Literature Overview of Public-Key Infrastructures, with Focus on Quantum-Safe Variants
Deliverable 4.1, HAPKIDO Project

Alessandro Amadori, João Diogo Duarte, Gabriele Spini

TNO

March 31, 2022

Abstract

This document provides an overview of the core concepts, usage, applications, and normative frameworks of public-key infrastructures. Moreover, special attention is devoted to the state-of-the-art research on quantum-safe public-key infrastructures, and to hybrid solutions that safeguard compatibility with legacy systems and that remain secure as long as at least one between their classical and their quantum-safe component is secure.

1 Management Summary

This document forms a deliverable for the HAPKIDO (Hybrid quantum-safe Public-Key Infrastructure Development for Organisations) project. Currently-used PKIs (Public-Key Infrastructures) are presented in the first part of the document, while the second part is devoted to a literature overview of quantum-safe PKIs.

The document takes a fairly broad approach, and also discusses cryptographic protocols which do not, strictly speaking, fall under PKIs, but that do use PKIs to manage their keys. This is motivated by the fact that a transition to quantum-safe PKIs cannot be done without taking into account the impact on the systems that rely on PKIs.

In terms of quantum-safe PKIs, a few proposals have focused on the X.509 standard, which is currently the most widely used certificate format for PKIs; these proposals describe how to extend the X.509 certificate format (and associated functions) to a hybrid quantum-safe variant.

Several widely-used protocols make use of PKIs, such as TLS, SSH and S/MIME. TLS is used for secure communication, SSH is used to operate network services securely over an unsecured network and S/MIME is used to encrypt and sign mails. With respect to the migration of these protocols to quantum-safe alternatives, TLS has received most attention, whereas SSH has been investigated to a lesser extent and S/MIME has only been addressed sporadically.

We also make the remark that some quantum-safe implementations of functionalities such as Virtual Private Network (VPNs) have been proposed, but with no accompanying publications. These are reported on but not fully analysed here.

2 Introduction

Modern digital societies, and the Internet in general, are currently extremely dependent on *asymmetric* cryptography, also known as public-key cryptography. This branch of cryptography, founded in the mid-seventies, greatly simplifies communication of keys and enables entirely new functionalities such as digital signatures and key exchange, compared to the exclusive usage of symmetric cryptography. The conceptual difference between these two types of cryptography is that asymmetric cryptography uses two different keys — a public key and a private key — whereas symmetric cryptography uses only one shared key.

However, management of keys of asymmetric cryptographic systems is a complex task, arguably made of capital importance by the large number of digital systems that rely on asymmetric cryptography nowadays. Keys of asymmetric cryptographic systems are typically managed by complex systems known as *public-key infrastructures,* or *PKIs* for short. PKIs can be seen as ecosystems comprising entities, functionalities and cryptographic specifications, which ensure the security of private keys and the authenticity and integrity of public keys.

The security of asymmetric cryptography, however, faces a threat posed by the looming availability of large-scale quantum computers. As showed by Shor in 1994 [62], such a device would be able to break most currently-used asymmetric systems, by efficiently solving the computational problems whose hardness is supposed to guarantee the security of the system. Symmetric cryptography is also affected by quantum computing, but to a much less severe degree, and extension of key length is believed to be sufficient to guarantee security in this setting.

While no large-scale quantum computer exists nowadays, progress is fast and such a device could be available within one or two decades [41]. While this might seem like a reasonable amount of time, experience shows that migration of widely-used cryptographic schemes is an extremely lengthy and complex process: previous and less invasive migrations like those to elliptic-curve cryptography and to new hash functions took a long time to be fully integrated. Furthermore, so-called "store-now, decrypt-later" attacks mean that a migration will have to be realised much earlier than the advent of a large-scale quantum computer for some specific sectors.

PKIs arguably form a challenging, yet extremely important part of this migration to quantum-safe cryptography, given their ubiquity and the large number of roles and parties that govern them.

This document is meant to provide an overview of state-of-the-art PKI systems and on previous research on quantum-safe PKIs. Particular attention is devoted to research on *hybrid*[1] systems, which support both classical (i.e., non-quantum safe) and quantum-safe cryptography. This is a highly desirable property, since a "big-bang" migration, occurring in a single step, to quantum-safe PKIs does not seem realistic given the high number of parties and legacy systems involved; therefore, safeguarding compatibility with legacy systems becomes necessary. Hybrid systems provide compatibility by their ability to switch off or ignore the quantum-safe part when interacting with legacy systems. Moreover, hybrid systems offer strongest-link security, meaning that they are secure as long as one of their two components is secure. This is a very desirable property as well, given quantum-safe solution have received less analysis and attention compared to classical ones: their exact security level and the necessary parameters to achieve these levels are thus less understood, and relying on quantum-safe systems alone could pose security risks.

Two types of solutions exists nowadays to secure asymmetric cryptography against quantum computers: those that rely on effects of quantum mechanics, and those that do not, and can therefore run on existing information-technology systems. The first one is referred to as quantum cryptography, and revolves around the concept of quantum-key distribution or QKD for short, while the second is called *post-quantum cryptography.* The focus of this document is on the latter, given that the HAPKIDO project aims to be compatible with current systems, and given that QKD does not form a direct replacement for asymmetric cryptography; consequently,

[1]We make the remark that the term "hybrid" is sometimes used with a different meaning in the context of cryptography, indicating systems that uses both asymmetric and symmetric cryptography.

we understand "quantum-safe" to mean "using post-quantum cryptography" in the rest of the document.

This overview document also discusses the systems that make use of PKIs to manage the associated keys, since these have to be taken into account as well when considering a migration to quantum-safe PKIs.

The rest of this document is organised as follows. Section 3 briefly presents the cryptographic concepts that are needed to read this document, and provides references for further reading. Section 4 introduces basic PKI notions. Section 5 discusses important protocols that make use of PKIs to manage keys and identities; applications of these protocols are presented in Section 6. Finally, Section 7 presents current state-of-the-art work on quantum-safe PKIs and associated protocols and applications.

3 Cryptographic Prerequisites

In order to make this document self-contained, this section provides a brief overview of the cryptographic notions needed to understand PKIs. Notice that this overview is kept on a succinct, high-level form; the reader can refer to, e.g., [38, 30] for a more complete and formal discussion of these concepts.

3.1 Symmetric Cryptography

Symmetric cryptography plays an important role in PKIs and their application. As mentioned in the introduction, an increase in key length is believed to be sufficient to guarantee security of symmetric cryptography against large-scale quantum computers; this increase is relatively modest and is therefore generally manageable by protocols and hardware that run symmetric cryptographic systems.

Ciphers. Symmetric encryption systems are typically called *ciphers.* Ciphers use only one key, which allows both to *encrypt,* i.e., turn a plaintext into a scrambled ciphertext, and to *decrypt,* which turns a ciphertext back into the original plaintext. There are various security notions for these ciphers that describe which types of attacks the cipher is secure against.

Hash functions. While they do not, strictly speaking, have an associated key at all, at least in the basic meaning of the concept, *(cryptographic) hash functions* are typically listed under symmetric cryptography. Hash functions take as input a plaintext of arbitrary length and turn it into a short message of fixed length called *digest.* The process is irreversible: it is unfeasible to recover the original message when given a digest (for an appropriate parameter choice). Furthermore, it is also unfeasible to find collisions, i.e., pairs of distinct plaintexts that lead to the same digest.

3.2 Asymmetric Cryptography

We list below two fundamental concepts from asymmetric cryptography. As mentioned in the introduction, virtually all currently-used asymmetric systems are vulnerable to attacks from large-scale quantum computers, and need to be replaced by quantum-safe alternatives.

Encryption Schemes and Key-Encapsulation Methods. An *asymmetric encryption scheme* provides a similar functionality as a symmetric cipher, but works with two keys. One, the public key, can be freely disseminated and only enables encryption of plaintexts. The other one, the private (or "secret") key, should be protected from unauthorized access and enables decryption of ciphertexts, thereby recovering the original message.

Since encryption and decryption of asymmetric schemes are typically quite inefficient when compared to symmetric ciphers, it is common to use asymmetric encryption to only encrypt the key material of a symmetric cipher, and then to use the cipher to encrypt data instead. Systems that describe the details of this process are called *key-encapsulation mechanisms* or *KEMs* for short.

Digital Signature Schemes. In contrast to encryption schemes, *digital signatures* do not have a natural symmetric counterpart. Digital signature schemes use a private key to associate to any given message a short digital element known as the digital signature. With the public key, which once again can be freely disseminated, users can verify validity of the signature. The security of a signature scheme dictate that it should be unfeasible to produce a valid signature (i.e., that will be accepted by a given public key) without knowledge of the associated private key. This property is referred to as *unforgeability.*

4 General PKI Notions

This section gives an overview of the core notions of PKIs, elaborating on the functionalities achieved by PKIs, on the involved parties, and on relevant certificate systems. Notice that this section does not aim to provide accurate definitions of these concepts; the reader can refer to existing introductory work [7] to this end.

4.1 Basic Functionalities and Trust Models

PKIs can be seen as ecosystems consisting of functionalities, entities, and procedures that enable the creation, distribution, verification, and management of keys of asymmetric systems. The core goal of PKIs is to ensure that users can obtain the public key of a given entity and be sure that this public key indeed belongs to the entity, and that is valid. After this check, the public key can then be used for its corresponding cryptographic functionality (encryption/verification of a digital signature).

PKIs also address other aspects of the key management of asymmetric systems; the most important examples are the following:

- PKIs ensure that users obtain certified information on how to use a given public key. This information includes the cryptographic scheme the key belongs to and its parameters, the validity period of the key, and possible restrictions on its usage.
- PKIs ensure that public keys that are compromised (or anyway no longer fit for usage) get no longer accepted by users.
- PKIs describe how both public and private keys should be created, in terms of the involved software and hardware, and how private keys should be stored and used while keeping them private.
- PKIs enable authentication of users in access-control mechanisms.

As stated before, PKIs can be seen as *ecosystems,* in the sense that they consist of the *functionalities* (intended as computer protocols) that are used to achieve the above goals, of the *entities* that use these functionalities, and of the *regulations* and procedures that describe which functionalities should be used by which identity, for which purpose.

We give more details on how PKIs achieve the above goals in the following sub-sections. Since management of private keys and of public keys are quite orthogonal to each other, we discuss them separately.

4.1.1 Private-Key Management

Private keys need to be maintained throughout their entire life cycle, from creation to their eventual archival or destruction. The procedures that describe how to perform all the necessary maintenance steps are typically understood to be part of PKIs, and we therefore present here these procedures.

Once again, the main goal of private-key management is to ensure that private keys remain private. To this end, special measure need to be taken, either by making use of specific software functionalities known as *Personal Security Environments,* or *PSEs* for short, or by resorting to specific hardware.

PSEs typically protect private keys by ensuring that only users possessing a certain personal identification number, or PIN, can access private keys. A common standard for this is given by PKCS#12 [48].

Examples of hardware to protect private keys are smart cards and hardware security modules. Both these systems exclusively manage cryptographic operations involving private keys, and prevent said keys from being accessed by systems that are interfaced with the hardware, e.g., smart card readers and computers connected to, or containing, hardware security modules.

4.1.2 Public-Key Management

In asymmetric cryptography, private keys do not need to be communicated in order to establish, for instance, secure communication channels or in order to verify digital signatures: only public keys need to be exchanged to this purpose, and these keys do not need to remain hidden from eavesdroppers. This forms one of the main advantages of asymmetric cryptography.

However, it remains necessary to ensure that the exchanged public keys are *authentic* and *valid.* Authenticity means that the public key which is claimed to belong to a given entity does indeed belong to that entity. Validity is a term with a wide meaning that encompasses, among other, assurance that the public key can perform the desired functionality, that it has not expired, and that it is fulfils the requirements of a given task (e.g., that is of sufficient length to guarantee security), where such a task is also specified and validated by a PKI. Furthermore, users must be sure that key properties have not been altered.

PKIs ensure the above goals in several ways, which will be described later. Furthermore, PKIs also provide a system to revoke, or anyway manage, keys which become insecure, for instance following a security breach or after a given expiration time window.

4.1.3 Trust Models

As discussed above, PKIs are essential for the usage of asymmetric cryptography. This subsection further explains *how* PKIs achieve this goal, focusing in particular on the establishment of trust in the authenticity and validity of public keys; we say that a user *trusts* the public key of an entity if they are convinced that that public key does indeed belong to that entity, *and* if they accept to use that key for its corresponding cryptographic protocol.

Several methods, with different degrees of complexity, are used for this purpose.

Direct trust. Direct trust is a first and very simple method: public keys are simply exchanged in a way which is deemed impervious to attacks or manipulations to malicious entities; users thus "directly" trust a given key-pair or entity. This is the case, for instance, for keys included in installation discs of Linux distributions, or for public keys included in web browsers.

The applications of direct trust are actually fairly limited, due to the difficulty of exchanging keys in a secure way; for this reason, more complex trust methods, described below, complement it nowadays.

Hierarchical trust. Hierarchical trust is a very widely used paradigm to establish trust in PKIs. Within this paradigm, keys are verified by some specific entities, known as *certificate authorities* or *CAs* for short, which typically also accept legal liability for this duty. CAs certify that keys are valid and belong to a given entity by issuing a *certificate;* we will elaborate on certificate purposes and structure in the following section, but the crucial point here is that a certificate is digitally signed with the private key of the CA, and that a certificate's validity can therefore be checked with the public key of the CA.

As an example, assume that a user obtains a public key $\mathtt{pk}_{\mathrm{A}}$ allegedly belonging to an entity Alice, and that they wish to check this. We assume that there is a CA that the user "trusts", meaning that they possess the CA public key $\mathtt{pk}_{\mathrm{CA}}$, are convinced of its authenticity, and are therefore willing to use it for its corresponding task, namely checking validity of certificates issued by the CA. If the user is provided by a certificate signed with the private key of the CA, which states that $\mathtt{pk}_{\mathrm{A}}$ does indeed belong to Alice, they can then use the CA public key $\mathtt{pk}_{\mathrm{CA}}$ to check the validity of this signature and thus be convinced of the authenticity of Alice's public key.

Now this procedure does generate a chicken-and-egg problem, since users still need to trust the public key of CAs. For some CAs, this is obtained by direct trust; we then speak of *root* CA. However, given that direct trust is difficult to obtain (due to the reason discussed above), users typically only trust a very limited number of CAs; moreover, for practical reasons, CAs can only certify limited numbers of public keys. Therefore, trust is also established in a different way, which justifies the usage of the name "hierarchical": CAs issue certificates that validate the public keys of "lower-level" CA certificates. Users thus get a chain of certificates that validate each validation step, and which ends in a certificate issued by a root CA.

An important remark is that when a CA issues a certificate for another CA, they also imply that they can vouch for the honest behaviour of said CA, and accept legal liability for it.

Web of trust. An alternative to the hierarchical trust, which nevertheless has nowadays a niche role, is given by the web of trust. This concept was introduced by Phil Zimmermann for the PGP (Pretty Good Privacy) system [74]. On the web of trust, a key-pair is trusted by a user it is either directly obtained from its owner or if it signed by sufficiently many users that are deemed trusted.

Typically, signers on the web of trust do not accept liability, which explains why most modern PKIs rely on hierarchical trust instead.

4.2 PKI Certificates

A fundamental tool in PKIs to establish trust is given by *certificates.* Certificates confirm the authenticity and validity of public keys, certify identities of users, and dictate how asymmetric keys are supposed to be used in order to establish secure communication.

A certificate is a digital record containing, among other, the following information:

- The identity of the issuer of the certificate;
- The identity of the subject of the certificate, i.e., of the entity the public key is confirmed to belong to;
- The public key associated to the subject;
- The cryptographic algorithm which is supposed to be used with the public key;
- The serial number, validity period and (optional) restrictions in the usage of the certificate.

Crucially, the issuer of a certificate *signs* it and appends the signature to the certificate. In this way, if the issuer is deemed trustworthy based on a certain trust model(cf. previous section), then we can use the issuer's public key to verify that the digital signature of the certificate is valid. If this is the case, then the subject of the certificate and the public key associated with the subject can also be trusted due to the unforgeability of the digital-signature scheme.

While several certificate formats exist, the most widely used is, by far, the X.509 format, specified in the ITU-T standard with same reference code [27].

5 Protocols that use PKIs

As discussed above, PKIs allow users to establish trust in electronic information systems. This section presents the most widely used cryptographic protocols for which PKIs establish the key infrastructure needed to securely run cryptographic applications. This is not meant to explain how PKIs are used in these protocols as this will be accomplished when discussing the use-cases. In this case, we define use-cases and a scenario in which one or more of these protocols are chosen and configured to meet the use-case's requirements. Note that some technical knowledge of basic networking protocols such as TCP and UDP are needed. For information on these protocols, please refer to resources such as [16].

5.1 Transport Layer Security (TLS)

The Transport Layer Security (TLS) is a TCP-based security protocol that ensures the authentication, confidentiality, and integrity of communication over the Internet given a reliable link between the communicating parties. It is maintained by the Internet Engineering Task Force (IETF) and, at the time of writing, the most recent version is TLS 1.3, which is standardized in RFC 8446 [56]. For further technical details of this protocol, please refer to the RFC. As per its name, it operates in the Transport Layer of the OSI model.

TLS is meant to be used as a building block in various applications as TLS is application independent. Hence, TLS is widely used in a variety of settings ranging from HTTPS, secure email all the way to smartcards and satellite communication.

TLS can be split into two sub-protocols, namely the handshake protocol and the record protocol. Essentially, the handshake protocol authenticates the parties, negotiates the cryptographic parameters and at the end, establishes session keys. These parameters include algorithms, key-sizes and shared secrets.

After the handshake protocol has terminated, the parties can start securely communicating using the cryptographic primitives that they agreed on in the handshake protocol. This is done using the record protocol, which cuts the messages into chunks and these chunks are then encrypted using the established session keys and sent to the other party. The encryption (and data integrity) is achieved by utilizing Authenticated Encryption with Associated Data (AEAD).

The output of decrypting via AEAD is either the plaintext or an error that indicates that either the decryption or authentication failed.

TLS defines a list of cryptographic algorithms that it supports, and this list differs from version to version. For example, TLS 1.3 dropped the support for many legacy ciphers, such as RC4 and DSA.

5.2 QUIC

QUIC is a UDP-based stateful and secure transport protocol [28]. In this case, secure is defined as providing the confidentiality, integrity, authenticity, and availability of the traffic. It operates on the Transport Layer of the OSI model. It is maintained by the Internet Engineering Task Force (IETF) and is standardized in RFC 9000 [28]. For further technical details of this protocol, please refer to the RFC. Its aim is to address the latency issues that have been associated with TCP [58].

QUIC ensures that, despite using UDP, each packet of the handshake is reliably delivered and in order. Since the QUIC encrypts and authenticates its packets, the handshake protocol is also encrypted and authenticated where possible. Furthermore, just like TLS, packet encryption and authentication is achieved via AEAD. QUIC also integrates TLS for the detection of lost packets.

5.3 Secure Shell Protocol (SSH)

The Secure Shell Protocol (SSH) is an application protocol that allows for parties to perform secure remote logins and multiple secure network services, such as remote execution of commands. It is maintained by the Internet Engineering Task Force (IETF) and the most recent version is SSH-2, which is standardized in RFC 4251 [36]. For further technical details of this protocol, please refer to the RFC.

SSH can be split into three sub-protocols, namely the *Transport Layer Protocol*, the *User Authentication Protocol* and *The Connection Protocol*. The Transport Layer protocol provides server authentication, confidentiality and integrity, the User Authentication Protocol authenticates the user and the connection protocol essentially allows for interactive login sessions, remote execution of commands through an encrypted channel [35].

For server authentication, the server contains a public host key, which allows for the user to verify that they are talking to the correct server. Clearly, the user must know the server's public-key beforehand, or have a way to verify the key. Verifying the key is typically done by either having the server and its corresponding key stored in a local database that belongs to the user or by validating the key with a Certification Authority. Furthermore, session keys are established during the Transport Layer Protocol.

Once the server is authenticated, the user needs to be authenticated. This can be achieved via a combination of public-key authentication, password-authentication and host-based authentication [34]. Public-key authentication is done by the server storing a user's public-key and having the user proving that they have knowledge of the private-key. Password-based authentication is done by the user inputting a password to authenticate themselves to the server. Host-based authentication means authenticating a user based on where the connection of the host is coming from and the username.

5.4 Secure/Multipurpose Internet Mail Extensions (S/MIME)

The Secure/Multipurpose Internet Mail Extensions (S/MIME) is a security standard that allows for MIME data to be signed or encrypted with public-key cryptography [59]. MIME data includes formats such as audio, video, non-ASCII characters. It is maintained by the Internet Engineering Task Force (IETF) and, at the time of writing, it is defined in RFC 8551 [59] and RFC 3369 [24] and its latest version is S/MIME Version 4.0. For further technical details of this protocol, please refer to the RFCs. RFC 8551 defines the Cryptographic Message Syntax (CMS), which is the syntax to encrypt, sign and authenticate arbitrary message content. S/MIME v4 essentially is the standard of signing and/or encrypting MIME data according to the CMS.

There exist three classes of S/MIME certificates, each of which provide a different level of assurance of the certificate owner's identity. It is important to note that whilst S/MIME is commonly used in email communication, it is present in many other settings. For example, according to [24], S/MIME can be used in any setting that involves MIME data, such as HTTP.

5.5 Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP) is a security protocol that provides confidentiality and data integrity via both symmetric and asymmetric encryption and digital signatures [18]. For further technical details of this protocol, please refer to the RFC. It also provides key management and certificate services. It is mainly used to secure MIME objects and email communication.

PGP is the generic name of the protocol, but there also exists proprietary commercial software called PGP that implements the PGP protocol and is owned by CA Technologies [46]. Hence, this manuscript will refer to OpenPGP when mentioning PGP, unless stated otherwise. OpenPGP is a security standard maintained by the Internet Engineering Task Force (IETF) and is defined in RFC 4880 [18] that specifies ow to implement the PGP protocol. A popular open source implementation of the PGP algorithm is GnuPG, also known as GPG [37]. It is developed by the GNU Project and is compliant with [18].

Confidentiality is done via symmetric and asymmetric encryption. Firstly, PGP will randomly generate a nonce that is used as a session key, which is then encrypted with both the sender's and recipient's public key. A session key can also be derived from a passphrase or any other shared secret. This allows for both parties to obtain the session key. The message is then encrypted with the session key and sent to the recipient, who is then able to decrypt it via the session key.

For authentication and data integrity, the sender will initially hash the message. A signature is then generated from the hash and the sender's private key, which is sent to the recipient alongside the original message. The recipient generates a hash of the received message and verifies it using the message's signature using the sender's public-key. Note that confidentiality and authentication can be performed on the same message at the same time. The signature is first generated and the signature and original message are encrypted as discussed above.

RFC 4880 introduces the concept of trust signatures. These have different levels of trust and are intended that the public-key does in fact belong to the sender. At Level 0, the trust signature provides the same level of trust as a regular signature. At Level 1, the key is asserted to belong to the sender and at Level 2, the key is asserted to belong to the sender by a Certificate Authority. Furthermore, PGP allows for the revocation of certificates.

An important concept that was introduced by PGP is the Web of Trust. As PGP communications occur, users will begin to accumulate many self-signed certificates from other users and may trust some of these users as *introducers*. These introducers distribute certificates that they trust to other users. As a consequence, the original author of PGP, Philip Zimmermann, stated that "this will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys" [74]. Note that this is a decentralized alternative to modern Public Key Infrastructures.

5.6 Internet Protocol Security (IPsec)

Internet Protocol Security (IPsec) is a security protocol that encrypts and authenticates IP packets between communicating parties [20]. It supports both IPv4 and IPv6 packets. Its main use is in VPNs and has the ability to provide either end-to-end or host-to-host security. A common alternative to IPSec for the usage of VPNs is OpenVPN [44]. As the name suggests, it operates on the Internet Layer (Layer 3) of the OSI model. It is maintained by the Internet Engineering Task Force (IETF) and, at the time of writing, it is defined in many RFCs, with RFC 6071 [20] providing an overview of all IPSec related RFCs. For further technical details of this protocol, please refer to the RFCs. The most recent version of the protocol is IPSec-v3. It is important to note that IPSec does not deal with the distribution of keys. This is usually done by the Internet Key Exchange protocol.

IPSec consists of two main protocols that provide traffic security: the Authentication Header (AH) and the Encapsulating Security Payload (ESP). The AH provides data integrity, data-origin authentication and relay-protection. Note how the AH does not provide any form of confidentiality. It can operate in either a transport-mode or a tunnel-mode.

The ESP provides confidentiality, data integrity, data-origin authentication and relay-protection. Just like AH, ESP can operate in either a transport or tunnel-mode. When used in transport-mode, it protects the upper-layer data but does not provide any form of protection to the IP header. When in tunnel-mode, it completely protects the inner-encapsulated IP packet, but not the outer IP header.

An important IPSec concept are Security Associations (SA). They are agreements between communicating parties on the security specifications of IPSec. To identify these agreements, each IPSec packet includes an identifier that maps to a unique SA. This identifier is called a Security Parameter Index (SPI). These are normally stored in each parties' Security Association Database (SAD). There is also a Peer Authorization Database (PAD), which contains data necessary to perform peer authentication.

5.7 Internet Key Exchange (IKE)

Internet Key Exchange (IKE) is a security protocol that, as the name suggests, is responsible for the exchanging of keys in the IPSec protocol [20]. It is maintained by the Internet Engineering Task Force (IETF) and it is defined by RFC 6071 [20]. For further technical details of this protocol, please refer to the RFC. The most recent version of IKE is IKEv2.

IKE performs mutual authentication between communicating parties and establishes an *IKE Security Association (SA)*. This SA includes the exchanged key and a set of cryptographic algorithms to be used in other steps in the IPSec.

During the IKE protocol, the flow of messages always have the structure of a request and then a response. The first set of messages called IKE_SA_INIT negotiates security parameters, sends nonces and any other cryptographic values. The second set of messages called IKE_AUTH verifies the identities of the communicating parties (normally via X.509 certificates), and sets up a further SA for either AH or ESP.

6 Use-Cases

The protocols presented in the previous section can be used in several application scenarios, which we call *use-cases* here. These use-cases are briefly discussed in this section; notice that a full and detailed description would be beyond the scope of this document, and we therefore only give a short and high-level overview.

6.1 HTTPS

One of the main uses of TLS is securing the Hypertext Transfer Protocol (HTTP). By default, HTTP is communicated in an unencrypted and unauthenticated manner through the Internet, which is not desirable when sending and receiving potentially sensitive information. Since TLS is application-independent, HTTP can be used over TLS and the process of doing such is detailed in RFC 2818 [55]. Henceforth, HTTPS provides confidentiality and integrity to the data being transmitted over the Internet.

Another important use case of HTTPS is for website authentication. This is the process of asserting the identity of the website and is done via X.509 certificates. As previously discussed, this certificate proves ownership of the public-key and hence, the website. This is done during the TLS handshake protocol [57]. A website may also request the user's certificate for mutual authentication.

6.2 Secure Email

There exist many techniques and technologies for securing email communications, such as DMARC [33] and SPF Records [32]. However, the most relevant technique relating to public-key infrastructure is via PGP.

As discussed in Section 5.5, PGP encrypts and signs (email) messages and authenticate them via PGP or X.509 certificates. The main difference is that at the end of an X.509 certificates chain, a Certificate Authority must be present whilst PGP certificates can be signed by anyone and not a Certificate Authority. However, as mentioned in Section 5.5, PGP certificates can also be purchased, which effectively means that the certificate is signed by a Certificate Authority. Hence, Certificate Authorities can (but do not have to) sign PGP certificates.

S/MIME can also be used to encrypt and sign MIME content in email messages with the aforementioned S/MIME certificates.

6.3 Secure Documents

Content that is accessed via HTTP can be encrypted via XML-Enc, which is specified by W3C [12] and/or signed via XML-DSig, which is specified in [42]. These can include X.509 certificates, which hence requires the usage of a PKI. Furthermore, ETSI defined XAdES, which extends XML-DSig to comply with eIDAS [15]. eIDAS is a European Union regulation concerning electronic identification and other trust services for electronic transactions [69]. Another document format that can make use of PKIs are PDFs. PDFs can be signed via the eIDAS compliant PAdES [14] to be legally recognized in the European Union.

6.3.1 Wireless Networks

One of the primary uses of PKIs for wireless networks is via the IEEE 802.1X protocol [26], which provides access control for connecting to local and metropolitan networks. For example, the Wi-Fi services, eduroam, which provides Internet access for users in higher education [13], utilizes 802.1X.

As a high-level overview, 802.1X involves three parties, a client (also known as a supplicant), the network device that authenticates the client and an authentication server. For the client to authenticate themselves, they must either provide a valid username/password combination or a valid certificate. In turn, the user may also provide a certificate to the user for mutual-authentication. These certificates are hence managed by a PKI.

6.3.2 Virtual Private Networks

With the advent of working from home in recent years, the topic of Virtual Private Networks has found its way into the common jargon. As mentioned in Section 5.6, VPNs often use IPSec to provide a secure virtual tunnel between endpoints. Other technologies include OpenVPN [44] or WireGuard [71]. Irrespectively of the technology employed by the VPN, either a digital X.509 certificate or a valid username/password combination are used for authenticating users. Furthermore, mutual authentication can be achieved by the server also providing a valid certificate. Once again, a PKI must be in place to manage these certificates.

6.3.3 Internet of Things

The main difference between PKIs for Internet of Things (IoT) and more "traditional" uses of PKIs such as VPNs is the amount of devices that require authentication via certificates. Each device connected to the Internet requires a certificate and given the number of IoT devices that can exist within a single household, this could put stress on the CA servers. Several CA now offer certificates and enrolment services specifically for IoT devices, such as GlobalSign [23]. These generally involve the ability of customizing fields in the X.509 to suit the IoT devices that are being certified.

7 State-of-the-Art Work on Quantum-Safe PKIs

As mentioned in the introduction, all currently-used public-key cryptographic schemes can be broken by a large-scale quantum computer, which is expected to be developed within the coming decades. Many components of PKIs, as well as many cryptographic protocols that use PKIs to manager their keys, therefore need to be patched with quantum-safe building blocks.

This section elaborates on the current state of research on quantum-safe alternatives for PKIs. We first discuss cryptographic building blocks, and then focus on PKIs and cryptographic protocols.

7.1 Cryptographic Building Blocks

As discussed in Section 3, large-scale quantum computers will render currently-used asymmetric building blocks insecure; these building blocks, namely asymmetric encryption schemes, key-encapsulation methods, and digital-signatures schemes, are crucial ingredients countless cryptographic protocols that are used in modern electronic systems, and a lot of effort has therefore been devoted into designing and analysing quantum-safe alternatives for these building blocks. To move to a hybrid infrastructure it is essential to combine pre-quantum and post-quantum key encapsulation and digital signatures algorithms.

The drawback of quantum-safe schemes is the size of the cryptographic material (keys, signatures, and ciphertexts): long public keys and slower algorithm execution may result in slower protocols, affecting the usability of the schemes in the current infrastructure.

7.1.1 Post-quantum NIST candidates

In 2016, the National Institute of Standards and Technology (NIST) initiated a process to determine which quantum-safe (or post-quantum) cryptographic building blocks are going to be used to replace pre-quantum ones. Several key encapsulation mechanisms (KEM), public-key encryption schemes (PKE) and digital signature schemes (DSS) have been submitted. These cryptographic schemes base their security on computational problems that are supposed to be computationally unfeasible to solve even for quantum computers. The majority of these schemes are Hash-based, Lattice-based, Code-based, Isogeny-based or Multivariate-based schemes:

- Hash-based schemes base their security on the security properties of functions;
- Lattice-based schemes base their security on a class of hard problems like LWE (learning with errors), SVP (shortest vector problem), CVP (closest vector problem) and more;
- Code-based schemes base their security on the hardness of decoding in a linear error correcting code;
- Isogeny-based schemes base their security on the hardness of computing a certain isogeny between elliptic curves;
- Multivariate-based schemes base their security on the hardness of solving a system of multivariate polynomial equations.

According to NIST guidelines, the evaluation of all the schemes is performed on 5 levels of security described in [43]

At the time of writing, the finalists of NIST contest have been published[2]. The final choice of NIST for the candidate schemes will be made official in April 2022. The finalist schemes are reported in Table 1.

Name	Scheme	Type
Classic McEliece	PKE/KEM	Code-based
Crystals-Kyber	PKE/KEM	Lattice-based
NTRU	PKE/KEM	Lattice-based
SABER	PKE/KEM	Lattice-based
Crystals-Dilithium	DSS	Lattice-based
Falcon	DSS	Lattice-based
Rainbow	DSS	Multivariate-based

Table 1: NIST Finalists

However, a new vulnerability has been recently found on Rainbow [4]. This fact stresses even more the necessity to move to a hybrid PKI until quantum-safe algorithms are mature enough to be deployed independently.

Along with the finalists, NIST has published a list of alternative candidates. The alternative candidates are reported in Table 2.

To ensure the maximum security, it is recommended to use strong cryptographic techniques to combine pre-quantum and post-quantum schemes: the hybrid schemes will remain pre-quantum safe until quantum computers are deployed while the security of post-quantum schemes reaches its maturity and once post-quantum schemes will be employed, the hybrid scheme will remain secure.

However, even with secure pre-quantum and post-quantum schemes, it is not straightforward to securely combine them together. To this end, scientific research is inspecting which cryptographic combining mechanisms ensure the best security.

[2]More information can be found at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`

Name	Scheme	Type
FrodoKEM	PKE/KEM	Lattice-based
NTRU Prime	PKE/KEM	Lattice-based
BIKE	PKE/KEM	Code-based
HQC	PKE/KEM	Code-based
SIKE	PKE/KEM	Isogeny-based
SPHINCS	DSS	Hash-based
GeMSS	DSS	Multivariate-based
Picnic	DSS	Hash-based

Table 2: NIST Alternative Candidates

7.1.2 Quantum-safe Cryptographic Combiners

Rigorously, the purpose of a cryptographic combiner is to turn two (or more) cryptographic schemes, e.g., key-encapsulation schemes, into a single scheme that offers the same (or a similar) functionality, i.e., is again a key-encapsulation scheme, and is secure as long as *at least one* of the two (or more) original schemes is secure. Thus, applying such a combiner to a pre-quantum, say factoring-based, and a post-quantum, say lattice-based, scheme, results in a new scheme that remains secure unless *both* underlying schemes can be broken. Thus, even if there turns out to be an unexpected weakness in the lattice-based scheme, as long as there is no sufficiently large and reliable quantum computer (and no other progress on the factoring problem), the combined scheme remains secure. And vice-versa, the combined scheme will withstand quantum attacks if the lattice-based scheme is as secure as expected.

There are different approaches and techniques towards constructing cryptographic combiners. For instance, one can aim for combiners that turn strongly secure (like CCA-secure) schemes into a strongly secure scheme again, as in [22], or one can aim for combiners that turn weakly secure schemes into a weakly secure scheme again and then apply a generic transformation to eventually obtain a strongly secure scheme, as in [25]. The latter approach appears favourable at first glance, since it puts weaker requirements on the underlying schemes; on the other hand, standardized scheme are most likely designed to offer strong secure anyway, and so the latter may introduce an unnecessary overhead by not exploiting the already-exist strong security. In any case, when aiming for such a hybrid approach, one needs to deal with the fact that combining schemes introduces a significant overhead in key- and ciphertext-size, and in computation time.

7.2 Quantum-Safe PKIs

Having given an overview of quantum-safe cryptographic primitives, we now focus on PKIs. In principle, simply replacing the quantum-vulnerable components of PKIs with quantum-safe ones would make PKI systems quantum-safe. However, three factors complicate an actual implementation of this approach:

- Quantum-safe building blocks are typically not as efficient (in terms of running time of the algorithms and of size of relevant elements) as pre-quantum ones.
- Designing *hybrid* quantum-safe PKIs (cf. Introduction) is not trivial and requires an analysis that depends on each component of a given PKI.
- The cryptographic protocols should be adjusted so that they can support the new functionality and still be compatible with the existing infrastructure.

This section discusses the scientific literature focused on adding quantum-security to X.509 certificates and on investigating their impact in terms of certificate size and efficiency loss.

7.2.1 Extended X.509 Certificates

A first attempt to extend X.509v3 certificates to support quantum-safe cryptography was proposed by Bindel et al. [6]. The idea is to use the extension field in the certificates and insert the quantum-safe signature and/or quantum-safe public-key. The extension field is considered not-critical and can therefore be handled by users that have not yet migrated to quantum-safe cryptography. In [6], a study is conducted whether the sizes of the quantum-safe algorithm would exceed the maximum certificate sizes supported by the GnuTLS, Java SE, mbedTLS, NSS and OpenSSL libraries. The results show that all libraries were able to correctly parse and verify the certificates with the long signatures in the extension field.

A similar approach was independently presented to IETF in an Internet Draft [68]. This document presents a method to extend the existing X.509v3 Certificates, Certificate Signing Requests (CSR) and X.509v2 Certificate Revocation Lists (CRL) in such a way they can handle multiple public-key algorithms. The document introduces non-critical extension fields that contain info of alternative public-key algorithms and provide instructions on how these should be handled.

Ultimately, the approach with which the two types of certificates are implemented is similar in that all extensions dictate the use of two extra non-critical fields: one for the public key (and its attributes) and one for the digital signature (and its attributes).

In the eventual scenario in which only a post-quantum scheme is used, due to the parameters of the post-quantum algorithms, the size of certificates will be around 4.3 to 54 kilobytes as opposed to pre-quantum certificates of 1 to 1.5 Kilobytes.

Scientific research was conducted separately on these types of certificates applied to mainly TLS but also SSH, IKEv2 and others. It is reasonable to assume that the results on one type of certificate also apply for the other ones, due to the similarity of the implementation of the certificates and the validation of the results provided by their analysis.

For every protocol within the X.509 PKI, we report the result concerning: compatibility, impact and integration. Concerning compatibility, we investigate if the current specification of the protocol allows for a smooth transition and full support of quantum-safe algorithms. Concerning impact, we investigate how the performance of the protocol changes due to their large key sizes. Concerning integration, we inspect if there exists a library that implements the protocol with quantum-safe algorithms.

7.2.2 CSR

The Certificate Signing Request (CSR) is an X.509 message sent from an applicant to a RA in order to apply for a digital identity certificate. Raavi et al. [54] investigates the effect of adding quantum security to X.509 certificates. Their primary focus is on the overhead in size and loss in efficiently for the different security levels.

Compatibility and Impact The smallest post-quantum and hybrid certificates sizes are provided by Falcon at level 1 with about 3000 bytes and level 5, with approximatively 4800 bytes certificate size, Dilithium is the best alternative at level 3 with circa 7500 byes certificate size (Falcon does not have a level-3 security variant).

Concerning key-generation, Dilithium is the most efficient with respect to the time needed to generate a valid key-pair. At security level 1 Dilithium is faster than RSA, and at level 3 and 5 it is faster than ECC and in particular, the key-generation time in Dilithium-II and Falcon-512 is faster than RSA by 7.08 - 28.11 times. Regarding Certificate Requests, Dilithium outperforms all post-quantum counterparts at all levels of security. On average, Dilithium, and Falcon reduce the Certificate Request Generation time by about 2 times. For Certificate Generation, Dilithium, and Falcon outperform all post-quantum counterparts at all levels of security. At security level 5 Dilithium is 10% - 15% faster than Falcon. For Certificate Verification, RSA is still the most efficient at the lowest level of security. Among post-quantum schemes, Dilithium, and Falcon have 7.61% and 4.76% more time overhead for verifications at level 1. Dilithium and Falcon are more efficient than ECDSA on all security levels. Falcon is the most efficient among the post-quantum algorithms for level 1 and 5.

Due to the difference in performances, Dilithium is the recommended for time-sensitive applications (any web server-client communication) but Falcon is a good candidate for Blockchain certificate verification since blockchain protocols require more verification than signatures.

7.2.3 OCSP

The Online Certificate Status Protocol (OCSP) is a protocol used for obtaining the revocation status of an X.509 certificate. Fan et al. [17] also investigates the effect of adding quantum security to X.509 certificates. The focus of their work is on the overhead in size and loss in efficiently for command line and browser applications of X.509 certificates.

The analysis was performed on hybrid certificates modified according to the Internet Draft [68] specifications based on the OpenSSL 1.1.1b libraries.

Compatibility and Impact For command-line applications, it is possible to set the maximum size of the response via the API function `OCSP_set_max_response_length` to allow OCSP to process large certificate or a chain of certificates: setting a large response size length performed successfully. For browser applications, tests were conducted on Firefox 66.0.3 and Internet Explorer 11.0.9600.19326. Internet Explorer works correctly for the tested sizes, but Firefox has a size limit for certificate sizes of 65.535 kilobytes.

7.2.4 CMP and EST

The Certificate Management Protocol (CMP) is a protocol used for obtaining X.509 digital certificates. Enrolment over Secure Transport (EST) is a protocol that describes an X.509 certificate management, targeting clients that need to acquire client certificates and associated certificate authority (CA) certificates. The analysis of post-quantum CMP and EST protocols is performed by [17]. No issues were found for CMP and EST as they support all sizes of the quantum-safe certificate with minimal execution time impact.

7.3 Cryptographic Protocols Using PKIs

For cryptographic protocols that use PKIs to manage the associated keys, a similar observation holds as for PKIs themselves (cf. beginning of Section 7.2): transitioning to a quantum-safe state is not trivial, even with quantum-safe building blocks available, and especially when requiring the new protocol to offer hybrid security and interoperability. As for Section 7.2, for every protocol, we report the result concerning compatibility, impact, and integration.

A large body of research has been conducted on the impact of post-quantum algorithms on the most commonly used protocols on the Internet. The core of the research was focused on client/server communication (and implicitly with HTTPS application) but some research was also conducted on embedded systems. Therefore, when we describe the scientific literature, we implicitly refer to client/server communication, and we will be explicit when talking about embedded systems. Furthermore, many initiatives have emerged with the goal to facilitate the migration to quantum-safe internet: several internet drafts have been submitted to IETF and the Open Quantum Safe (OQS) project has developed the library *liboqs* with quantum-safe cryptosystems to be included in the libraries that implement quantum-safe protocols.

7.3.1 TLS 1.2

As one of the most widely used protocol on the internet, TLS 1.2 has received the attention of [6, 29, 10, 17] for Internet applications and client-server communication and [8] for embedded systems.

The Internet drafts [61, 9] have been submitted to IETF.

Compatibility and Impact TLS supports data fragmentation for certificates up to 16.777 megabytes, which is more than sufficient for quantum-safe signatures, but TCP fragmentation is enforced and fragments are at most 16 kilobytes large. This can create high loss rates especially with UDP [6, 29].

Long certificates would not highly benefit from techniques like compression or caching. Caching has the potential to speed up connection, but it could also potentially introduce vulnerabilities. On the other hand, compression would not significantly reduce the sizes of the certificates because the digital signatures are very large and are difficult to compress due to their large entropy.

The most common libraries and browsers were tested by [6] to determine if they support long certificate sizes. Among all tested libraries, the library that could support the largest certificates was Java SE with 1333.0 kilobytes. Almost all other libraries could process up to 43.0 Kilobytes except for mbedTLS, which has a limit of 9.0 kilobytes. Among the web browsers, Safari could support the largest certificates with 1333.0 kilobytes. Almost all other browsers, like Chrome, Opera and Firefox, could support up to 43.0 kilobytes, with the exceptions of Microsoft Edge and Internet Explorer, which could support up to 9.0 kilobytes.

An extension of this analysis was performed by [17] where the analysis was conducted using signature schemes with very large parameters like SPHINCS, Picnicl5rl and GeMSS, validating the results of [6]. The results show that the size limit of OpenSSL 1.0.2 is set to 102.4 Kilobytes, but it can be increased by calling the appropriate API.

The work in [10] tested the compatibility of NIST candidates in OpenSSL 1.0.2. Most of the schemes were handled without issues; NTS-KEM was the only one that failed. FrodoKEM-1344 failed as well to be handled by OpenSSL 1.0.2, but as shown by [17], a change in the parameters of the implementation would allow FrodoKEM-1344 to be used correctly.

The work in [29] takes a similar approach, and they test HSS in hybrid certificates. For different parameters of a hash-based signature scheme, the average handshake took about 4 times longer than pre-quantum certificates with RSA signatures and the average of the number of exchanged packets is slightly increased. When these certificates are handled by browsers like Chrome and Firefox, in alignment with the results of [6], fragmentation and segmentation were working properly for a certificate chain of total size 19 kilobytes. For certificate chains of 135 kilobytes, the browsers were failing to parse the certificates. Other experiments show that the failure in the parsing was due to the certificate chain length and not the chain size.

In the embedded system setting, Kyber for key-exchange and SPHINCS$^+$ for digital signature were tested on a Raspberry Pi, an ESP32, a fieldbus option card and an LPC. Kyber was recorded to be more performing than ECDHE on all platforms in terms of performance and resource management. SPHINCS$^+$ was noticeably less efficient than ECDSA.

Concerning the possibility to use cryptographic combiners, TLS 1.2 does not seem suitable to support them except the concatenation combiner as shown by [6, 10].

Integration An extension of TLS 1.2 on OQS-OpenSSL 1.0.2 based on the implementation by Bindel et al. [5] exists. This implementation supports hybrid and quantum-safe only schemes. In the hybrid certificates, public keys are concatenated and parsed when received. However, this version of OpenSSL is deprecated and not maintained[3]. Another implementation of TLS 1.2 supporting post-quantum algorithms is now available in s2n [1, 2] based on the Internet Draft [9].

DTLS 1.2 is integrated in WolfSSL [73].

The Internet drafts [61, 9] have been submitted to IETF to help the integration of quantum-safe schemes into TLS 1.2.

7.3.2 TLS 1.3

As TLS 1.3 is soon to be widely-used as a successor of TLS 1.2, it has been subject to analyses of various levels. A large body of scientific research has been performed by [10, 47, 64, 63, 45] on Internet application and client-server communication and [21] for embedded systems.

Compatibility and Impact The design of TLS 1.3 allows for a more flexible negotiation of cryptographic schemes, as every algorithm is negotiated separately. Therefore, post-quantum algorithms can be negotiated directly with a new identifier. As opposed to TLS 1.2 using different methods for combining the digital signatures can be supported by TLS 1.3. The key schedule of TLS 1.3 is more complex than the one from TLS 1.2, so techniques inspired from [5] can be easily applied, even though the proposed approach concatenates the cryptographic material by default. As for the key exchange, TLS 1.3 supports multiple algorithms and can negotiate them separately. TLS 1.3 usually conveys digital signatures via X.509 certificates, and it allows the sending of multiple certificates. Therefore, in principle, TLS 1.3 would support the deployment of two separate certificates (one for pre-quantum schemes and the other for post-quantum ones) instead of one with multiple signatures would be deployed. The `CertificateVerify` message used to convey a signature cannot be extended via built-in mechanisms, so it could only be extended or duplicated with a change in the logic of the protocol.

The authors in [10] deeply analyse the technical side of integrating quantum-safe algorithms into TLS 1.3. During the Key exchange protocol, the maximum size of the key value in the `key_share` extension of `Client_Hello` is 65.535 kilobytes during the key exchange process. This limitation can create issues with schemes like FrodoKEM-1344. In fact, even if 65.535 kilobytes is large enough for FrodoKEM parameters, the limited size of Server Hello (20 kilobytes) is not enough. If this limit were to be enlarged, FrodoKEM would function correctly. On the other hand, schemes like NTS-KEM and Classic McEliece are too large to be handled correctly. Concerning the digital signatures, the maximum X.509 certificate size in TLS 1.3 is 16.777 megabytes which accommodates the size of all NIST finalists. However, as for TLS 1.2, the maximum size of the supported signature is 65.535 kilobytes which is too small for schemes like Picnic or Rainbow. As for TLS 1.2, record fragmentation is applied for certificates of size larger than 16 kilobytes.

When deployed in TLS 1.3, Falcon and Dilithium provide the best performance: Falcon has fast verification time, but has slow signing performance, Dilithium, on the other hand, has very fast signing time and slower verification time. The cryptographic scheme with a bigger impact

[3]`https://openquantumsafe.org/applications/tls.html#oqs-openssl-provider`

on performance was SPHINCS that cause up to 190% latency, because the large sizes lead to multiple round trips. When benchmarked in realistic Internet conditions, the packet loss rate is around 3-5%.

Hardware optimization (like AVX2) can be extremely beneficial for the performance of cryptographic operations: Falcon performs 20 times faster and Dilithium-IV with hardware optimization performs faster than Dilithium-II.

When employed in a TLS handshake, the time needed to convey Dilithium-II or Falcon-512 public keys is competitive with RSA and ECDSA (with a maximum recorded delay of 55ms). For higher levels of security, the impact of Dilithium-IV and Falcon-1024 is more sensible.

To reduce time overhead even more, it is recommended to use different cryptographic algorithms across certificate chains. This technique reduces the overall handshake time of about 25% compared to single Dilithium-IV and 33% compared to single Falcon-1024. Further research proposes to mix Dilithium and Falcon and ECC at a low level of hierarchy (intermediate CA and endpoints) and SPHINCS and XMSS at root level.

When TLS 1.2 is employed in embedded systems, the authors of [21] perform an analogous analysis and similar results are recorded. Experiments were conducted on two microcontrollers with limited memory and computing power. For key-exchange schemes, Kyber and Saber are the most performing schemes as their performance is comparable to ECDHE. Concerning signature schemes, Dilithium, and Falcon confirm their performances as for Internet client/server communication as they are both well performing. Dilithium is very efficient for key-generation and signature operations: Dilithium-II is circa 2.8 times slower than ECDSA, and is approximately as efficient as RSA. For verification operations, Dilithium is on average 1.89 times slower than RSA.

Falcon is well performing for verification operations: Falcon-512's verify operation time is about 3.6 times the corresponding RSA verify operation and Falcon-1024 verify is about 1.7 times faster than RSA. Falcon is on average 17.90% slower than RSA for signature operations.

When Dilithium and Falcon are deployed for handshake protocols, the post-quantum schemes provide similar performances to their pre-quantum counterparts. Experiments show that in embedded system the size overhead increases by a few order of magnitude due to the large sizes of the post-quantum cryptographic material, but the power consumption is dominated by the communication transmission cost.

Integration An implementation of a hybrid version of TLS 1.3 is provided in an OQS fork of OpenSSL 1.1.1 [52], in an OQS fork of BoringSSL [51] and in an OQS fork of WolfSSL [72].

The Internet Drafts like [31, 60, 65, 70] have been submitted to IETF to help the integration of quantum-safe schemes in TLS 1.3.

7.3.3 SSH

The works by Crockett, Paquin and Stebila [10] and the short paper by Sikeridis et al. [63] are the only scientific works that deal with the integration of post-quantum algorithms in SSH-2. Crockett et al. focuses on the overall compliance of the current state of the art of SSH with respect to key exchange and authentication. Sikeridis et al. focuses on inspecting the latency of the handshake and study the possible overhead when SSH packets are being transported via TCP.

Compatibility and Impact Extending the SSH cipher-suite with the new post-quantum algorithms and combinations is supported by SSH and requires no adjustment.

Messages in SSH are 32-bits long and can accommodate any post-quantum algorithm of Round 3 of the NIST competition.

Concerning key-exchange, the maximum packet size supported by OpenSSH is 262.144 kilobytes bytes which is large enough to support all Round 2 candidates (and by extension Round 3) except for NTS-KEM and Classic McEliece. In SSH it is possible to exchange classic and quantum-safe keys without inconveniences [10]: "In SSH-2, each *key exchange* method gets to define its own message format for its messages, so it is possible for hybrid exchange methods to provide distinct fields for each component value". For authentication (i.e., digital signatures), the maximum payload for packets is 32.768 bytes and is able to support the sizes of public keys of the finalists.

Regarding the possibility of integrating combiners, OQS's fork to SSH by the default concatenates public keys (classic keys first and quantum-safe keys second).

Results reported in [10] show that OpenSSH could successfully perform key-exchange for every post-quantum encapsulation scheme except for NTS-KEM. It also could successfully perform authentication for all digital signature schemes except for Rainbow with large parameters (Rainbow-III and Rainbow-V).

When post-quantum algorithms are integrated in SSH, [63] report that the handshake can have a latency from a minimum of 0.5% till a maximum of 50% (Dilithium is the faster while SPHINCS is the algorithm that causes the most latency). To circumvent this issue it is possible to increase the TCP initial congestion window, at the cost of increased packet loss rate.

Integration An early Internet draft [66] outlines how to implement a hybrid key-exchange in SSH. An implementation of hybrid SSH is provided in a OQS fork of OpenSSH 7.9 [53] and of libssh [50].

7.3.4 IKEv2

The question of integrating post-quantum X.509 certificates in IKEv2 is tackled by [29]. In their paper, the authors inspect whether the hybrid certificates can be supported by the IKE protocol and run some analysis using a stateless Hash Signature Scheme (HSS) as the post-quantum algorithm.

Compatibility and Impact IKEv2 uses fragmentation for large packets that exceed the size of the MTU. Large certificates were observed to perform "without special issues".

However, fragmentation is applied only to IKE_AUTH messages that are in charge of authentication, but fragmentation will not apply to IKE_SA_INIT messages that transport the public keys.

The negotiation for the session key with a hash-based signature scheme instead of RSA took on average 200-400 ms more and the number of packets increased, but the whole sessions lasted less than half a second. This shows that extending IKE with quantum-safe schemes will not have a strong impact on performance.

They tested the handling of the Truskowsky [68] type of certificate implemented in StrongSwan 5.5.0 with RSA and HSS. The results show that IKE_AUTH fragmentation works correctly even with long certificate chains with post-quantum algorithms.

Integration An implementation of quantum-safe OQS fork of StrongSwan can be found at [49]. Related work on the integration of quantum-safe algorithm in IKE can be found in the Internet-Drafts [67, 19], but they solely concern key-exchange.

7.3.5 QUIC

The question of integrating post-quantum X.509 certificates in QUIC is tackled by [29]. As for IKE (Sec. 7.3.4), the authors inspect whether the hybrid certificates can be supported by the QUIC protocol and run some analysis using a stateless hash-based signature scheme as post-quantum algorithm.

Compatibility and Impact QUIC supports the use of fragmentation, compression, and caching for the transport of large packets. These mechanisms allow large X.509 certificates to be handled without errors.

The tests performed in [29] show that all certification chains were transferred correctly. Timeout errors were registered at the server side due to errors in the implementation and not on the design of the protocol. According to the authors QUIC will have minimal impact as: “It is evident that QUIC will operate with lengthy certificates with no issue” [29].

Integration The analysis were performed on Google’s proto-quic implementation which is deprecated at the time of writing. More details about integration of QUIC with quantum-safe schemes can be found at [3].

7.3.6 S/MIME and CMS

The hybrid approach to S/MIME has only been investigated by [6]. This work analyses how the hybrid X.509 certificates are handled by S/MIME and CMS and if they can support cryptographic combiners.

Compatibility and Impact In S/MIME the employed algorithms are specified in a header and more precisely in a CMS `SignedData` object that contains several fields. Among these the object `SignerInfo` can occur several times. This object contains the attributes about the signer, the algorithm, and the signature and other fields and can be used to deliver combined secrets.

Analysis was conducted on some libraries that implement S/MIME with different configurations. Results show that backwards compatibility is reached when quantum-safe signatures and key are stored on non-critical fields. Among the libraries tested, only Mozilla Thunderbird failed in parsing large attributes.

Integration An integration of CMS and S/MIME can be found in the OQS fork of OpenSSL 1.1.1 [11].

7.3.7 OpenVPN

A post-quantum implementation of OpenVPN has been developed by Microsoft Research [39], but no scientific literature has been published. It supports FrodoKEM, SIDH, Picnic and qTESLA as quantum-safe cryptographic schemes. The experimental implementation of OpenVPN provided by Microsoft is a fork of OQS-OpenSSL [40].

8 Conclusion

Before transitioning to a fully quantum-safe PKI, a hybrid migration is necessary. Adapting a hybrid temporary solution ensures that we can rely on the security of pre-quantum schemes until quantum-safe schemes are fully trusted.

The adoption of quantum-safe schemes, especially in a hybrid setting, comes with a few challenges. Firstly, the large sizes of the ciphertexts and signatures of the new schemes is going to impact the communication as there are bigger data chunks that need to be transferred. Then, moving to a quantum-safe PKI does not only entail replacing existing cryptographic schemes with a quantum-safe variant, but also involves changing the existing infrastructure. The quantum-safe schemes should be compatible with the current PKI and the pre-quantum PKI should also be able to function along with the quantum-safe schemes.

In this document we have discussed the scientific work conducted on the impact of quantum-safe schemes in the most used protocols of the Internet. Table 3 summarizes the discussion in Section 7.2 and Section 7.3.

Protocol	Support in Libraries and Browsers	Communication Overhead	Support for cryptographic combiners
CSR	No adjustment required	Dilithium and Falcon provide the smallest overhead	Not tested
OSCP	No adjustment required	minimal overhead compared to pre-quantum schemes	Not tested
CMP-EST	No adjustment required	Minimal overhead	Not tested
TLS 1.2	The implementation of the libraries requires adjustments on size limits	Sensitively increase with hash-based schemes. Not tested for Dilithium and Falcon	Only concatenation can be supported
TLS 1.3	Size limits not compatible with sizes of some PKE/KEM schemes, changes in the implementation are required	Dilithium and Falcon provide the smallest overhead, SPHINCS provides the biggest impact	Potentially supported
SSH	no adjustment required	Dilithium and Falcon provide the smallest overhead, SPHINCS provides the biggest impact	Current implementation only supports concatenation
IKEv2	No adjustments required	Only hash-based signatures tested and no big impact was recorded	Not investigated
QUIC	No adjustment required	Timeout errors caused library implementation	Not investigated
S/MIME-CMS	Small adjustment required for Mozilla Thunderbird to increase size limit	Not tested	Potentially supported with a small adjustment of the implementation
OpenVPN	Implemented only for some schemes	No literature	No literature

Table 3: Summary of Quantum-safe Protocols

References

[1] AWS. Post-quantum TLS now supported in AWS KMS. `https://aws.amazon.com/blogs/security/post-quantum-tls-now-supported-in-aws-kms/`. 04/11/2019. [Accessed: 18/02/2022].

[2] AWS. Round 2 post-quantum TLS is now supported in AWS KMS. `https://aws.amazon.com/blogs/security/round-2-post-quantum-tls-is-now-supported-in-aws-kms/`. 16/11/2020. [Accessed: 18/02/2022].

[3] Igor B. The QUIC Protocol and Quantum-Safe Cryptography - Presenting a Platform for Much-Needed Future Experiment. `https://www.linkedin.com/pulse/quic-protocol-quantum-safe-cryptography-presenting-future-igor/`, 2022. [Accessed: 10/02/2022].

[4] Ward Beullens. Breaking Rainbow Takes a Weekend on a Laptop. Cryptology ePrint Archive, Report 2022/214, 2022. `https://ia.cr/2022/214`.

[5] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*, volume 11505 of *Lecture Notes in Computer Science*, pages 206–226. Springer, 2019.

[6] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. Transitioning to a Quantum-Resistant Public Key Infrastructure. In *Post-Quantum Cryptography*. Springer International Publishing, 2017.

[7] Johannes Buchmann, Evangelos G. Karatsiolis, and Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. Springer, 2013.

[8] Kevin Bürstinghaus-Steinbach, Christoph Krauß, Ruben Niederhagen, and Michael Schneider. Post-Quantum TLS on Embedded Systems: Integrating and evaluating Kyber and SPHINCS+ with Mbed TLS. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20. Association for Computing Machinery, 2020.

[9] Matt Campagna and Eric Crockett. Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS). Internet-Draft draft-campagna-tls-bike-sike-hybrid-01, IETF Secretariat, May 2019. `https://www.ietf.org/archive/id/draft-campagna-tls-bike-sike-hybrid-01.txt`.

[10] Eric Crockett, Christian Paquin, and Douglas Stebila. Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. *IACR Cryptol. ePrint Arch.*, page 858, 2019.

[11] CROSSINGTUD. GitHub - CROSSINGTUD/openssl-hybrid-certificates: Fork of OpenSSL from liboqs that adds backwards compatible hybrid certificates. — github.com. `https://github.com/CROSSINGTUD/openssl-hybrid-certificates`. [Accessed: 08/02/2022].

[12] Donald Eastlake, Joseph Reagle, Thomas Roessler, and Frederick Hirsch. XML encryption syntax and processing version 1.1. W3C recommendation, W3C, April 2013. https://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/.

[13] eduroam. `https://eduroam.org/`, 2022. [Accessed: 09/03/2022].

[14] ETSI. Electronic Signatures and Infrastructures (ESI); PAdES digital signatures. ETSI EN 319 142-1, April 2016.

[15] ETSI. Electronic Signatures and Infrastructures (ESI); XAdES digital signatures. ETSI EN 319 132, April 2016.

[16] K.R. Fall and W.R. Stevens. *TCP/IP Illustrated*. Number v. 1 in Addison-Wesley professional computing series. Addison-Wesley, 2011.

[17] Jinnan Fan, Fabian Willems, Jafar Zahed, John Gray, Serge Mister, Mike Ounsworth, and Carlisle Adams. Impact of post-quantum hybrid certificates on PKI, common libraries, and protocols. *International Journal of Security and Networks*, 16(3):200–211, 2021.

[18] Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney L. Thayer, and David Shaw. OpenPGP Message Format. RFC 4880, November 2007.

[19] Scott Fluhrer, Panos Kampanakis, David McGrew, and Valery Smyslov. Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security. RFC 8784, June 2020.

[20] Sheila Frankel and Suresh Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, February 2011.

[21] Tasopoulos George, Jinhui Li, Apostolos P. Fournaris, Raymond K. Zhao, Amin Sakzad, and Ron Steinfeld. Performance Evaluation of Post-Quantum TLS 1.3 on Embedded Systems. Cryptology ePrint Archive, Report 2021/1553, 2021. `https://ia.cr/2021/1553`. Under review at AsiaCCS'22.

[22] Federico Giacon, Felix Heuer, and Bertram Poettering. Kem combiners. In *IACR International Workshop on Public Key Cryptography*, pages 190–218. Springer, 2018.

[23] GlobalSign. `https://www.globalsign.com/en/internet-of-things/iot-device-certificates`, 2022. [Accessed: 09/03/2022].

[24] Russ Housley. Cryptographic Message Syntax (CMS). RFC 3369, September 2002.

[25] Loïs Huguenin-Dumittan and Serge Vaudenay. Fo-like combiners and hybrid post-quantum cryptography. In *Cryptology and Network Security*. Springer International Publishing, 2021.

[26] IEEE. IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control, 2020.

[27] ITU-T. Information technology–open systems interconnection–the directory: Public-key and attribute certificate frameworks. *ITU-T Recommendation*, 1999.

[28] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[29] Panos Kampanakis, Peter Panburana, Ellie Daw, and Daniel Van Geest. The Viability of Post-quantum X.509 Certificates. *IACR Cryptol. ePrint Arch.*, page 63, 2018.

[30] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014.

[31] Franziskus Kiefer and Kris Kwiatkowski. Hybrid ECDHE-SIDH Key Exchange for TLS. Internet-Draft draft-kiefer-tls-ecdhe-sidh-00, IETF Secretariat, November 2018. `https://www.ietf.org/archive/id/draft-kiefer-tls-ecdhe-sidh-00.txt`.

[32] Scott Kitterman. Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1. RFC 7208, April 2014.

[33] Murray Kucherawy and Elizabeth Zwicky. Domain-based Message Authentication, Reporting, and Conformance (DMARC). RFC 7489, March 2015.

[34] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Authentication Protocol. RFC 4252, January 2006.

[35] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Connection Protocol. RFC 4254, January 2006.

[36] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006.

[37] Nikos Mavrogiannopoulos and Simon Josefsson, 2021. [Accessed: 22/02/2022].

[38] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[39] Microsoft. Post-quantum Crypto and VPNs. `https://www.microsoft.com/en-us/research/project/post-quantum-crypto-vpn/`. [Accessed: 10/03/2022].

[40] Microsoft. Pscrypto-vpn. `https://github.com/microsoft/PQCrypto-VPN`, 2020. [Accessed: 10/03/2022].

[41] Michele Mosca and Marco Piani. Quantum threat timeline report 2020. *Website: https://globalriskinstitute. org/publications/quantum-threat-timeline-report-2020/*, 2021.

[42] Magnus Nyström, David Solo, Thomas Roessler, Donald Eastlake, Frederick Hirsch, Joseph Reagle, and Kelvin Yiu. XML signature syntax and processing version 1.1. W3C recommendation, W3C, April 2013. https://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/.

[43] National Institute of Standards and Technology. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`, 2016. [Accessed: 10/03/2022].

[44] OpenVPN. GitHub - OpenVPN/openvpn: OpenVPN is an open source VPN daemon. — GitHub.com. `https://github.com/OpenVPN/openvpn`. [Accessed 15-Feb-2022].

[45] Christian Paquin, Douglas Stebila, and Goutam Tamvada. Benchmarking post-quantum cryptography in TLS. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 72–91. Springer, 2020.

[46] United States Patent and Trademark Office. `https://tmsearch.uspto.gov/bin/showfield?f=doc&state=4805:6aezou.2.26`, 2022. [Accessed: 18/02/2022].

[47] Sebastian Paul, Yulia Kuzovkova, Norman Lahr, and Ruben Niederhagen. Mixed Certificate Chains for the Transition to Post-Quantum Authentication in TLS 1.3. *IACR Cryptol. ePrint Arch.*, page 1447, 2021.

[48] RSA PKCS. v1. 0: Personal information exchange syntax standard, june 1999, 12.

[49] pq-strongswan. strongX509/docker. `https://github.com/strongX509/docker/tree/master/pq-strongswan`. [Accessed: 10/02/2022].

[50] Open Quantum Safe Project. open-quantum-safe/libssh: Fork of libssh that includes prototype quantum-resistant algorithms based on liboqs. `https://github.com/open-quantum-safe/libssh`. [Accessed: 10/02/2022].

[51] Open Quantum Safe Project. OQS-boringSSL. `https://github.com/open-quantum-safe/boringssl`.

[52] Open Quantum Safe Project. OQS-openSSL_1_1_1-stable. `https://github.com/open-quantum-safe/openssl`.

[53] Open Quantum Safe Project. OQS-OpenSSH, 2019.

[54] Manohar Raavi, Pranav Chandramouli, Simeon Wuthier, Xiaobo Zhou, and Sang-Yoon Chang. Performance Characterization of Post-Quantum Digital Certificates. In *30th International Conference on Computer Communications and Networks, ICCCN 2021, Athens, Greece, July 19-22, 2021*, pages 1–9. IEEE, 2021.

[55] Eric Rescorla. HTTP Over TLS. RFC 2818, May 2000.

[56] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[57] Eric Rescorla, Hannes Tschofenig, and Nagendra Modadugu. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-dtls13-43, Internet Engineering Task Force, April 2021. Work in Progress.

[58] Jim Roskind. Experimenting with QUIC. `https://blog.chromium.org/2013/06/experimenting-with-quic.html`, Jun 2013. [Accessed: 15-Feb-2022].

[59] Jim Schaad, Blake C. Ramsdell, and Sean Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification. RFC 8551, April 2019.

[60] John M. Schanck and Douglas Stebila. A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret. Internet-Draft draft-schanck-tls-additional-keyshare-00, IETF Secretariat, April 2017. `https://www.ietf.org/archive/id/draft-schanck-tls-additional-keyshare-00.txt`.

[61] John M. Schanck, William Whyte, and Zhenfei Zhang. Quantum-Safe Hybrid (QSH) Ciphersuite for Transport Layer Security (TLS) version 1.2. Internet-Draft draft-whyte-qsh-tls12-02, IETF Secretariat, July 2016. `https://www.ietf.org/archive/id/draft-whyte-qsh-tls12-02.txt`.

[62] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[63] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Assessing the Overhead of Post-Quantum Cryptography in TLS 1.3 and SSH, 2020.

[64] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-Quantum Authentication in TLS 1.3: A Performance Study. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[65] Douglas Stebila, Scott Fluhrer, and Shay Gueron. Hybrid key exchange in TLS 1.3. Internet-Draft draft-ietf-tls-hybrid-design-04, IETF Secretariat, January 2022. `https://www.ietf.org/archive/id/draft-ietf-tls-hybrid-design-04.txt`.

[66] E. Crockett T. Hansen, M. Campagna. Hybrid Key Exchange Integration in the Secure Shell Transport Layer, 2018.

[67] C. Tjhai, M. Tomlinson, grbartle@cisco.com, Scott Fluhrer, Daniel Van Geest, Oscar Garcia-Morchon, and Valery Smyslov. Framework to Integrate Post-quantum Key Exchanges into Internet Key Exchange Protocol Version 2 (IKEv2). Internet-Draft draft-tjhai-ipsecme-hybrid-qske-ikev2-04, IETF Secretariat, July 2019.

[68] Alexander Truskovsky, Philip Lafrance, Daniel Van Geest, Scott Fluhrer, Panos Kampanakis, Mike Ounsworth, and Serge Mister. Multiple Public-Key Algorithm X.509 Certificates. Internet-Draft draft-truskovsky-lamps-pq-hybrid-x509-00, IETF Secretariat, March 2018. `https : / / datatracker . ietf . org / doc / html / draft-truskovsky-lamps-pq-hybrid-x509-01`.

[69] European Union. Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. Official Journal of the European Union, August 2014.

[70] William Whyte, Zhenfei Zhang, Scott Fluhrer, and Oscar Garcia-Morchon. Quantum-safe hybrid (qsh) key exchange for transport layer security (tls) version 1.3. Internet-Draft draft-whyte-qsh-tls13-06, Internet Engineering Task Force, 2017. Work in Progress.

[71] WireGuard. wireguard-linux: WireGuard for the Linux kernel. `https://git.zx2c4.com/wireguard-linux`. [Accessed: 09/03/2022].

[72] WolfSSL. Hybrid post quantum groups in TLS 1.3. `https : / / www . WolfSSL . com / hybrid-post-quantum-groups-tls-1-3/`. 24/09/2021. [Accessed: 22/02/2022].

[73] WolfSSL. Quantum safety and wolfSSL. `https : / / www . WolfSSL . com / quantum-safety-WolfSSL/`. 13/09/2019. [Accessed: 18/02/2022].

[74] Philip Zimermann. *PGP(tm) User's Guide.* Pretty Good Software, 2.6.2 edition, oct 1994. `https://web.pa.msu.edu/reference/pgpdoc1.html`. Accessed: [22/02/2022].