInfo` containing the verification public key.
(5) Paste the resulting `Signature` into the original XML document. The result is a signed XML document.

Figure 1 provides a simplified diagram of the signing procedure.

Verification of XML signature. The 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 2 provides a simplified diagram of the verification procedure. While the order of the verification steps can be reversed (first verify the references, then verify the signature), this order is not recommended as it can introduce potential DDoS attacks [78].

2.2 XML public key encryption

The XML Encryption standard [77] defines a method for encrypting data and representing the result as an XML element. The standard does not impose any restrictions on the plaintext data, i.e., it can be an XML element or a binary file.

Similar to the XML signature standard [77], the encryption standard defines multiple ways how to achieve the goal. We focus on a specific case where the data is encrypted using a public key encryption algorithm since our overall objective is post-quantum SAML SSO and many SAML SSO implementations [6, 20, 47, 53, 54] use XML public key encryption. Specifically, it is an instance of a *hybrid encryption*, combining public key and symmetric encryption.

Encryption. While the XML encryption standard can be implemented with different cryptographic primitives, for illustrative purposes we describe the case where RSA is the public key encryption scheme and AES-256 is the symmetric encryption scheme:

(1) Generate an AES-256 key.
(2) Encrypt the plaintext using AES-256 with the generated key.
(3) Encrypt the AES-256 key using RSA with the 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`.

See Figure 4 (Appendix B) 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. Listing 1 in the appendix shows an example XML encrypted document.

Decryption. The decryption process simply reverses the encryption process, i.e., the AES key is decrypted using the RSA private key, and the AES key is used to decrypt the actual data. See Figure 5 (Appendix B) for a simplified diagram of the decryption procedure.

Authentication. Although the XML encryption standard [77] does not explicitly provide for authentication or integrity of the ciphertext, this can be solved for the data ciphertext by using an authenticated encryption (AE) scheme, such as AES-GCM [24]. However, the encrypted symmetric key ciphertext is not authenticated/integrity checked because RSA-OAEP [39] is not AE. For this reason, XML encryption is usually combined with XML signatures.

2.3 Quantum threat

We briefly explain the threat that quantum attackers pose to the current XML standards.

XML signatures. The XML signature standard [79] currently supports only the classical signature schemes RSA, (EC)DSA and EdDSA [25]. The security of all these schemes reduces to the integer factorization or the discrete logarithm problem. Since these hardness assumptions can be solved in polynomial time by a cryptographically relevant quantum computer, any quantum adversary can forge current XML signatures.

XML encryption. The XML encryption standard [77] supports both public key encryption and symmetric key encryption. Since symmetric encryption is not severely affected by quantum computers, doubling key sizes is considered sufficient protection (see Appendix C for more details). The standard currently supports only RSA as a public key encryption scheme, which is not secure against quantum computers. In particular, a quantum attacker who obtains a RSA-encrypted ciphertext (from current or future conversations) can learn the private message.

3 SAML AND SINGLE SIGN-ON

Security Assertion Markup Language (SAML) is an XML-based security framework [52]. It has two main parts: the first defines the structure of SAML messages, which are XML documents, and the second defines various protocols that use SAML messages.

The primary use case for SAML is to establish a single sign-on (SSO) authentication scheme. SSO allows a user to log in to multiple service providers with a single set of credentials from a single identity provider. The SAML SSO protocol is not only supported by all major identity providers on the Internet, but also by many universities and research institutes in Europe and North America [20, 30, 35, 46, 63]. Researchers can use this tool to log in to publishers' websites or research databases that contain sensitive data (e.g., health data).

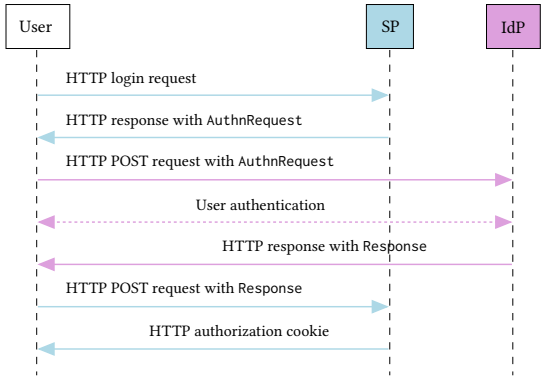

Figure 3: SAML SSO protocol flow.

3.1 Protocol

The SAML SSO protocol has three participants: the *user*, the *service provider (SP)*, and the *identity provider (IdP)*. We assume that the user has previously registered with the IdP and has a set of credentials.

Setup. In the typical scenario, the user wants to log in to the SP using the credentials they have established with the IdP. The protocol assumes that the SP and IdP have a pre-existing relationship, i.e., they have exchanged configuration details and agreed on user identifiers, such as email.

Overview. The main protocol flow of SAML SSO is as follows:

(1) The SP creates a SAML message `AuthnRequest` and sends it to the IdP.
(2) The IdP processes the `AuthnRequest` and authenticates the user in question.
(3) The IdP creates a SAML message `Response` containing the result of the authentication and sends it to the SP.

The protocol gets more complex at the lower level, where it needs to define how the SAML messages are actually transported from the SP to the IdP and vice versa. This is what *bindings* are for; they specify how to transport SAML messages. Throughout this paper we implicitly assume that the *HTTP POST Binding* (as depicted in Figure 3) is used.

3.2 Security

The security mechanisms in SAML SSO fall into two categories, those that depend on the binding and those that do not.

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 [60], 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 [60], 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 3, there are two TLS channels: user–SP and user–IdP.

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 2.1) provide integrity and authenticity of SAML messages, and XML encryption (Section 2.2) is used to encrypt some of the 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, i.e. the `Assertion` element within `Response` is encrypted and the result of the XML encryption is within `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 public key infrastructure (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 *Certificate Authority (CA)* that is trusted by both parties.

Recall from Section 2 that XML signatures and XML public key encryption have the `KeyInfo` element, which contains information about the signing/encryption key. We assume that `KeyInfo` always contains an X.509 [14] public key certificate signed by the trusted CA or, in the case of XML public key encryption, an encrypted 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 such proof.

3.3 Quantum threat

Based on Section 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 called the "store now, decrypt later" attack.
(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. Thus, in this paper we discuss the post-quantum solutions to all of these security mechanisms.

4 POST-QUANTUM CRYPTOGRAPHY

For more than two decades, the cryptographic community has been developing new public key algorithms that are assumed to be secure even against a quantum adversary. These algorithms are referred to as *post-quantum* [8].

4.1 Overview

There are several approaches to post-quantum cryptography that differ in their underlying hardness assumptions and in their performance:

- The security of *lattice-based* schemes is based on the hardness of the Shortest Vector Problem (or related problems). Typically, lattice-based schemes do not have the smallest key/signature sizes, but their operations are the fastest.
- *Code-based* schemes depend on the hardness of code decoding problems, such as the Syndrome Decoding Problem. While code-based schemes are generally slower than lattice-based schemes and their key sizes are also larger, code-based hardness assumptions tend to be more conservative.
- *Hash-based* schemes are also considered a conservative option because their security is tied to the security of the underlying hash function, which is a well-studied primitive. Hash-based schemes have small public keys but large signatures, and they are also slower than lattice-based schemes.
- *Isogeny-based* and *multivariate* schemes have promising performance profiles, but most of these schemes have been broken [9, 17]. Although there are newer constructions such as CSIDH [18], SQISign [23], or MAYO [10], they need further evaluation.

Overall, while post-quantum algorithms have larger size requirements than their classical counterparts, some of them are faster than classical algorithms, as illustrated in Table 8 in the appendix.

Key Encapsulation Mechanism. A central cryptographic building block of post-quantum cryptography is *Key Encapsulation Mechanisms (KEMs)*. Using a KEM, anyone with access to the public key pk (of a particular recipient) can run the randomized encapsulation algorithm of the KEM to obtain a fresh symmetric key k and a public-key ciphertext e that encrypts k under pk. The recipient can use its secret key sk to decrypt the ciphertext e.

The primary application of KEMs is to implement hybrid public-key encryption schemes (Appendix B). The sender (with input m and pk) first runs the encapsulation scheme of the KEM to obtain a symmetric key k and an encryption of k under pk, and then symmetrically encrypts the message m with k. The recipient first decrypts the KEM ciphertext with its secret key sk and then uses the decrypted symmetric key k to recover the message m.

4.2 Standardization

In 2016, NIST launched a process to find suitable post-quantum replacements for classical algorithms [49]. NIST is seeking post-quantum signatures and key encapsulation mechanisms (KEMs). In 2022, NIST announced [2] that it had selected one KEM (Kyber [61]) and three signatures (Dilithium [43], Falcon [58], and SPHINCS+ [33]) for standardization from the 69 algorithms at the beginning of the process [1]. NIST also extended the process to a fourth round, with the goal of selecting additional KEMs for standardization from four candidates: BIKE [5], HQC [45], Classic McEliece [3] and SIKE [38].[2] In 2022, NIST also started a parallel standardization process to find alternative post-quantum signatures [50].

Due to the importance of this standardization process for post-quantum migration in practice, we focus on the closer candidate environment in this paper.

In their call for proposals [49], NIST defined five security levels, from 1 (least secure) to 5 (most secure), which are used to categorize the security of post-quantum algorithms. These security levels are defined as the number of operations it takes to attack block ciphers and hash functions. For example, the best attack on an algorithm achieving security level 1 must take resources greater than doing a key search attack on AES-128.

4.3 Hybrid post-quantum cryptography

Since post-quantum algorithms are still relatively new compared to classical algorithms, such as RSA, it is not surprising that some of them (such as SIKE [17] and Rainbow [9]) turned out to be less secure than intended (even in the classical setting).

To mitigate the risk, it is beneficial to combine (well-studied) classical with (less studied, but potentially stronger) post-quantum cryptographic algorithms. This *hybrid* approach is often recommended [16, 66] in the transition phase, as it ensures that the resulting scheme is secure as long as one of the components is secure.

Another practical advantage of hybrid approaches is the ease of compliance with existing regulations, such as eIDAS in the EU. Since updating such high-level regulations is part of a political process and therefore slow, a regulated institution or company (such as a trust service provider), that wants to start its post-quantum migration now, can use a hybrid approach to still comply with current regulations.

Since these benefits come at the expense of performance, it is important to understand which hybrid combinations strike a good balance between security and efficiency. This is a major focus of this paper, in which we propose both *purely post-quantum* and *hybrid post-quantum* solutions for each problem.[3]

4.4 Post quantum TLS

In the transition to post-quantum SAML SSO, the TLS connections must also be made post-quantum. Since TLS is a security mechanism independent of the SAML SSO protocol itself (see Section 3), post-quantum TLS is not the focus of this paper. Instead, we refer the reader to [4] for an overview of the state of the art.

4.5 Post-quantum certificates

XML signatures and XML public-key encryption use X.509 public key certificates to determine which public key to use for verification or encryption (see Section 2). Therefore, when moving to post-quantum XML and SAML SSO, these certificates must also be post-quantum, meaning that the entire certificate chain must be signed with post-quantum signatures. Since this aspect is independent of the actual XML and SAML processes (similar to TLS, as explained above), we refer the reader to [13, 40] for more details on this topic.

5 POST-QUANTUM XML SIGNATURES

In this section, we design and test purely post-quantum (Section 5.1) and hybrid post-quantum (Section 5.2) XML signatures.

5.1 Purely post-quantum

Design. Since the XML signature standard is *crypto-agile*, it can be implemented with any signature scheme. Thanks to this property, the design of purely post-quantum XML signatures is reduced to the creation of appropriate XML identifiers for these post-quantum algorithms. In our (anonymized) implementation, we have used custom experimental identifiers which can be found at https://github.com/PQSAML/index.

Experimental setup. We conducted an experiment in which we signed XML documents with different purely post-quantum XML signatures and tested their performance. For all experiments in this paper, we created new forks of OpenSAML [72], Apache Santuario [70] and BouncyCastle [71] libraries. The machine used for measuring was Oracle Cloud VM.Standard.A1.Flex (Arm64) running Ubuntu 22.04.3 LTS. OpenSAML is used to create, sign and verify SAML messages and 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, 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

[2] SIKE is no longer considered because it is broken [17].

[3] We use the term *purely post-quantum* exclusively for solutions that do not combine post-quantum and classical algorithms.

`AuthnRequest` and `Response`, and the average size of the complete XML document size.

Results. We have summarized our experimental results in Table 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 because `Signature` has constant size.

The signing algorithms of all post-quantum schemes are slower than the one of ECDSA (P-256), but the average signing times of Dilithium (both at security levels 2 and 5) and Falcon (security level 1) are better than those of RSA and ECDSA (P-521). In particular, SPHINCS+'s signing times are more than 100× slower than the others, except for 15360-bit RSA.

Falcon's (security level 1) verification time is more than twice as fast as ECDSA (P-256), and Dilithium's (security level 2) and Falcon's (security level 5) times are also about 30% faster than ECDSA (P-256). SPHINCS+ verification is still slower than ECDSA (P-256), but only by a factor of 4 (as opposed to a factor of 3654 for signing).

From post-quantum signatures, Dilithum (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, Dilithium and Falcon are faster than ECDSA (P-521) and RSA. Due to the slow signing time SPHINCS+ is the slowest (together with 15360-bit RSA) in total.

In unstable or low-bandwidth networks, the larger signature sizes of Dilithium and the other post-quantum algorithms can make these solutions even slower than the lighter classical ones.

5.2 Hybrid post-quantum

Hybrid post-quantum XML signatures are more complex to realize than purely post-quantum ones. There are essentially two approaches: using a hybrid post-quantum signature scheme (Section 5.2.1) or using two separate signature schemes (Section 5.2.2).

Design goals. Before we present and test our solutions, we first define the 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.
- *Non-separability* [13]: An adversary cannot strip the post-quantum component of the signature and obtain a valid classical signature.

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 (Section 4.3).

5.2.1 Composite hybrid. In this approach, we use a single hybrid post-quantum signature scheme to implement the XML signature, which means that we have only one `Signature` element in the signed XML document. Such hybrid signature schemes are sometimes referred to as *composites* or *composite hybrid signatures* [56]. Note that this approach is inherently not backward compatible.

Design. The simplest implementation is to use concatenation, where the hybrid public and private keys are the results of concatenating the respective classical and post-quantum keys. The hybrid signature is then the concatenation of two signatures, and the hybrid signature is valid if both signatures are valid. Such a hybrid scheme can be treated as a single signature scheme in the standard XML signing and verification processes. We describe the construction in more detail in Appendix D.

However, without further means, such a concatenation scheme is separable and it is disadvantageous from a technical point of view, for example, because the decoding is based on the lengths of the different components. To address these issues, our implementation is based on the IETF draft specification [56], which is non-separable and uses ASN.1 structures for encoding.

Experimental setup. We used essentially the same setup as in Section 5.1, except that here we use hybrid composite post-quantum signatures instead of purely post-quantum signatures. Due to the large number of possible combinations we present only four pairings of ECDSA with Dilithium/Falcon.[4]

Performance results. We present our results in Table 2. While hybrid composite 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. For example, the document signed with the ECDSA (P-256) + Dilithium (security level 2) composite signature is only $3-4\%$ larger than the document signed with Dilithium alone, because the classical signature is already small. On the other hand, the total time is about 1.4× the total time with Dilithium alone, because Dilithium already has fast operations. We see similar results for composite signatures with Falcon.

5.2.2 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 presented earlier, the separate hybrid approach can provide backwards compatibility because the signatures can be verified separately.

Design. 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 [79] 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 (Section 2.1) is used, then (at least) one of the XML signatures will fail verification. This is because the enveloped signature

[4]The specification [56] does not explicitly define composites with ECDSA with P-521, only P-384. We used P-521 because it provides 256-bit security.

Table 1: Performance of purely post-quantum XML signatures

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
Dilithium (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)
Falcon (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)
Dilithium (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)
Falcon (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)
SPHINCS+ (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)
SPHINCS+ (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.

Table 2: Performance of composite hybrid post-quantum XML signatures

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) + Dilithium (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) + Falcon (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) + Dilithium (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) + Falcon (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.

transform, which is the only transform allowed by the SAML standard [52], only removes its parent `Signature` from the reference digest calculation, not the other `Signature`.
- It does not provide the desirable feature of non-separability (see above).

Our solution to achieve non-separability is to sign sequentially. In this case, the XML document 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, because S_A, a signature over d, contains only the enveloped signature transformation, which does not remove S_B from the hash calculation. To avoid this problem, we decided to add an extra transformation that removes the other signature as well. To do this, we tailored the extra transformation to the underlying protocol, since it relies on the location of the `Signature` elements within the signed document. In Section 7.2 we present our solution tailored to the SAML SSO protocol.

Experimental setup. We used the same setup as for the composite hybrid approach (Section 5.2.1) and with the same signature algorithm combinations to directly compare these two approaches.

Performance results. We have summarized our results in Table 3. We see that 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) + Dilithium (security level 2), the increase in size (compared to the composite hybrid) is about 17%, and for ECDSA (P-256) + Falcon (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) + Dilithium (security level 2) is about 61% slower than the equivalent composite hybrid. For example, at security level 5, overhead becomes smaller for ECDSA + Falcon to about 37%.

5.3 Discussion

We have seen that the purely post-quantum solutions for XML signatures are easy to implement, while we had to overcome some technical barriers to realize the two hybrid post-quantum XML signature versions. We found that regarding hybrid approaches, the most straightforward solutions were not the best. Instead, we had to look at the specifics of XML signatures to find efficient and effective solutions. Let us now discuss the main features of the two hybrid versions that we have designed and tested.

The composite hybrid (Section 5.2.1) is more efficient in terms of size and speed. The construction can also be universally applied to any XML-based protocol because it does not change the document structure, and the hybrid signing and verification is typically delegated to the cryptographic library that implements the primitive. We estimate that the implementation effort of this hybrid post-quantum XML signing solution is similar to purely post-quantum XML signatures presented in Section 5.1 where all thats needed, assuming the hybrid signature scheme is already implemented, is to integrate new XML algorithm identifiers. On the downside, the composite hybrid post-quantum XML signature is not backward compatible as it cannot be verified by a purely classical verifier.

Table 3: Performance of separate hybrid post-quantum XML signatures

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) + Dilithium (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) + Falcon (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) + Falcon (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) + Dilithium (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.

The separate hybrid (Section 5.2.2) is slightly slower than the composite approach and also produces larger documents. On the positive side, however, this approach provides backwards compatibility, allowing the signer to create hybrid post-quantum XML signatures without worrying about whether the verifier can support post-quantum algorithms. The main disadvantage of the separate approach is that there is no universal solution that can be applied to any XML-based protocol, resulting in more complex and fragmented implementations.

6 POST-QUANTUM XML ENCRYPTION

In this section, we design and test purely post-quantum (Section 6.1) and hybrid post-quantum (Section 6.2) solutions for XML encryption. As mentioned in Section 2.3, symmetric encryption schemes can be made post-quantum by doubling their key sizes (see Appendix C for 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.

6.1 Purely post-quantum

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 (Section 6.1.1), or replace it with a post-quantum key encapsulation mechanism (KEM) (Section 6.1.2).

6.1.1 Compatible solution. This solution is directly compatible with the current XML standard.

Design. We replace the classical PKE scheme with a post-quantum PKE scheme.

Experimental setup. The experimental setup is similar to the XML signature experiments except that we encrypt and later decrypt the `Assertion` element within the `Response` SAML message (see Section 3). The result of the encryption process is the `EncryptedAssertion` element. We include the RSA-signed encryption certificate in `KeyInfo`.

We used the implementations of the post-quantum PKE schemes Kyber, BIKE and Classical McEliece from BouncyCastle [71]. We also attempted to benchmark their HQC implementation but we were unsuccessful because it regularly returned decryption errors.

Performance results. We have summarized the results of our experiment in Table 4.

As we can see, Kyber is significantly faster (about 20×) than classical 3072-bit RSA-OEAP public key encryption. The Kyber 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 Kyber's XML ciphertext at security level 1. BIKE at security level 5 is the slowest option we tested, taking about 13 times as long as RSA; its XML ciphertext is about 2.3× the size of Kyber'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 8). 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 encryption algorithm. In terms of speed, McEliece is about 40% slower than RSA at security level 1.

6.1.2 More efficient solution. Recall from Section 2.2 that the XML public key encryption is a hybrid encryption scheme that combines a PKE scheme, such as RSA-OAEP [39], with a symmetric cipher, such as AES-GCM [24].

The more efficient solution is to replace the classical PKE component within the hybrid encryption with a post-quantum KEM. This way we avoid a double symmetric encryption as in the simple solution (see above), because in the simple solution the post-quantum PKE component is itself a hybrid scheme that already contains a symmetric encryption.

We did not test this solution because we wanted to stay close to the XML encryption standard [77], which does not (currently) support KEMs. Since we are interested in working solutions for the post-quantum migration phase, we have refrained from making major modifications to the XML standard and the OpenSAML [72] and Apache Santuario [70] libraries.

6.2 Hybrid post-quantum

Our main idea for implementing hybrid post-quantum XML PKE 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 4: Performance of purely post-quantum XML public key encryption

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
Kyber (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)
Kyber (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 [39].

Design. 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 6.1).

Although not specifically intended for the post-quantum setting, such layered encryption is called *super-encryption* in the XML encryption standard [77]. 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 [77], we use the `MimeType` attribute of `EncryptedData` with a value such as `enc/multilayer`.

Experimental setup. To evaluate the performance of our hybrid post-quantum XML public key encryption, we used the same experimental setup as in Section 6.1. We combined the classical XML encryption algorithm 3072-bit RSA-OAEP [39] with Kyber, BIKE and Classic McEliece at security levels 1 and 5.

Performance results. We present our full results in Table 5. We see that using hybrid post-quantum encryption with RSA and Kyber 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 Kyber 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.

Alternative approaches. We present two alternative approaches that are more efficient, but would require significant modifications to the standard, which we want to avoid in this work (see above).

The first alternative is to combine a classical Diffie-Hellman (DH) key exchange with a post-quantum key exchange using a KEM that together outputs two shared secrets. The symmetric key could then be derived from the two shared secrets using a hybrid combiner [11, 29, 57]. Although the XML encryption standard supports deriving the symmetric key from a DH key exchange, it does not currently support either KEM-based key exchange or hybrid combiners.

The second (simpler) alternative is to encrypt only the symmetric key twice (once with the classical and once with the post-quantum PKE scheme), but not the symmetrically encrypted ciphertext. This construction is more time and memory efficient, especially for larger messages, but it requires significant conceptual changes in the existing implementations.

6.3 Discussion

We have shown that purely post-quantum XML public-key encryption can be realized if the underlying cryptographic library provides a post-quantum PKE construction. Our experiments demonstrated that using purely post-quantum XML PKE can be significantly (up to 20 times) faster than using classical RSA PKE at the cost of larger cryptographic material.

We could also see that the increased size comes from the way the XML ciphertext is represented. If the `KeyInfo` element in the XML ciphertext contains the encryption certificate, then the size of the post-quantum solutions is larger. However, there are many use cases where it is not necessary to include the certificate, so post-quantum ciphertexts in XML could in principle also be smaller.

We have also shown how to implement hybrid post-quantum XML public key encryption. In particular, our experiments demonstrate that combining RSA with Kyber is only a bit slower than using RSA alone while doubling the ciphertext size. We have identified two ways (using KEMs and more efficient nested encryption) to reduce this overhead. Since neither method is compatible with the current standard, we advocate that this limitation be removed in the next revision of the standard.

7 POST-QUANTUM SAML SINGLE SIGN-ON

In this section, we describe how to integrate the post-quantum cryptographic building blocks from the previous sections to make the SAML Single Sign-On (SSO) protocol post-quantum. As before, we present two post-quantum versions, one that uses only post-quantum cryptography (Section 7.1) and one that combines classical and post-quantum cryptography (Section 7.2). We implemented these solutions and we present the results of our experiments.

Certificates. Recall from Section 3 that in our SAML SSO setup we assume that the Service Provider (SP) and the Identity Provider (IdP) share a trust anchor and that the signing and encryption certificates are unknown to the recipient. Therefore, certificates are sent inside SAML messages: certificates for signatures inside the corresponding `Signature`, and certificates for public key encryption as `KeyInfo` inside `Extensions` of `AuthnRequest`.

Table 5: Performance of hybrid post-quantum XML public key encryption

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
+ Kyber (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)
+ Kyber (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 [39].

Post-quantum TLS. As explained in Section 3, the TLS protocol is commonly used in SAML SSO to secure the channels between the different parties. Therefore, in order to make the entire SSO protocol post-quantum, we need to use post-quantum TLS as well. However, since the main SSO protocol treats the mechanism for securing the communication channels as a black box, we abstract away from TLS in this section and focus only on the protocol-specific parts of SAML SSO. We refer the reader to [4] for the state of the art in post-quantum TLS.

7.1 Purely post-quantum

Design. We build a purely post-quantum SAML SSO solution by replacing the classical XML signatures and public key encryption with purely post-quantum XML signatures and public key encryption (Sections 5.1 and 6.1).

Experimental setup. The basic setup is as in Sections 5 and 6. In our experiment, the SP creates an `AuthnRequest` with a KEM certificate inside `Extensions` and signs the `AuthnRequest` using the post-quantum XML signature. The IdP verifies the signature of the `AuthnRequest`. The IdP creates a corresponding `Response` with an `EncryptedAssertion` encrypted with the public key from the KEM certificate and signs the `Response`. The SP verifies the signature of the `AuthnRequest` and decrypts the `EncryptedAssertion` with its KEM private key. Signing and encryption certificates are verified.

Performance results. In Table 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 side, the processing time on the IdP side, and the total processing time. Due to the number of possible combinations, we only provide results for the lowest security levels. We use the classical combination of ECDSA signature and RSA PKE as a baseline.

We see that certain combinations of post-quantum algorithms result in faster execution times: Dilithium with Kyber is four times faster, and Falcon with Kyber is more than three times faster than the baseline. Clearly, all post-quantum algorithm combinations result in higher bandwidth compared to the baseline. The most bandwidth-efficient post-quantum combinations are Falcon with Kyber or BIKE; however, these combinations still consume about 2× the baseline's bandwidth. The worst performing combination is the security-conservative pair SPHINCS+ with Classic McEliece with 82× the baseline's bandwidth and 160× times the baseline's total time.

7.2 Hybrid post-quantum

We present two approaches to implementing hybrid post-quantum SAML SSO: the first uses composite hybrid post-quantum signatures (Section 5.2.1) and is not backward compatible, and the second uses separate hybrid post-quantum signatures (Section 5.2.2) 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, while it can still be processed by classical verifiers.

7.2.1 *No backward compatibility.*

Design. The simplest way to implement hybrid post-quantum SAML SSO is to use hybrid composite post-quantum XML signatures (Section 5.2.1) to sign SAML messages in combination with hybrid post-quantum XML public key encryption (Section 6.2) to encrypt messages. This approach does not make any changes to the standard SAML message as each message contains one `Signature` element that uses the hybrid composite post-quantum signature.

The hybrid post-quantum XML PKE can be also implemented without modifying the SAML layer as only the underlying XML encryption library needs to be modified to process the layered encryption, i.e., decrypt twice if the `EncryptedData` has the `MimeType` attribute `enc/multilayer`.

Since neither composite signatures nor hybrid post-quantum encryption are backward compatible, this hybrid solution is also not backward compatible. For example, an IdP cannot issue SAML responses protected by post-quantum algorithms that are verifiable by SPs that have not (yet) migrated their systems.

Experimental setup. To compare the performance between purely and hybrid post-quantum SAML SSO (without backward compatibility), we performed essentially the same experiment as in Section 7.1. In this experiment, we sign a SAML message and construct signature certificates using hybrid composite signatures. The classical encryption (RSA) certificate is signed with the classical component of the composite signature (ECDSA), while the post-quantum certificate is signed with the post-quantum component (Dilithium/Falcon). We chose this configuration because it allows a fair comparison with the backward compatible solution presented in the next section.

Table 6: Performance of purely post-quantum SAML SSO

Sig. 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
Dilithium (2)	Kyber (1)	33 733 (×3.56)	1.38 ± 0.10 (×0.12)	1.80 ± 0.42 (×1.09)	3.18 ± 0.43 (×0.24)
Falcon (1)	Kyber (1)	17 924 (×1.89)	1.89 ± 0.20 (×0.17)	2.07 ± 0.20 (×1.25)	3.95 ± 0.30 (×0.30)
Dilithium (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)
Falcon (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)
Dilithium (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)
Falcon (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)
SPHINCS+ (1)	Kyber (1)	75 203 (×7.93)	1027.41 ± 6.50 (×90.28)	1028.46 ± 6.58 (×623.31)	2055.87 ± 12.17 (×157.78)
SPHINCS+ (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)
SPHINCS+ (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, SPHINCS+ parameters: SHA256, 128s/256s, simple. RSA-OAEP with MGF1-SHA1 [39].

Table 7: Performance of non-backward compatible hybrid post-quantum SAML SSO

Sig. 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
+ Dilithium (2)	+ Kyber (1)	39 888 (× 4.21)	12.26 ± 0.56 (×1.08)	3.05 ± 0.63 (×1.85)	15.32 ± 0.97 (×1.18)
+ Falcon (1)	+ Kyber (1)	24 114 (× 2.54)	12.79 ± 0.25 (×1.12)	3.23 ± 0.24 (×1.96)	16.02 ± 0.38 (×1.23)
+ Dilithium (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)
+ Falcon (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)
+ Dilithium (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)
+ Falcon (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, SPHINCS+ parameters: SHA256, 128s/256s, simple. RSA-OAEP with MGF1-SHA1 [39].

Performance results. We present the results of our experiment in Table 7. They show that using hybrid SAML SSO by combining ECDSA with Dilithium or ECDSA with Falcon for signing and combining RSA with Kyber for public key encryption is only about 20% slower overall. Bandwidth increases 4.2 times for the Dilithium version and 2.54 times for the Falcon version. Using BIKE instead of Kyber 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 roughly 300% slowdown.

7.2.2 Backward compatibility. We present our solution to backward compatible 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 we cannot use hybrid post-quantum XML public-key encryption when we want to achieve backward compatibility. Since classical SPs or IdPs do not support post-quantum public-key encryption, they cannot process the nested hybrid post-quantum ciphertexts.

We have therefore decided to not make secrecy of `Assertion` post-quantum, but to use the classical encryption for this purpose. We still use hybrid post-quantum TLS to secure the network channels. We justify our decision with the fact that only the encrypted `Assertion` on the user's local computer is not quantum-safe, which we believe is a practically minor risk.

Design. We use the separate hybrid XML signatures presented in Section 5.2.2 to implement our backward-compatible solution. Recall that the separate hybrid post-quantum XML signature creates two `Signature` elements, where the post-quantum `Signature` contains an additional step that removes the classical `Signature` during the verification process.

We define the signing process of a SAML SSO `AuthnRequest` as follows (similarly for `Response`); see Listing 2 for an illustration:

(1) The `AuthnRequest` is signed by the 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 with the attribute `subtract` as the XPath expression being `/saml2p:*/ds:Signature[1]`.[5]
(2) The `AuthnRequest` (with the post-quantum signature in `Extensions`) is signed by the classical algorithm and the `Signature` is placed as a child of `AuthnRequest`; its only transforms are the canonicalization and the enveloped signature transforms.

[5]Recall from Section 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, this attribute needs to be modified accordingly.

A classical SP or IdP (who does not support post-quantum signatures) can verify the resulting SAML message as follows. It verifies the classical signature, as this is compliant with the current 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 SAML libraries should only allow the transforms specified in the standard, since allowing any transform poses a security threat [67]. This means that the verifier will accept the hybrid post-quantum SAML message if and only if both signatures are valid.

Non-separability. Recall from Section 5.2.2 that our hybrid post-quantum XML signature is inseparable, which guarantees that an attacker cannot extract the classical `Signature` and use it as a standalone signature of `AuthnRequest`.

On the other hand, an attacker could extract the post-quantum `Signature` and try to use it as a standalone post-quantum XML signature of the `AuthnRequest`. However, you can tell that this signature was extracted from a hybrid XML signature because the additional `Transform` element is also signed. Therefore, a SAML-compliant verifier will reject this separated XML signature.

Experimental setup. As in the previous sections, we have implemented and tested the performance of our SAML SSO solution. We use the separate hybrid signatures and the hybrid XML public-key encryption for a direct comparison with the non-backward-compatible one. However, as mentioned before, using the hybrid XML PKE makes the protocol non-backward compatible.

Performance results. We present the results of our experiment in Table 9.

Let us now describe the performance differences between the backward-compatible versions and the non-backward-compatible versions (Section 7.2.1) that result from the different hybrid post-quantum XML signatures used.

For example, we can see that the backward-compatible solution is 17% slower than the non-backward-compatible solution for the combination of Dilithium with Kyber, and 12% slower for the combination of Falcon with Kyber. If BIKE is used instead of Kyber, the difference is 6%, and combinations with Classic McEliece are about 30% slower.

The messages in the backward-compatible solution are also larger because they contain two `Signature` elements instead of one. As a result, for example, the combination of Dilithium and Kyber uses 9% more bandwidth.

7.3 Discussion

We presented three solutions for post-quantum SAML.

The first and simplest solution is the purely post-quantum solution (Section 7.1). This solution has the advantage that it does not modify the SAML layer and only requires modifying the underlying libraries to support the post-quantum signatures and KEMs. It is also significantly faster and uses less bandwidth than the others because it does not use the additional classical algorithms. The drawbacks of this solution are that it does not have hybrid post-quantum security and it is not interoperable with classical-only parties.

The second solution is the non-backward-compatible hybrid solution that uses composite signatures (Section 7.2.1). It provides hybrid post-quantum security in terms of authentication/integrity and also confidentiality. Compared to the backward-compatible solution (see below), it does not change anything at the SAML layer, is faster, and uses less bandwidth. Its obvious disadvantage is that it is not backward compatible.

The third solution is the backward-compatible solution (Section 7.2.2). While we had to significantly modify the SAML processing layer for this method and it is the least efficient solution, it is the only one that provides backward compatibility, which is especially useful for the upcoming migration period (see Section 4.3). It should also be noted that the solution only provides classical security in terms of confidentiality of the SAML assertion.

8 CONCLUSION

With our work, we have laid the foundation for securing XML against future quantum attacks through the use of post-quantum cryptography. The particular focus of this work was on the post-quantum migration of the SAML single sign-on protocol. We intend to extend our study to other practically relevant XML-based security protocols in the future.

Through our extensive testing, we have shown which post-quantum schemes can be used to efficiently implement the cryptographic primitives in XML and the SAML SSO protocol, and which are less likely to do so. For practical reasons, we studied not only purely post-quantum versions, but also hybrid versions, which are particularly useful for the upcoming migration period.

However, we also found in this work that the current XML and SAML standards prevent more efficient post-quantum solutions, in particular by not directly supporting KEMs for public-key encryption and by doubling the symmetric layer in XML's super-encryption mode for nested encryption. We therefore advocate that the next revision of the XML and SAML standards should consider improving these aspects to enable a smooth migration to the post-quantum era.

ACKNOWLEDGMENTS

Johannes Müller was supported by the Luxembourg National Research Fund (FNR), under the CORE Junior project FP2 (C20/IS/14698166/FP2/Mueller). Jan Oupický was supported by the industrial partnership project between the interdisciplinary research center SnT and LuxTrust.

REFERENCES

[1] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. 2019. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process.* Technical Report NIST Internal or Interagency Report (NISTIR) 8240. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8240

[2] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. 2022. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process.*

Technical Report NIST Internal or Interagency Report (NISTIR) 8413. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8413-upd1

[3] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Info von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. 2022. Classic McEliece – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[4] Nouri Alnahawi, Johannes Müller, Jan Oupický, and Alexander Wiesmaier. 2023. SoK: Post-Quantum TLS Handshake. https://eprint.iacr.org/2023/1873

[5] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zemor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. 2022. BIKE – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[6] Auth0. 2024. Sign and Encrypt SAML Requests. https://web.archive.org/web/20240124152304/https://auth0.com/docs/authenticate/protocols/saml/saml-sso-integrations/sign-and-encrypt-saml-requests

[7] Daniel J Bernstein. 2009. Cost Analysis of Hash Collisions: Will Quantum Computers Make SHARCS Obsolete? (2009). https://cr.yp.to/hash/collisioncost-20090823.pdf

[8] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen (Eds.). 2009. *Post-Quantum Cryptography*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88702-7

[9] Ward Beullens. 2022. Breaking Rainbow Takes a Weekend on a Laptop. In *Advances in Cryptology – CRYPTO 2022 (Lecture Notes in Computer Science)*. Springer Nature Switzerland, 464–479. https://doi.org/10.1007/978-3-031-15979-4_16

[10] Ward Beullens. 2022. MAYO: Practical Post-quantum Signatures from Oil-and-Vinegar Maps. In *Selected Areas in Cryptography (Lecture Notes in Computer Science)*, Riham AlTawy and Andreas Hülsing (Eds.). Springer International Publishing, Cham, 355–376. https://doi.org/10.1007/978-3-030-99277-4_17

[11] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. 2019. Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In *Post-Quantum Cryptography*. Vol. 11505. Springer International Publishing, 206–226. https://doi.org/10.1007/978-3-030-25510-7_12

[12] Nina Bindel, Cas Cremers, and Mang Zhao. 2023. FIDO2, CTAP 2.1, and WebAuthn 2: Provable Security and Post-Quantum Instantiation. In *2023 IEEE Symposium on Security and Privacy (SP)*. 1471–1490. https://doi.org/10.1109/SP46215.2023.10179454

[13] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. 2017. Transitioning to a Quantum-Resistant Public Key Infrastructure. In *Post-Quantum Cryptography (Lecture Notes in Computer Science)*, Tanja Lange and Tsuyoshi Takagi (Eds.). Springer International Publishing, Cham, 384–405. https://doi.org/10.1007/978-3-319-59879-6_22

[14] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Request for Comments RFC 5280. Internet Engineering Task Force. https://doi.org/10.17487/RFC5280

[15] Gilles Brassard, Peter Hoyer, and Alain Tapp. 1998. Quantum Algorithm for the Collision Problem. Vol. 1380. 163–169. https://doi.org/10.1007/BFb0054319 arXiv:quant-ph/9705002

[16] BSI. 2023. *Cryptographic Mechanisms: Recommendations and Key Lengths*. Technical Report. Federal Office for Information Security, Germany. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=6

[17] Wouter Castryck and Thomas Decru. 2023. An Efficient Key Recovery Attack on SIDH. In *Advances in Cryptology – EUROCRYPT 2023 (Lecture Notes in Computer Science)*. Springer Nature Switzerland, 423–447. https://doi.org/10.1007/978-3-031-30589-4_15

[18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. 2018. CSIDH: An Efficient Post-Quantum Commutative Group Action. In *Advances in Cryptology – ASIACRYPT 2018 (Lecture Notes in Computer Science)*. Springer International Publishing, 395–427. https://doi.org/10.1007/978-3-030-03332-3_15

[19] ChartRequest. 2024. SSO: Single Sign-on / SAML. https://app.chartrequest.com/single-sign-on-detail

[20] Cisco Duo. 2024. Single Sign-On for Generic SAML Service Providers. https://web.archive.org/web/20240125125117/https://duo.com/docs/sso-generic

[21] Cloudflare. 2023. Generic SAML 2.0 · Cloudflare Zero Trust Docs. https://developers.cloudflare.com/cloudflare-one/identity/idp-integration/generic-saml/

[22] Eric Crockett, Christian Paquin, and Douglas Stebila. 2019. *Prototyping Post-Quantum and Hybrid Key Exchange and Authentication in TLS and SSH*. Technical Report 858. https://eprint.iacr.org/2019/858

[23] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. 2020. SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies. In *Advances in Cryptology – ASIACRYPT 2020 (Lecture Notes in Computer Science)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer International Publishing, Cham, 64–93. https://doi.org/10.1007/978-3-030-64837-4_3

[24] Morris Dworkin. 2007. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Technical Report NIST Special Publication (SP) 800-38D. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-38D

[25] Donald E. Eastlake 3rd. 2022. *Additional XML Security Uniform Resource Identifiers (URIs)*. Request for Comments RFC 9231. Internet Engineering Task Force. https://doi.org/10.17487/RFC9231

[26] Enterprise Healt. 2024. SAML-Based Single Sign-On. https://docs.enterprisehealth.com/functions/system-administration/security/saml-based-single-sign-on/

[27] ETSI. 2017. *Limits to Quantum Computing Applied to Symmetric Key Sizes*. Technical Report ETSI GR QSC 006. ETSI. https://www.etsi.org/deliver/etsi_gr/QSC/001_099/006/01.01.01_60/gr_QSC006v010101p.pdf

[28] ETSI. 2022. Electronic Signatures and Infrastructures (ESI); XAdES Digital Signatures; Part 1: Building Blocks and XAdES Baseline Signatures. https://www.etsi.org/deliver/etsi_en/319100_319199/31913201/01.02.01_60/en_31913201v010201p.pdf

[29] Federico Giacon, Felix Heuer, and Bertram Poettering. 2018. KEM Combiners. In *Public-Key Cryptography – PKC 2018 (Lecture Notes in Computer Science)*, Michel Abdalla and Ricardo Dahab (Eds.). Springer International Publishing, Cham, 190–218. https://doi.org/10.1007/978-3-319-76578-5_7

[30] Google Cloud. 2023. Single Sign-on | Cloud Architecture Center. https://web.archive.org/web/20240228134116/https://cloud.google.com/architecture/identity/single-sign-on

[31] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. https://doi.org/10.48550/arXiv.quant-ph/9605043 arXiv:quant-ph/9605043

[32] HL7. 2023. FHIR v5.0.0 - Digital Signatures. https://hl7.org/fhir/signatures.html

[33] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. 2022. SPHINCS+ – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[34] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Fiona Johanna Weber, and Philip R. Zimmermann. 2021. Post-Quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy (SP)*. 304–321. https://doi.org/10.1109/SP40001.2021.00030

[35] InCommon. 2024. InCommon Federation Info: Organizations. https://incommon.org/custom/federation/info/all-orgs.html

[36] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky. 2012. Bleichenbacher's Attack Strikes Again: Breaking PKCS#1 v1.5 in XML Encryption. In *Computer Security – ESORICS 2012 (Lecture Notes in Computer Science)*, Sara Foresti, Moti Yung, and Fabio Martinelli (Eds.). Springer, Berlin, Heidelberg, 752–769. https://doi.org/10.1007/978-3-642-33167-1_43

[37] Tibor Jager and Juraj Somorovsky. 2011. How to Break XML Encryption. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. Association for Computing Machinery, New York, NY, USA, 413–422. https://doi.org/10.1145/2046707.2046756

[38] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hitchinson. 2022. SIKE – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[39] Jakob Jonsson and Burt Kaliski. 2003. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. Request for Comments RFC 3447. Internet Engineering Task Force. https://doi.org/10.17487/RFC3447

[40] Panos Kampanakis, Peter Panburana, Ellie Daw, and Daniel Van Geest. 2018. *The Viability of Post-quantum X.509 Certificates*. Technical Report 063. http://eprint.iacr.org/2018/063

[41] Jonathan Katz and Yehuda Lindell. 2020. *Introduction to Modern Cryptography*. CRC Press.

[42] Dennis Kupser, Christian Mainka, Jörg Schwenk, and Juraj Somorovsky. 2015. How to Break XML Encryption - Automatically. In *Proceedings of the 9th USENIX Conference on Offensive Technologies (WOOT'15)*. USENIX Association, USA, 11.

[43] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. 2022. CRYSTALS-Dilithium – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[44] John Mattsson, Ben Smeets, and Erik Thormarker. 2021. Quantum Technology and Its Impact on Security in Mobile Networks. *Ericsson Technology Review* (2021). https://www.ericsson.com/4ae3c7/assets/local/reports-papers/ericsson-technology-review/docs/2021/ensuring-security-in-mobile-networks-post-quantum.pdf
[45] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. 2022. HQC – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
[46] Microsoft. 2023. Plan a Single Sign-on Deployment - Microsoft Entra ID. https://web.archive.org/web/20240224062457/https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/plan-sso-deployment
[47] Microsoft. 2023. SAML Token Encryption - Microsoft Entra ID. https://web.archive.org/web/20240124152123/https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/howto-saml-token-encryption?tabs=azure-portal
[48] miniorange. 2024. WordPress SSO for Healthcare & Medical Institutes. https://plugins.miniorange.com/wordpress-single-sign-on-sso-for-healthcare-medical-institutes
[49] NIST. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.rip/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf
[50] NIST. 2022. Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf
[51] NSA. 2022. *Commercial National Security Algorithm Suite 2.0.* Technical Report. https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF
[52] OASIS. 2012. SAML V2.0 Specification. https://web.archive.org/web/20230118004327/https://wiki.oasis-open.org/security/FrontPage
[53] Okta. 2024. Customization Options for Inbound SAML. https://web.archive.org/web/20240124151537/https://help.okta.com/en-us/content/topics/security/idp-inbound-saml-reference.htm
[54] OneLogin. 2024. OneLogin Developers - SAML Response. https://web.archive.org/web/20240124151314/https://developers.onelogin.com/saml/examples/response
[55] Open Quantum Safe project. 2023. Liboqs Documentation - Falcon. https://github.com/open-quantum-safe/liboqs/blob/2f4a25c8de89b2bb98607f24b2f401856482737e/docs/algorithms/sig/falcon.md
[56] Mike Ounsworth, John Gray, Massimiliano Pala, and Jan Klaußner. 2023. *Composite Signatures for Use in Internet PKI.* Internet-Draft draft-ounsworth-pq-composite-sigs-10. Internet Engineering Task Force / Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ounsworth-pq-composite-sigs/10/
[57] Mike Ounsworth, Aron Wussler, and Stavros Kousidis. 2024. *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. https://datatracker.ietf.org/doc/draft-ounsworth-cfrg-kem-combiners/05/
[58] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2022. Falcon – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
[59] Saptarshi Purkayastha, Judy W Gichoya, and Abhishek Siva Addepally. 2017. Implementation of a single sign-on system between practice, research and learning systems. *Applied clinical informatics* 26, 01 (2017), 306–312.
[60] Eric Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3.* Request for Comments RFC 8446. Internet Engineering Task Force. https://doi.org/10.17487/RFC8446
[61] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. 2022. CRYSTALS-Kyber – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
[62] ServiceNow. 2023. Multi-Provider Single Sign-on - SAML. https://docs.servicenow.com/bundle/vancouver-platform-security/page/integrate/saml/concept/c_SAML2.0WebBrowserSSOProfile.html
[63] Shibboleth Consortium. 2024. Shaping the Future of Shibboleth Software. https://www.shibboleth.net/
[64] P.W. Shor. 1994. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 124–134. https://doi.org/10.1109/SFCS.1994.365700
[65] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. 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, New York, NY, USA, 149–156. https://doi.org/10.1145/3386367.3431305
[66] SOG-IS Crypto Working Group. 2023. Agreed Cryptographic Mechanisms. https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf
[67] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. 2012. On Breaking SAML: Be Whoever You Want to Be. In *Proceedings of the 21st USENIX Conference on Security Symposium (Security'12)*. USENIX Association, USA, 21.
[68] D. Stebila, S. Fluhrer, and S. Gueron. 2023. Internet-Draft: Hybrid Key Exchange in TLS 1.3. https://www.ietf.org/archive/id/draft-ietf-tls-hybrid-design-06.html
[69] Douglas Stebila and Michele Mosca. 2016. Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project. In *Selected Areas in Cryptography – SAC 2016 (Lecture Notes in Computer Science)*, Roberto Avanzi and Howard Heys (Eds.). Springer International Publishing, Cham, 14–37. https://doi.org/10.1007/978-3-319-69453-5_2
[70] The Apache Santuario projet. 2024. Apache Santuario Library. https://santuario.apache.org/
[71] The Legion of the Bouncy Castle. 2024. Bouncy Castle Library. https://www.bouncycastle.org/
[72] The OpenSAML project. 2024. OpenSAML Library. https://shibboleth.atlassian.net/wiki/spaces/OSAML/overview
[73] Maran van Heesch, Niels van Adrichem, Thomas Attema, and Thijs Veugen. 2019. Towards Quantum-Safe VPNs and Internet. https://eprint.iacr.org/2019/1277
[74] W3C. 2000. Simple Object Access Protocol (SOAP) 1.1. https://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[75] W3C. 2010. XML Path Language (XPath) 2.0 (Second Edition). https://www.w3.org/TR/xpath20/
[76] W3C. 2012. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. https://www.w3.org/TR/xmlschema11-1/
[77] W3C. 2013. XML Encryption Syntax and Processing Version 1.1. https://www.w3.org/TR/xmlenc-core1/
[78] W3C. 2013. XML Signature Best Practices. https://www.w3.org/TR/2013/NOTE-xmldsig-bestpractices-20130411/
[79] W3C. 2013. XML Signature Syntax and Processing Version 1.1. https://www.w3.org/TR/xmldsig-core1/
[80] W3C. 2017. XSL Transformations (XSLT) Version 3.0. https://www.w3.org/TR/xslt-30/
[81] Bas Westerbaan. 2022. NIST's Pleasant Post-Quantum Surprise. http://blog.cloudflare.com/nist-post-quantum-surprise/
[82] Christof Zalka. 1999. Grover's Quantum Searching Algorithm Is Optimal. *Physical Review A* 60, 4 (Oct. 1999), 2746–2751. https://doi.org/10.1103/PhysRevA.60.2746

A DETAILED PERFORMANCE RESULTS

In this section, we collect all of the detailed performance results that we summarized in the main body, but did not have room to include in the main body due to page limitations. See Tables 8 and 9.

B CONSTRUCTING A PKE FROM A KEM

There is a well-known construction of a public key encryption scheme from a key encapsulation mechanism and a symmetric cipher. It can be found in [41], but we also present it here for the convenience of the reader.

Let $K = (K.\mathsf{Gen}, K.\mathsf{Encaps}, K.\mathsf{Decaps})$ be a KEM, let $S = (S.\mathsf{Gen}, S.\mathsf{Enc}, S.\mathsf{Dec})$ be a symmetric cipher and let $\lambda \in \mathbb{N}$ be the security parameter. We require that the key returned by the encapsulation mechanism is a valid key for S.

From these two primitives we construct a public key encryption scheme $P = (P.\mathsf{Gen}, P.\mathsf{Enc}, P.\mathsf{Dec})$ as follows

- $P.\mathsf{Gen}(1^\lambda)$ runs $K.\mathsf{Gen}(1^\lambda) \rightarrow (\mathrm{pub}_K, \mathrm{priv}_K)$ and returns $(\mathrm{pub}_P, \mathrm{priv}_P) = (\mathrm{pub}_K, \mathrm{priv}_K)$, i.e., the PKE keypair is the KEM keypair.
- $P.\mathsf{Enc}(m, \mathrm{pub}_P)$ runs $K.\mathsf{Encaps}(\mathrm{pub}_P) \rightarrow (c_K, k)$ and then runs $S.\mathsf{Enc}(m, k) \rightarrow c_S$. The output ciphertext is $c_P \leftarrow (c_K, c_S)$.
- $P.\mathsf{Dec}(c_P, \mathrm{priv}_P)$ runs $K.\mathsf{Decaps}(c_K, \mathrm{priv}_P) \rightarrow k$ and then $S.\mathsf{Dec}(c_S, k) \rightarrow m$ where $(c_K, c_S) \leftarrow c_P$. The output is the plaintext m.

Table 8: Efficiency of post-quantum algorithms

Algorithm	Primitive	Security level	Public key size	Signature/ciphertext size	Sig./s	Verif./s	Encaps./s	Decaps./s
RSA	Sig. & KEM	1 (3072 bits)	387	384	113	6 842	6 712	112
		5 (15360 bits)	1 923	1 920	1	313	311	1
ECDSA	Signature	1 (P-256)	32	64	22 526	6 591	N/A	N/A
		5 (P-521)	66	132	197	266		
Dilithium	Signature	2	1 312	2 420	5 156	15 829	N/A	N/A
		5	2 592	4 595	2 572	5 592		
Falcon	Signature	1	897	666	4 232	25 382	N/A	N/A
		5	1 793	1 280	2 090	13 196		
SPHINCS+	Signature	1	32	7 856	2	1 626	N/A	N/A
		5	64	29 792	1	726		
Kyber	KEM	1	800	768	N/A	N/A	50 000	55 556
		5	1 568	1 568			21 739	23 255
BIKE	KEM	1	1 540	1 572	N/A	N/A	3 676	210
		5	5 122	5 154			498	27
HQC	KEM	1	2 249	4 481	N/A	N/A	4 405	2 976
		5	7 245	14 469			965	667
Cl. McEliece	KEM	1	261 120	128	N/A	N/A	16 129	46
		5	1 044 992	240			4 545	11

Sizes are in bytes. Security level refers to the claimed NIST security level for the specific parameter set. Source for sizes: [2], Section D with the exception of Falcon and Cl. McEliece, where data is from [55] as values in [2] seem to be incorrect. Speed performance measured on Oracle Cloud VM.Standard.A1.Flex, Open Quantum Safe OpenSSL fork [69] (OpenSSL 3.2.1, liboqs 0.9.2, oqsprovider 0.5.3, KEM speeds were measured using provided liboqs's `speed_kem` command). SPHINCS+ parameters: SHA256, 128s/256s, simple.

Table 9: Performance of backward compatible hybrid post-quantum SAML SSO

Sig. 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
+ Dilithium (2)	+ Kyber (1)	43 559 (× 4.59)	13.68 ± 0.50 (×1.20)	4.21 ± 0.66 (×2.55)	17.89 ± 1.04 (×1.37)
+ Falcon (1)	+ Kyber (1)	27 718 (× 2.92)	13.74 ± 0.34 (×1.21)	4.18 ± 0.31 (×2.53)	17.92 ± 0.49 (×1.38)
+ Dilithium (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)
+ Falcon (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)
+ Dilithium (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)
+ Falcon (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, SPHINCS+ parameters: SHA256, 128s/256s, simple. RSA-OAEP with MGF1-SHA1 [39].

It is well known that such a construction achieves IND-CCA security iff the KEM is IND-CCA secure and the symmetric cipher is IND-CCA secure [41]. All post-quantum KEMs in the NIST competition are IND-CCA secure. Any authenticated encryption scheme, such as AES-GCM, is IND-CCA secure.

C QUANTUM SECURITY OF SYMMETRIC PRIMITIVES

In this section, we discuss the quantum security of the symmetric ciphers and hash functions which are used in XML signatures [79] and XML encryption [77].

C.1 Symmetric ciphers

XML encryption supports AES with 128-bit, 192-bit or 256-bit keys, as well as 3DES, which has 168-bit keys.

Grover's algorithm [31] achieves a quadratic speedup over classical brute-force key search, i.e., it takes "only" 2^{64} operations on quantum computer to recover a 128-bit key instead of the 2^{128} operations on a classical computer.

However, quantum operations are different from classical operations and there is no known way how to effectively parallelize Grover's algorithm [82]. For this reason, some organizations [16, 44, 49, 81] still consider 128-bit ciphers to be secure even against quantum adversaries.

Nevertheless, the conservative option is to double the key size as recommended by ETSI [27] or the NSA [51].

Conclusion. In our implementation, we use AES-256-GCM, which we also recommend for adopters of post-quantum XML encryption as it is provides a conservative security and the performance cost is typically negligible in the context of SAML or other XML-based protocols.

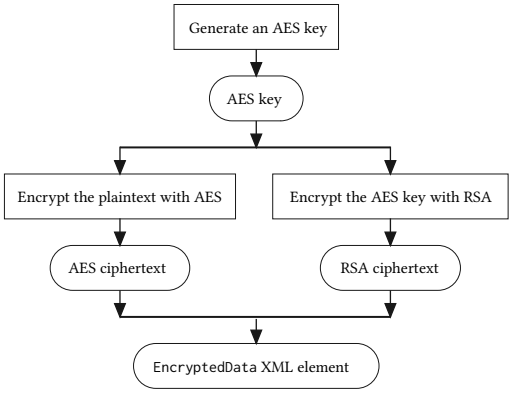

Figure 4: XML encryption process.

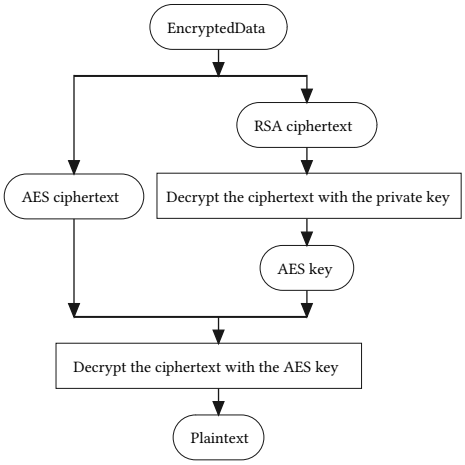

Figure 5: XML decryption process.

C.2 Hash functions

Hash functions are mainly used within XML signatures [79] to hash the document before applying the signature, but they are also internal components of signatures. The XML signature standard supports MD5, SHA-1, SHA-2, SHA-3 and Whirpool [25, 79]. It is well known that MD5 and SHA-1 should not be used as they are insecure.

The analysis of the quantum security of hash functions is basically the same as for symmetric ciphers because all quantum speedups are also based on Grover's algorithm [31].

Pre-image resistance. Finding a pre-image of a hash function (with an output size of n bits) by brute force requires 2^n classical operations, Grover's quantum algorithm can reduce this complexity to $2^{\frac{n}{2}}$.

Collision resistance. Finding a collision requires on average $2^{\frac{n}{2}}$ classical operations but the BHT quantum algorithm [15] can reduce the complexity to $2^{\frac{n}{3}}$. Note that the practicality of the BHT algorithm is questionable as, according to [7], the BHT algorithm has "fundamentally worse price-performance ratio" than classical collision search algorithms.

Conclusion. Any cryptographic hash function with an output size of ≥ 256 bits is considered quantum-secure. In our implementation we use SHA2-512 for all reference digests. Ideally, we recommend to use SHA2-256 or SHA3-256 with signatures that claim security levels 1 or 2 and SHA2-512 or SHA3-512 with signatures that claim security levels > 2.

This is because NIST security levels 2 and 4 are defined as the quantum security of 256-bit and 512-bit hash functions, so it does not make sense to use, e.g., SHA2-256 with security level 5 Dilithium as the hash function will be the weakest link and the attacker will only need to find a collision to forge an XML signature.

D CONSTRUCTING HYBRID POST-QUANTUM SIGNATURES

We present a simple construction of a hybrid post-quantum signature from a classical signature and a post-quantum signature.

Let $C = (C.\text{Gen}, C.\text{Sign}, C.\text{Verif})$ be a classical signature and let $PQ = (PQ.\text{Gen}, PQ.\text{Sign}, PQ.\text{Verif})$ be a post-quantum signature.

We assume the *Verif* operation returns 1 if signature is valid and 0 otherwise. Let $\lambda \in \mathbb{N}$ be the security parameter.

Let $e : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an encoding function and $d : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be its corresponding decoding function, i.e., $\forall x, y \in \{0,1\}^* : d(e(x,y)) = (x,y)$.

We define the hybrid post-quantum signature $H = (H.\text{Gen}, H.\text{Sign}, H.\text{Verif})$ as follows

- $H.\text{Gen}(1^\lambda)$ runs $C.\text{Gen}(1^\lambda) \to (\text{pub}_C, \text{priv}_C)$, $PQ.\text{Gen}(1^\lambda) \to (\text{pub}_{PQ}, \text{priv}_{PQ})$ and returns $(\text{pub}_H, \text{priv}_H)$ where $\text{pub}_H \leftarrow e(\text{pub}_C, \text{pub}_{PQ}), \text{priv}_H \leftarrow e(\text{priv}_C, \text{priv}_{PQ})$.
- $H.\text{Sign}(m, \text{priv}_H)$ decodes the private key $(\text{priv}_C, \text{priv}_{PQ}) \leftarrow d(\text{priv}_H)$, runs $C.\text{Sign}(m, \text{priv}_C) \to \sigma_C$ and $PQ.\text{Sign}(m, \text{priv}_{PQ}) \to \sigma_{PQ}$ and returns σ_H where $\sigma_H \leftarrow e(\sigma_C, \sigma_{PQ})$.
- $H.\text{Verif}(m, \sigma_H, \text{pub}_H)$ decodes the public key $(\text{pub}_C, \text{pub}_{PQ}) \leftarrow d(\text{pub}_H)$, decodes the signature $(\sigma_C, \sigma_{PQ}) \leftarrow d(\sigma_H)$ and returns $C.\text{Verif}(m, \sigma_C, \text{pub}_C) \cdot PQ.\text{Verif}(m, \sigma_{PQ}, \text{pub}_{PQ})$, i.e., the signature is valid if and only if both component signatures are valid.

The simplest encoding may be concatenation, if the classical and post-quantum signatures/keys have a fixed size so that they can be successfully decoded. More robust constructions have been proposed, such as [56] (which we have implemented and used in our solutions).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="abcd" Type="http://www.w3.org/2001/04/xmlenc#Element">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#aes256-gcm" />
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <xenc:EncryptedKey Id="bbcd">
            <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            </xenc:EncryptionMethod>
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate><!-- base64 encoded RSA certificate --></ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
            <xenc:CipherData>
                <xenc:CipherValue><!-- base64 encoded encrypted key --></xenc:CipherValue>
            </xenc:CipherData>
        </xenc:EncryptedKey>
    </ds:KeyInfo>
    <xenc:CipherData>
        <xenc:CipherValue><!-- base64 encoded encrypted data --></xenc:CipherValue>
    </xenc:CipherData>
</xenc:EncryptedData>
```

Listing 1: XML public key encryption example

```
<!-- 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>
            <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 2: Example of a backward compatible SAML `AuthnRequest`