DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Post-quantum algorithms for digital signing in Public Key Infrastructures

MIKAEL SJÖBERG

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Post-quantum algorithms for digital signing in Public Key Infrastructures

MIKAEL SJÖBERG

Master in Computer Science
Date: June 30, 2017
Supervisor at PrimeKey: Markus Kilås
Supervisor at KTH: Douglas Wikström
Examiner: Johan Håstad
Swedish title: Post-quantum-algoritmer för digitala signaturer i Public Key Infrastructures
School of Computer Science and Communication

Abstract

One emerging threat to Public Key Infrastructures is the possible development of large-scale quantum computers, which would be able to break the public-key cryptosystems used today. Several possibly post-quantum secure cryptographic algorithms have been proposed but so far they have not been used in many practical settings. The purpose of this thesis was to find post-quantum digital signature algorithms that might be suitable for use in Public Key Infrastructures today.

To answer the research question, an extensive literature study was conducted where relevant algorithms were surveyed. Algorithms with high-grade implementations in different cryptographic libraries were benchmarked for performance. Hash-based XMSS and SPHINCS, multivariate-based Rainbow and lattice-based BLISS-B were benchmarked and the results showed that BLISS-B offered the best performance, on par with RSA and ECDSA. All the algorithms did however have relatively large signature sizes and/or key sizes.

Support for post-quantum digital signature algorithms in Public Key Infrastructure products could easily be achieved since many algorithms are implemented in cryptographic libraries. The algorithms that could be recommended for use today were SPHINCS for high-security applications and possibly BLISS-B for lower security applications requiring higher efficiency. The biggest obstacles to widespread deployment of post-quantum algorithms was deemed to be lack of standardisation and either inefficient operations compared to classical algorithms, uncertain security levels, or both.

Sammanfattning

Ett nytt hot mot Public Key Infrastructures är den möjliga utvecklingen av storskaliga kvantdatorer som kan knäcka de asymmetriska kryptosystem som används idag. Ett flertal eventuellt kvantsäkra algoritmer har presenterats men de har än så länge inte sett mycket praktisk användning. Målet med detta examensarbete var att försöka identifiera eventuellt kvantsäkra signaturalgoritmer som skulle kunna lämpa sig för användning i Public Key Infrastructures idag.

För att besvara forskningsfrågan gjordes en utredande litteraturstudie där relevanta signaturalgoritmer identifierades. Därefter prestandatestades de algoritmer som var implementerade i kryptografiska bibliotek. De algoritmer som prestandatestades var de hash-baserade algoritmerna XMSS och SPHINCS, flervariabel-baserade Rainbow och gitter-baserade BLISS-B. Resultaten visade att BLISS-B hade bäst prestanda och att prestandan var i nivå med RSA och ECDSA. Samtliga algoritmer hade emellertid relativt stora signatur- och/eller nyckelstorlekar.

Eventuellt kvantsäkra algoritmer skulle redan idag kunna stödjas i Public Key Infrastructures eftersom många algoritmer finns implementerade i kryptografiska bibliotek. SPHINCS kunde rekommenderas när hög säkerhet krävs medan BLISS-B möjligtvis skulle kunna användas när lägre säkerhet kan tolereras i utbyte mot bättre prestanda. Största hindren för utbredd användning ansågs vara en brist på standardisering samt ineffektiva operationer jämfört med klassiska algoritmer och/eller tveksamma säkerhetsnivåer.

Contents

Chapter 1

Introduction

This chapter gives an introduction and a short background to the relevance of the thesis. The problem statement, aim of the thesis and the chosen research question are also presented. Additionally, some limitations imposed on the thesis are explained and motivated.

1.1 Background

Cryptography is used in all kinds of applications today where secure communication is wanted. Cryptographic encryption and signature algorithms are used to try to ensure confidentiality, integrity and authenticity of messages sent during communication [1]. One form of cryptography is known as *public-key cryptography*, where each entity has a private key and a public key. In a public-key signature scheme, the signer has a private signing key that can be used to sign messages. The public key, which can be shared with anyone, can be used to verify that the signature is valid and, if the signature scheme is secure, that no one but the signer could have generated the signature.

In order to bind identities to public keys, Public Key Infrastructures (PKIs) are often used. Certificate Authorities (CAs) are a central part of PKIs. A CA is a mutually trusted party that uses digital signature algorithms to sign certificates containing a public key and information of its owner [2]. This allows anyone who trusts the CA to also trust the public key by verifying that the certificate has a valid signature.

The security of public-key cryptography is based on number theoretic problems that are thought to be hard to solve for anyone without access to the information available in the private key. One example is the public-key algorithm RSA, which is based on the hardness of prime factorisation of large integers. It must be noted that there are no proofs that RSA or any other cryptographic algorithm is completely secure. Instead, the RSA algorithm has been under scrutiny for decades without any major breakthroughs in solving the factorisation problem efficiently, which makes most people believe that it is secure. The security of public-key cryptography used today might however be at risk by a new emerging threat: quantum computers.

During the last few years there have been several advances in research on developing quantum computers [3]. There are even small-scale quantum computers available to the public today [4]. If a large-scale quantum computer is built in the future, the public-key cryptosystems that are in use today would be broken by an algorithm developed by Shor [5] in 1994. This is because Shor's algorithm is capable of factorising integers and finding discrete logarithms, the cornerstones of traditional public-key cryptosystems, in polynomial time.

Researchers have estimated that quantum computers capable of breaking RSA-2048 might be available in 2030 at a cost of approximately one billion dollars, something the National Institute of Standards and Technology (NIST) sees as a serious threat to current cryptosystems [6]. The European Telecommunications Standards Institute (ETSI) has been even more cautious by recommending any organisation with a need to archive encrypted data longer than 2025 should be worried about quantum computers [3]. To counteract this threat, standardisation institutes have started looking at standardising post-quantum algorithms, i.e. algorithms that are thought to be safe from attacks from quantum computers [6, 3]. NIST has started the process by calling for post-quantum algorithm proposals to be standardised, ending in December 2017 [7].

Deployments of post-quantum algorithms in vendor applications have been rare, most likely due to lack of confidence in the security of post-quantum algorithms. Post-quantum algorithms also often have worse efficiency compared to currently used algorithms and no post-quantum algorithm has so far been standardised. One way to promote further research and guide standardisation might be to develop proof-of-concepts where post-quantum algorithms are implemented in existing software solutions. Several such proof-of-concepts for post-quantum key-exchange algorithms have been developed, for example for TLS [8] and OpenVPN [9], but so far few are available for digital signing in Public Key Infrastructures.

1.2 Problem statement

Even though the number of available post-quantum digital signature algorithms is large, there had been no research on their practical usability in PKIs prior to this thesis. Thus the problem at hand was to survey the large number of post-quantum algorithms available in order to find some candidate algorithms that could be implemented by the PKI community in a near future. This involved finding the necessary requirements on a digital signature algorithm used in a PKI. In addition to finding specific algorithms, identifying characteristic properties of some algorithm families was deemed to be helpful in order to guide PKI community in finding suitable algorithms for the future.

1.3 Aim

The aim of this thesis is to survey the post-quantum digital signature algorithms available today in regards to their usability in digital signing in PKI. Using the results from the survey are to be used to identify several candidate algorithms suitable for digital signing in PKI and that could be implemented in PKI vendor products in a near future. The suitability will be determined by studying the code availability in cryptographic libraries, benchmarking the performance of the algorithms and comparing it to the needs of PKIs and applications relying on PKIs.

By analysing and comparing several post-quantum algorithms for a specific use case it could be possible to identify important requirements for post-quantum algorithms in PKI. The results could also give a better understanding about what the largest hindrances are to widespread deployment of post-quantum algorithms.

In addition, proofs-of-concept X.509 certificate are to be generated and signed using post-quantum digital signature algorithms. Showing the proofs-of-concept could hopefully help drive research and company interest forward in the field of post-quantum cryptography.

1.4 Research question

The research question studied in this thesis is:

> *What post-quantum digital signature algorithms available today are suitable for digital signing in Public Key Infrastructures?*

The research question was deemed broad enough to encompass surveying a large number of post-quantum digital signature algorithms while still being limited enough to find concrete results. The thesis focuses on digital signing in PKI in order to find algorithms suitable for a specific use case. Post-quantum encryption and key-exchange algorithms are other interesting and relevant topics for PKI that were left for future work.

1.5 Limitations

For a post-quantum digital signature algorithm to be considered suitable for deployment in PKI today it must have a working implementation. Algorithms without publicly available, high-grade implementations were surveyed and discussed but not included in the performance benchmarks.

Studying the security of post-quantum algorithms in depth by performing cryptanalysis is also outside of the scope of this thesis. The algorithms considered therefore had to have estimated security levels for both classical and quantum security. Security levels were determined by examining the original papers and any eventual cryptanalyses done by other researchers.

Due to a limited amount of time and resources available for this thesis, all available post-quantum digital signature algorithms could naturally not be researched in detail. In order to still have a good survey coverage, algorithms from the most widely recognised categories of post-quantum algorithms were researched to at least some extent in order to find good candidate algorithms.

1.6 Outline

In *Chapter 2*, the concept of Public Key Infrastructures is presented. The focus lies on PKIs using X.509 certificates.

In *Chapter 3*, the concept of post-quantum cryptography is explained. This includes explaining how and why currently used cryptosystems can be broken by quantum computers. Shor's algorithm and Grover's algorithm are explained briefly. Furthermore, several post-quantum digital signature algorithms are explained with some technical details omitted. For more in-depth information about an algorithm, the reader is directed to the literature.

In *Chapter 4*, the methodology used to produce the results of the thesis are presented. It includes an extensive literature study providing necessary background knowledge on

PKIs and post-quantum cryptography. Empirical data was gathered from a performance benchmark on some of the algorithms identified during the literature study.

In *Chapter 5*, requirements on a digital signature algorithm used in a PKI are identified and ranked. The post-quantum algorithms chosen for further analysis from the literature study are compared with regards to security levels, signature sizes, key sizes and code availability.

In *Chapter 6*, the performance benchmark is explained and the empirical evidence consisting of experimental data is presented. The benchmark measured average running times, median running times and sample standard deviations for key generation, signature generation and signature verification using four different post-quantum algorithms: XMSS, SPHINCS, Rainbow and BLISS-B.

In *Chapter 7*, discussions about the benchmark results are presented together with more general discussions about post-quantum algorithms in PKI. Some recommendations for the PKI community are also presented.

In *Chapter 8*, some concluding remarks and ideas for future work are presented.

Chapter 2

Public Key Infrastructure

Public Key Infrastructures (PKIs) are used to ensure the efficient and secure management of cryptographic public key pairs during their whole life cycle [2]. The life cycle of a key pair can be divided into three steps: Key generation, key usage, and key invalidation.

In the key generation step, a new key pair is created. This can be done either by the end-entity, by hardware such as smart-cards or Hardware Security Modules (HSMs), or by some authority in the PKI [2]. Regardless of how the key pair is generated, the PKI must ensure that the key pairs are secure. If end-entities generate their own key pairs they can prevent the private keys from being exposed to any unauthorised persons.

In the key usage step, digital signature and encryption operations are performed using the key pair previously generated [2]. Signing and decryption is done using private keys while signature verification and encryption is performed using public keys. In this step, the PKI must ensure that end-entities can access the public keys of other end-entities in order to verify signatures and encrypt data. The PKI must also make it possible for end-entities to verify the authenticity and validity of a public key, as well as knowing its properties. Properties might for example be the allowed key usage or the security policy applied when generating the key.

In the key invalidation step, the key pair becomes invalid for some reason. Such reasons might be that the validity period of a key pair has ended or that a private key has been compromised [2]. A key pair can for example be compromised by a smart-card being stolen, a computer being infected by malware or, relevant to this thesis, if the underlying cryptosystem has been broken by a quantum computer. When key pairs have been compromised, the PKI should make sure that all users are made aware of the compromise and stop using and trusting the compromised keys.

One of the most common ways of storing and distributing public keys is in the form of certificates. More specifically, the standardised X.509 Public Key Certificate format [10] is used in many commercial applications [2]. Other types of certificates exist as well but in this thesis the focus lies on X.509 certificates. Certificates are used to bind public keys to entities. This means that the identity of the certificate owner must be established in a secure way. This is usually done by a Registration Authority (RA) in the PKI. After the identity of the entity has been verified, the RA sends the information to a Certification Authority (CA). Information exchanged between RAs, CAs and other parts of a PKI are cryptographically protected by encryption or digital signatures. After the CA receives the information it needs it generates the certificate and signs it using the CA's private signing key. Any certificate issued by the CA can later be verified using the CA's public key included in the CA certificate. In a PKI, the CA is seen as a mutually trusted third party. This makes it possible for entities to trust each other indirectly through their direct trust in the CA.

2.1 X.509 certificates

The standardised X.509 certificate is a public key certificate format, encoded using the ASN.1 Distinguished Encoding Rules (DER) [10]. An X.509 certificate consists of the following three elements:

- *tbsCertificate* - The To-Be-Signed public key certificate
- *signatureAlgorithm* - An algorithm identifier, consisting of OID and optional parameters, for the signature algorithm used by the CA to sign the certificate
- *signatureValue* - A bit string containing the value of the digital signature

The minimum contents of an X.509 TBS certificate are:

- *version* - X.509 certificate version (if not present, version 1 is presumed)
- *serialNumber* - A unique serial number for each certificate issued by the issuing CA
- *signature* - An algorithm identifier of the signature that must be the same as signatureAlgorithm
- *issuer* - Identifies the issuing CA
- *validity* - Validity period of the certificate
- *subject* - Identifies the entity associated with the public key stored in the certificate
- *subjectPublicKeyInfo* - The algorithm identifier describing the public key algorithm and the value of the public key

Optional contents for X.509v3 certificates are:

- *issuerUniqueId*
- *subjectUniqueId*
- *extensions* - Such as allowed key usage, basic constraints and any custom extension

The X.509 standard does not impose any restrictions on the type of public key or the digital signature algorithm used for signing the certificate. Furthermore, the X.509 standard allows arbitrary length signatures and public keys. This makes X.509 certificates highly flexible when transitioning to new cryptosystems. However, other protocols might impose certain size limitations on X.509 fields [3].

2.1.1 X.509 certificate generation

A certificate is generated after a certification application has been initiated by some entity in the PKI [2]. The application is followed by a registration, for which the RA is responsible, where the identity of the certificate owner and all other information relevant to issuing a certificate is collected and verified. The RA then forwards this information to the CA that is to issue a certificate.

In addition to the registration information, the CA needs the public key that is to be included in the certificate before a certificate can be issued. This is either done by having the CA generate the key pair or by letting the end-entity generate it and present the public key to the RA during registration. One way to apply for a certificate while keeping the private key hidden from the CA is to issue a Certificate Signing Request (CSR), such as PKCS#10 [11], to a CA. The CSR contains information about the applicant and the public key to be included in the certificate. A PKCS#10 CSR is self-signed using the corresponding private key.

With the registration information and the public key of the certificate owner, the CA can issue the certificate. This is done by digitally signing the certificate using the CA's private signing key. After the certificate has been issued and verified to be correct by the certificate owner, it can be distributed and used by other entities.

Decryption keys and signature keys should be different and consequently, the corresponding encryption keys and signature verification keys should be stored in separate certificates. In the case of RSA keys, incorrect usage of the same key pair for decryption and signing could lead to an adversary being able to decrypt messages by tricking the signer into signing encrypted messages. Another reason for having different keys is to enable key escrow for decryption keys. Key escrow means that the private key is not only stored by end-entities but also by another trusted party [2]. This is a safety measure to make sure that encrypted data can be accessed even if an entity loses its private decryption key. However, in order to maintain the non-repudiation property, private signing keys are usually not held in key escrow. This is because the consequence of losing a private signing key is simply that a certificate must be issued for a new key pair.

2.1.2 X.509 certificate validation

The validity of certificates needs to be verified to ensure that the public key does in fact belong to the certificate owner. Verification is performed by verifying the certificate signature, checking the validity period of the certificate and the revocation status [2].

To verify a certificate signature, the verifier uses the issuer's public key, obtained from the issuing CA's certificate. Different digital signature algorithms can be used for signing and verification of certificates in a PKI. Two signature schemes commonly used today are RSA and the Elliptic Curve Digital Signature Algorithm (ECDSA).

RSA RSA is an encryption algorithm that can also be used for digital signatures. The security of RSA is based on the hardness of prime factorisation of large integers [12]. An RSA key pair consists of a public encryption key (e, n) and a private decryption key (d, n), where e, d and n are positive integers. A message M is represented as an integer between 0 and $n-1$, where longer messages are split into series of such blocks. A ciphertext C is generated by calculating:

$$C \equiv M^e \mod n$$

The ciphertext C is decrypted by calculating:

$$M \equiv C^d \mod n$$

This works due to the mathematical relationships between e, d and n explained briefly below.

The modulus n is the product of two large, randomly chosen, primes p and q. For example, in the RSA-3072 scheme, n is a 3072-bit integer. The value of d is chosen to be a large, random integer that is relatively prime to $(p-1)(q-1)$. The value of e is computed from p, q and d to be the multiplicative inverse of d modulo $(p-1)(q-1)$. This means that:

$$e \cdot d \equiv 1 \mod (p-1)(q-1)$$

If these properties are fulfilled then the encryption scheme works because:

$$C^d \equiv (M^e)^d \equiv M^{e \cdot d} \equiv M \mod n$$

RSA can be used as a digital signature scheme by using RSA encryption "in reverse". The general idea is that to sign a message M, the signer will first compute a message hash $h = H(M)$ using some cryptographic hash function H. The signer then encrypts the message hash h using the signer's private decryption key to obtain h'. The signed message (M, h') is sent to the verifier. A verifier can then verify the message by decrypting h' using the signer's public key and verifying that it is equal to the message hash of M. This works due to the special property of RSA that:

$$E\big(D(M)\big) = D\big(E(M)\big),$$

where E is the encryption operation and D is the decryption operation.

To make the signature scheme properly secure an advanced padding scheme, as specified in the PKCS#1 specification [13], should be used. In order to speed up the signing operation, RSA private keys today usually contain several additional values. In the PKCS#1 specification, an RSA private key contains:

$$\big(n, e, d, p, q, (d \mod (p-1)), (d \mod (q-1)), (q^{-1} \mod p)\big)$$

This means that for RSA-3072, the private key becomes roughly 1728 bytes. The public key is (n, e), which produces a public key of roughly 384 bytes. This is because in practice e is usually chosen to be a small integer, such as 3 or 65537, which speeds up the verification operation.

ECDSA The Elliptic Curve Digital Signature Algorithm (ECDSA) is a signature algorithm based on the hardness of solving discrete logarithms in elliptic curve groups [14]. Prior to using ECDSA, the signer and verifier must decide on a set of elliptic curve domain parameters to be used. A large number of standardised curves exist and are used today but it is also possible to use custom curves. One standardised curve is the curve NIST P-256, also known as *secp256r1* or *prime256v1*, where all operations are performed modulo a 256-bit prime integer.

To generate an ECDSA key pair the signer first chooses a random secret integer:

$$d \in \{1, n-1\},$$

which acts as the private key in the scheme. The public key is an elliptic curve point $Q = dG$, where G is the base point generator of an elliptic curve group.

To sign a message, a random value $k \in \{1, n-1\}$ is first selected. An elliptic curve point is then computed as:

$$kG = (x_1, y_1)$$

One of the signature values, r, is then computed as:

$$r = x_1 \mod n$$

If r happens to be equal to 0, the process is repeated with a new random value k. A method for converting field elements to integers exists making the previous computation possible. The signer then proceeds by computing $e = H(m)$, where H is a cryptographic hash function. Finally, the signature value s is computed as:

$$s = k^{-1}(e + dr) \mod n$$

If $s = 0$, a new k is chosen and the whole process is repeated. Otherwise, the signature generation is successful and the signature (r, s) is returned.

To verify this signature, the verifier first ensures that:

$$r, s \in \{1, n-1\}$$

A hash value $e = H(m)$ and the two values

$$u_1 = es^{-1} \mod n$$

$$u_2 = rs^{-1} \mod n$$

are then computed. An elliptic curve point

$$X = u_1G + u_2Q$$

is computed and, if $X = (x_1, y_1)$ is not equal to the identity element, the value

$$v = x_1 \mod n$$

is computed. A signature is accepted if $v = r$. The signature verification is valid since:

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv u_1 + u_2d \pmod{n}$$

and since:

$$u_1G + u_2Q = (u_1 + u_2d)G = kG,$$

it is required that $v = r$ for a signature to be valid.

For ECDSA over the curve NIST P-256 with $n = 256$, the public key size is $2n = 512$ bits and the private key size is 768 bits since the private key usually contains both d and Q. Signatures are two n-bit integers, making the signature size 512 bits.

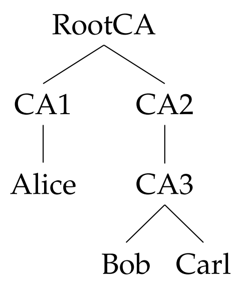

Figure 2.1: A hierarchical CA structure

Trust models A common trust model used in PKIs is the hierarchical trust model [2]. In a hierarchical PKI, trust in public keys depends on the trust of a uniquely determined certificate signer, a trust anchor. The trust anchor is an entity that all entities directly trust. The trust anchor is often called the root CA whilst other CAs in the same hierarchy are called intermediate CAs. In order to validate a certificate, the verifier must have an unbroken chain of certificates from the certificate to be verified to the trust anchor.

Figure 2.1 shows an example where the trust anchor and root CA is *RootCA*, nodes *CA1-3* are intermediate CAs and leaf nodes are end-entities. It can be assumed that all entities trust RootCA and is in possession of RootCA's certificate.

If the user Alice wants to validate the certificate of Bob, Alice must be in possession of the certificates of RootCA, CA2 and CA3. In many protocols, this is achieved by Bob providing a certificate chain to Alice, consisting of his own certificate and all CA certificates up to the root CA. The root CA certificate is in many cases not transferred since it is assumed that everyone in the PKI is already in possession of it. Alice can then verify the authenticity of Bob's certificate by first verifying that the certificate signature in Bob's certificate was signed by CA3. Alice then verifies that CA3's certificate was signed by CA2. Lastly, Alice verifies that CA2's certificate was signed by RootCA and if all signatures were valid, Alice trusts Bob's certificate through her trust in RootCA.

In the case that Bob wants to communicate with Carl, CA3 can be seen as a root CA since it is trusted by both parties.

2.1.3 X.509 certificate revocation

Certificate revocation, the invalidation of public keys, can be done in a variety of ways. Two commonly used ways are distribution of Certificate Revocation Lists (CRLs) [10] and offering an online service with the Online Certificate Status Protocol (OCSP) [15].

A CRL is a list of identifiers for all certificates issued by a CA that have been revoked [2]. To prove the authenticity of the CRL, it is digitally in a similar way to X.509 certificates. CRLs can be directly signed by the CA who issued the revoked certificates or indirectly by an appointed CRL issuer. The CRL must be made available to all entities that perform authentication and updated periodically to ensure that no entities incorrectly trust a revoked certificate. A verifier will download the CRL, verify the signature and then check if a specific certificate is in the CRL.

One drawback of CRLs is that they increase in size and can become quite large over time if expired certificates are not removed [2]. One solution to this problem is to use delta-CRLs, containing only identifiers for certificates that have been revoked since a certain complete CRL was issued, the *Base CRL*. Delta-CRLs need to be signed with the same signature key used to sign the Base CRL. By issuing delta-CRLs it is possible to issue smaller updates to the CRL with a higher frequency.

A CA using OCSP will instead have an online service that entities can query to learn the revocation status of a specific certificate [15]. All OCSP responses need to be digitally signed. One advantage of using OCSP is that a verifier only needs to fetch information about the specific certificate to be verified (in contrast to CRLs where information about all revoked certificates are retrieved). Furthermore, information about a revoked certificate can be retrieved almost immediately after it has been revoked instead of having to wait for a new CRL to be released.

2.2 Applications relying on PKI

PKIs are used for a broad number of applications that require authentication of entities and public key distribution. This section will describe a number of such applications in order to better understand the practical requirements of the signature algorithm used in a PKI.

2.2.1 Internet (TLS)

One important use of PKIs today is maintaining certificates for use in the TLS protocol [16] on the Internet. The TLS protocol supports confidential and authenticated channels between clients and servers [2]. TLS is used in combination with a multitude of other protocols such as HTTPS, IMAP, SMTP and FTP. The HTTPS protocol uses TLS to authenticate web servers and establish secure communication between a connecting client and the server by exchanging a symmetric session key. During the TLS handshake, the server's certificate and any intermediate CA certificate's are transferred to the client. The client then verifies the server certificate by verifying the whole certificate chain leading up to a root CA that the client knows and trusts.

TLS can also be used for mutual authentication by requiring client authentication, which is needed when a client needs to verify itself before it can connect to a secure service. The client then needs to send a valid certificate to the server during the TLS handshake. Client authentication can for example be used to replace password logins.

The total amount of data sent during the TLS handshake depends on the amount of certificates that need to be transferred and consequently, the size of signatures and public keys contained in those certificates. For server authentication, the certificate of the server along with all certificates up to the root certificate, the certificate chain, must be sent to the client [16]. The server can choose to not send the root certificate because it is presumed that it has already been distributed to the client in some other way, for example coupled with a web browser installation or operating system. For client authentication, the client also needs to send its own certificate along with its certificate chain.

2.2.2 E-mail (S/MIME)

A common method for enabling signing and encryption of e-mails is to use the security standard: Secure/Multipurpose Internet Mail Extensions (S/MIME) [17, 18]. S/MIME allows users to have confidential e-mail communication and verify both authenticity and integrity of e-mails.

A sender will sign an e-mail using its private signing key. Encryption of an e-mail is performed using the recipients public encryption key found in the recipient's certificate. The sender can then send the e-mail to the recipient who can decrypt the e-mail and verify the sender.

Certificate distribution in S/MIME can be performed in several different ways. The simplest form of distribution is to manually distribute the certificate to all recipients. The certificate can also be sent by the sender as an e-mail attachment when establishing first contact. A more scalable solution is for the recipient to do a database or directory lookup to find certificates.

To minimise data transfers, certificates are often stored in the recipient's e-mail client. The recipient must however still verify the validity of the stored certificate by retrieving revocation information from the CA.

2.2.3 Code, document and file signing

Code signing is the process of digitally signing software distributions, including software updates, to ensure authenticity and integrity of the software [2]. Code signing helps protect against viruses and Trojan horses since a user can verify that downloaded software comes from the correct source and has not been tampered with.

Similarly to code signing, document signing and file signing is used to ensure authenticity and integrity of various documents and files. The most prominent example is legal documents which can be signed using a digital signature instead of a physical signature in many countries [2].

In order to allow Long-Term Validation, validation of signatures that are valid even after the signer certificate has expired, a Time-Stamping Authority (TSA) can be used. The TSA is used to timestamp a datum to provide a proof-of-existence at a certain point in time [19]. The timestamping process can be used to verify that a valid digital signature existed at a certain point in time. In this process, the trust is transferred from the original signer to the TSA.

Signing services and TSAs might have certain customer or regulatory requirements on the signature generation time. For example, a TSA needs to provide timestamps with an accuracy of 1 second or better as specified in [20].

2.3 The future of PKI

With the emerging Internet of Things (IoT) trend, the need to secure potentially billions of devices on the Internet has become apparent and one way to do it is by the use of PKI. For example, CSS predicts that PKI will emerge as the best practice for identification, authentication and secure communications for IoT devices [21]. They also state that this will increases the need to find scalable, cost-effective and efficient solutions for secure authentication using PKI.

There is also a need to find scalable solutions for the many connected vehicles and one way to do it is with a vehicular PKI. Different methods have been proposed to protect the privacy of connected vehicle owners [22]. One proposed method is to have CAs periodically issue a batch of short-lived certificates for each vehicle. The short-lived certificates would then be used for only a short time before being discarded in order to make it difficult for an adversary to track a specific vehicle. This does however increase the importance of having an efficient CA and consequently, an efficient signature algorithm.

The PKI architecture will most likely continue being relevant for years to come due to an increasing number of connected devices. It is therefore important to identify not only what needs to be done to make PKIs more efficient but also to identify what needs to be done to secure PKIs from different threats. A big threat to cryptography in general, which means it is also a threat to PKI, is the threat from quantum computers that can break the currently used cryptographic algorithms. The quantum threat and ways to protect against it are presented in the next chapter.

Chapter 3

Post-quantum cryptography

Post-quantum cryptography, also known as quantum-safe or quantum-resistant cryptography, is cryptography using cryptosystems that are thought to be secure against algorithms running on quantum computers. The security of public-key cryptosystems used in practice today depends on number theoretic problems that are deemed to be intractable on classic computers [23]. The two most commonly used number theoretic problems are prime number factorisation of large integers and finding discrete logarithms, which are problems that have been studied for a long time. Cryptosystems based on these problems include RSA, DSA, and ECDSA. A large-scale quantum computer would be able to break all these widely used cryptosystems in polynomial time using a quantum algorithm known as *Shor's algorithm* [5]. Key-exchange protocols such as Diffie-Hellman and Elliptic Curve Diffie-Hellman are also broken by future quantum computers since they rely on the same discrete logarithm problems.

Symmetric key cryptography schemes such as AES are also at risk by *Grover's algorithm* [24], a quantum search algorithm with quadratic speedup compared to classical search algorithms [23]. However, the threat is not as critical since it can easily be countered by doubling the key length. For example, using AES-256 instead of AES-128 will ensure 128-bit security against quantum adversaries. The same search algorithm can be applied to hash functions so using hash functions with larger output sizes is necessary to protect against quantum attacks.

In the remainder of this chapter the basics of quantum computers and quantum algorithms are explained. Furthermore, a large set of post-quantum digital signature algorithms are explained. Hash-based, lattice-based and multivariate-based signature schemes are explained in more detail while code-based and elliptic curve isogeny-based signature schemes are only touched upon briefly.

3.1 The quantum threat

The quantum threat to cryptography comes from the many recent advancements in the field of quantum computing. No one knows exactly when large-scale quantum computers will be available or even if a large-scale quantum computer can be built. However, recent advances in the field have led some to believe that the time until large-scale quantum computers are available might be as soon as the year 2025 [3].

Quantum computers are computers based on quantum mechanics and the state of quantum computers can not be described by a single string of bits in the same way as classical computers can be [23]. Instead, the state of a quantum computer is expressed using quantum bits, or *qubits*. Qubits are more powerful in the sense that they allow a quantum computer to be in a superposition of states, compared to classical computer which are limited to being in a single state. Quantum algorithms make use of this and transforms all the states at once. These properties enable a quantum computer to run algorithms that can solve problems deemed intractable for classical computers, which are restricted to a single state.

The reason quantum computers are considered a threat to current public-key cryptosystems is mainly because of two proposed quantum algorithms: *Shor's algorithm* [5] and *Grover's algorithm* [24]. The general ideas behind the two algorithms are explained briefly below.

Shor's algorithm The quantum algorithm known as Shor's algorithm [5] can be used factor integers and find discrete logarithms in polynomial time using a quantum computer. This is an exponential speedup compared to the best known classical algorithm, the number field sieve, which runs in sub-exponential time.

The algorithm for factorisation makes use of the fact that the factorisation of n can be reduced to finding the order of an element x in the multiplicative group $\pmod{n}$. In other words, finding r such that:

$$x^r \equiv 1 \pmod{n}$$

The algorithm chooses a random integer $x \pmod{n}$ and finds its order r in polynomial time using a quantum computer. The algorithm then computes:

$$gcd(x^{r/2} - 1, n),$$

where *gcd* is the Greatest Common Divisor. Due to the following relationship:

$$(x^{r/2} - 1)(x^{r/2} + 1) = x^r - 1 \equiv 0 \pmod{n},$$

the value of $gcd(x^{r/2} - 1, n)$ will only fail to be a non-trivial divisor of n if r is odd or if:

$$x^{r/2} \equiv -1 \pmod{n}$$

This means that a non-trivial factor will be found with probability at least:

$$1 - \frac{1}{2^{k-1}},$$

where k is the number of distinct odd prime factors of n. If the algorithm fails, the algorithm can be repeated with a new random integer x.

The algorithm for finding discrete logarithms uses modular exponentiation and a quantum operation called quantum Fourier transform to find a discrete logarithm in polynomial time [5]. The algorithm has later been expanded to also find discrete logarithms in elliptic curve groups [25]. The elliptic curve discrete logarithm algorithm was shown to be more efficient than the factoring algorithm, potentially requiring a quantum computer with less than half as many qubits for large values of n. For example, factoring a 3072-bit RSA modulus was estimated to require around 6144 logical qubits while an equally secure 256-bit elliptic curve cryptographic key was estimated to require only 1800 logical qubits to break. It should be noted that for a quantum computer to have a certain amount of logical qubits, the number of physical qubits will need to be several times higher.

Grover's algorithm The quantum algorithm known as Grover's algorithm [24] is a search algorithm capable of finding an element in an unordered database of $N = 2^n$ elements in only $\mathcal{O}(\sqrt{N})$ steps on a quantum computer. This is a quadratic speedup compared to the classical computer approach, which requires on average $\frac{N}{2}$ steps using linear search.

The algorithm works by first initialising the system to a distribution with the same amplitude to be in each of the N number of n-bit states, where the square of the absolute value of the amplitude in a state equals the probability to be in that state. The algorithm then repeats a loop of operations $\mathcal{O}(\sqrt{N})$ times. The operations performed are an evaluation of the state by a quantum oracle, a conditional phase rotation depending on the previous state evaluation and a diffusion transform.

For each iteration of the previously mentioned loop, the amplitude in the desired state is increased by $\mathcal{O}(\frac{1}{\sqrt{N}})$. This means that after $\mathcal{O}(\sqrt{N})$ iterations, the amplitude in the desired state reaches $\mathcal{O}(1)$. When sampling the resulting state it will be in the desired state, meaning that the searched element has been found, with probability at least $\frac{1}{2}$.

3.2 Security level estimation

No cryptosystems are proven to be secure against all attacks so the security level of a cryptographic scheme can only be estimated. Estimating the security of a cryptographic algorithm is not easy. One common way to present the estimated security level of a cryptographic scheme is to use the notion of *bit security*. An attack against a cryptographic scheme with an estimated b-bit security level can be expected to require $\mathcal{O}(2^b)$ operations [26].

The same notion of bit security has been adopted in several papers when talking about attacks using quantum computers as well. In this thesis, the distinction is made by using the term *classical bit security* when talking about adversaries using classical computers and *quantum bit security* when talking about adversaries with access to quantum computers.

For most common public-key cryptosystems, the security relies on well-defined mathematical problems that are conjectured to be difficult to solve [26]. The security depends on the fact that no efficient solutions exist to those problems. The bit security measurement is obtained by looking at all known general attacks on the mathematical problem and the cryptographic scheme as a whole. This naturally means that unknown attacks might exist, which could potentially make the security non-existent. Confidence in the security level of an algorithm therefore heavily relies on the underlying building blocks of an algorithm, their security assumptions and the amount of scrutiny they have been under.

3.3 Hash-based signature schemes

Like all signature schemes, hash-based signature schemes use cryptographic hash functions [23]. The security of many hash-based signature schemes relies solely on the security of the underlying hash function instead of the hardness of a mathematical problem. It has been shown that one-way functions, such as a cryptographic hash function, is necessary and sufficient for secure digital signatures [27]. This means that hash-based sig-

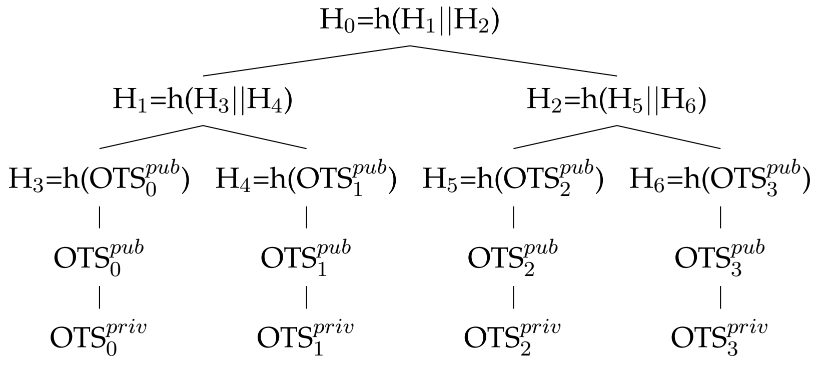

Figure 3.1: An example of a small Merkle tree that can be used to sign four messages using OTS key pairs

nature schemes can be seen as the most fundamental type of digital signature schemes [23].

Another advantage of hash-based signature schemes is that they are not tied to a specific hash function [28]. As long as the hash function is considered secure, it is possible to change it for increased efficiency or security. Hash functions have a limited lifespan, so being able to replace one without changing the underlying structure contributes to the longevity of hash-based signature schemes.

Hash-based signature schemes are a relatively old invention, starting with the Lamport One-Time Signature (OTS) scheme [29]. In the Lamport OTS scheme, the signer chooses pairs of random integers that are kept as the private key. The public key is the hashes of those random integers. To sign a message, the signer reads the message bitwise and presents one value from each secret integer pair depending on the bit value. The verifier can then verify that the hash of all secret integers is equal to the corresponding hash value in the public key.

It is clear that a OTS key pair can be used only once since it reveals information of the private key. This makes it impractical for many real-world applications. To solve this, the use of OTS schemes was later expanded upon by Merkle's tree scheme [30], which creates a binary hash tree structure in which each leaf represents a OTS key pair. This makes it possible to sign several messages by using a different OTS key for each message, while still having only a single public key for the Merkle tree scheme.

Figure 3.1 shows a small Merkle tree as an example. The public key of a Merkle tree signature scheme is the value of the root node, H_0. The value of a node is equal to the hashed value of the concatenation of both its child nodes. The values of the lowermost level of nodes, H_3-H_6 are equal to the hashed value of a OTS public key. The signature in a Merkle tree scheme consists of a OTS and an authentication path consisting of the necessary hash values to reach the root from the leaf, that is one hash value for each level of the Merkle tree. For example, a signature generated with the private key OTS_0^{priv} contains the values H_4 and H_2 so a verifier can verify that:

$$h(h(OTS_0^{pub}||H_4)||H_2) = H_0$$

The authentication path proves that the provided OTS public key is in fact a key in the Merkle tree scheme.

Even though the advantages of hash-based signature schemes seem to be many, there are some downsides as well. The primary downside is that most hash-based signature schemes are stateful [28]. This is due to the fact that a OTS key pair can be used only once and therefore whenever a signature is generated, the private key must be updated as well. This does not fit common software interfaces, impacts performance and makes key storage conditions more complicated. Copying or backing up a key must be avoided or handled with extreme care to not compromise the entire system. Another downside is that the number of signatures that can be generated from a key pair is limited.

The remainder of this section will explain two promising hash-based signature schemes that exist today. The two schemes are XMSS [31], which is a stateful scheme, and SPHINCS [32], which is stateless. The security of both schemes depends only on the properties of cryptographic hash functions.

3.3.1 XMSS and XMSS^{MT}

XMSS [31], the eXtended Merkle Signature Scheme, is one of many variants of the Merkle tree scheme and has been prepared for standardisation by describing the algorithm in an Internet Draft [33]. Several previous variants have been proposed by the same authors and XMSS can be seen as a more efficient and more secure version of these. The security of XMSS is based solely on the properties of cryptographic hash functions. More specifically, only a second-preimage resistant function family and a pseudorandom function family is required. XMSS can also be instantiated to be provably forward secure in the standard model. Forward security means that even if the private key is compromised, all signatures created before the compromise remain valid. Forward security can be replaced by existential unforgeability under chosen message-attacks to obtain a more efficient scheme.

XMSS uses, just like the Merkle signature scheme, a binary hash tree with OTS key pairs as leaf nodes [31]. The OTS scheme used is a slightly modified version of Winternitz-OTS (W-OTS), first proposed in [30]. The modified version eliminates the need for a collision resistant hash function family.

Public and private keys There are several different ways for increased performance or smaller size. The smallest XMSS private key consists of only a cryptographic seed for a pseudorandom function. The pseudorandom function is then used to generate the W-OTS keys when needed and the leaf index i corresponding to the next W-OTS key pair to be used. A cryptographic seed for randomised hashing and the public key can also be part of the private key. An XMSS public key consists of the root node value and bitmasks used in intermediate levels of the hash tree.

Signature generation To sign a message using XMSS, a current leaf index i is used to determine which W-OTS key pair should be used [31]. The signature (i,σ,AUTH) consists of the index i, the W-OTS signature σ and the authentication path AUTH for the leaf node. The authentication path consists of the hash values of H different nodes in the XMSS tree, one for each layer of the tree. After a message has been signed, the current leaf index i contained in the XMSS private key is updated.

Signature verification To verify an XMSS signature, the verifier first verifies the W-OTS signature σ using the corresponding W-OTS public key that is generated by the verifier.

The verifier then verifies the authentication path by traversing the tree using AUTH to obtain p_H. If p_H is equal to the root node value in the XMSS public key, the signature is accepted. If not, the signature is rejected.

Statefulness It is clear that XMSS is stateful since the value i must be updated after a signature has been generated. Due to this fact, and the fact that there are a limited number of states, the number of signatures that can be created from a single key pair is limited. The maximum number of signatures that can be created is 2^H, where H is the height of the XMSS tree. In an Internet Draft for XMSS [33], a number of different parameter sets are proposed. The value for H in those sets range from 10 to 20, giving a maximum of roughly a thousand to a million signatures.

XMSS$^{\text{MT}}$ XMSS$^{\text{MT}}$ [34], Multi Tree XMSS, is an extension of the regular XMSS. To increase the maximum number of signatures, XMSS$^{\text{MT}}$ builds several layers of XMSS trees. The lowermost layers of XMSS trees are used to sign messages while trees on higher layers are used to sign the roots of XMSS trees on the layer below. Using XMSS$^{\text{MT}}$ it is possible to sign a virtually unlimited number of messages. The downside is that signature sizes increase and signing operations require more computations.

Attacks, parameter sets and implementations In [35] a multi-target attack against hash-based signature schemes such as XMSS and SPHINCS is presented and an improved scheme for XMSS, named XMSS-T, that is not susceptible to multi-target attacks is constructed. The attack stems from the fact that the same hash function key is used several times. For a scheme such as XMSS$^{\text{MT}}$ with $H = 60$, an attacker can learn $d = 2^{66}$ outputs of the same hash function [35]. An attacker will then be able to invert one of the d values with probability $d/2^n$ instead of the wanted probability $1/2^n$. This means that if it suffices to invert the hash function on any one out of d outputs to break the security of the scheme, the attack complexity is reduced from $\mathcal{O}(2^n)$ to $\mathcal{O}(2^n/d)$.

To mitigate the attack, XMSS-T uses a new hash tree construction and a new W-OTS variant. The main idea is that a different hash function key is used for every hash function call within a hash tree or hash chain. The original XMSS scheme has been discarded in favour of XMSS-T as can be seen in the Internet Draft for XMSS [33], where XMSS-T is actually presented.

An example of a parameter set for XMSS is the XMSS_SHA2-256_W16_H10 parameter set proposed in the Internet Draft [33], which aims to provide 256-bit classical security and 128-bit quantum security in the standard model. With $H = 10$ it is possible to generate 2^{10} signatures from a single key pair but with other parameters it is possible to reach 2^{20} signatures. The signature size is 2500 bytes and public keys (excluding OID) are 64 bytes. The size of the private key differs depending on the implementation since it can be reduced heavily by sacrificing performance. This is due to the fact that the W-OTS keys can either be stored as part of the private key or be generated using a pseudorandom function from a secret 32 byte cryptographically secure seed. A private key using this technique can be 132 bytes since it should also store a 32 byte seed for randomised message hashing, the 64 byte public key and a 4 byte current leaf index.

For XMSS$^{\text{MT}}$, the parameter set XMSSMT_SHA2-256_W16_H20_D2 can generate 2^{20} signatures from a single private key but with other parameter sets where $H = 60$ it is possible to generate 2^{60} signatures [33]. The signature size is 4964 bytes and public keys are 64 bytes. Similarly to XMSS, a compact private key can be constructed by generating W-OTS keys for all XMSS trees pseudorandomly, giving a private key of size 132 bytes when using a 4-byte current leaf index.

An implementation of XMSS was available in the Botan C++ Cryptography Library [36]. A reference implementation of both XMSS and XMSS$^{\text{MT}}$ could also be found at one of the author's website: `https://huelsing.wordpress.com/code/`. The reference implementation was found to be several times faster due to various speedup techniques being applied.

3.3.2 SPHINCS

SPHINCS [32] is a hash-based signature scheme based on the Merkle tree approach. Just like XMSS, the security of SPHINCS relies solely on the properties of a cryptographic hash function. The SPHINCS hash tree is, similar to XMSS$^{\text{MT}}$, a hypertree consisting of several layers of hash trees. The leaf nodes of top layers in the hypertree contain Winternitz One-Time Signature (W-OTS$^+$) keys that are used for signing the root nodes of trees on a lower level. The Few-Time Signature (FTS) scheme HORS with Trees (HORST) is used for signing messages and the HORST keys are contained in the leaf nodes of the lowermost layer of trees.

SPHINCS is, compared to XMSS and many other hash-based signature schemes, a completely stateless digital signature scheme. Other hash-based signature schemes based on Merkle trees store a leaf index counter as part of the private key. This counter is updated for every new signature generated, to prevent reuse of the same key pair. The statelessness of SPHINCS is accomplished by instead picking a leaf index corresponding to a key pair randomly, without any regard to if the key pair has been used before. This randomised leaf index selection naturally opens up the risk of reusing the same key pair several times. If a One-Time Signature scheme was used this would lead to a complete compromise of the system. However, since SPHINCS uses a FTS scheme, this threat is minimised.

A Few-Time Signature scheme such as HORST is similar to a OTS scheme but can be used to sign a few messages without compromising the private key. However, with each message signed the probability of a forgery being possible increases. The parameters for SPHINCS-256 are chosen in such a way that the probability of a forgery should always be sufficiently small to ensure 128-bit security against quantum attackers. An example shows that even if 2^{50} messages are signed, which would take more than 30 years if a million messages are signed per second, the probability of a post-quantum attack with cost smaller than 2^{128} should be below 2^{-48} [32]. However, security degrades as the number of signatures increases which consequently means that the number of signatures that can be generated is limited.

Public and private keys A SPHINCS key pair is generated by first sampling two secret values $(SK_1, SK_2) \in \{0,1\}^n \times \{0,1\}^n$ [32]. SK_1 is used for pseudorandom key generation and SK_2 is used to generate an unpredictable index and to randomise the message hash. Also, a small number of bitmasks Q are generated. The bitmasks are used for all W-OTS$^+$ and HORST instances as well as for the trees.

To generate the public key, the root node must be generated. To do this, the W-OTS$^+$ key pairs for the topmost tree are generated. All the keys are generated pseudorandomly from SK_1. The leafs consisting of W-OTS$^+$ public keys are used to build the binary hash tree and calculate the root node value PK_1. The private key consists of (SK_1, SK_2, Q) and the public key consists of (PK_1, Q).

Signature generation Signatures on a message $M \in \{0,1\}^*$ are generated by first generating a pseudorandom value $R = (R_1, R_2)$ by feeding M and SK_2 into a pseudorandom function. A randomised message digest D is computed as the randomised hash of M using R_1 as randomness. To sign the message digest D, a HORST key pair must be chosen. The index used to choose a HORST key pair is computed using R_2. The index chooses both the tree and the leaf index inside the chosen tree.

A SPHINCS signature contains an index i, the randomness R_1 and a HORST signature σ. Furthermore, one W-OTS$^+$ signature and authentication path per layer of trees is required to verify a signature. These values are calculated during the signing process by generating one binary hash tree for each layer of the SPHINCS hypertree. Signing is deterministic since all required randomness is generated using a pseudorandom function.

Signature verification To verify a SPHINCS signature, the verifier must verify the HORST signature σ and one W-OTS$^+$ signature and authentication path per layer of trees. By doing this, the verifier can compute a value for the root node. The signature is valid if the computed value is equal to the value PK_1 of the public key.

Attacks, parameter sets and implementations The multi-target attack explained in Section 3.3.1 can be applied not only to XMSS but to SPHINCS as well. An estimated reduction in bit security is not specified and it is not trivial to calculate it given that SPHINCS is different to XMSS in many ways. Patching SPHINCS is however said to be possible by applying the changes presented in [35]. These changes would decrease the signature size of SPHINCS but most likely reduce the performance of the algorithm as well.

SPHINCS, and most other hash-based signature schemes, can be instantiated in a number of different ways with trade-offs between security level, signature size, and signature generation times [32]. The proposed instantiation of SPHINCS called SPHINCS-256 has 41000 byte signatures, 1056 byte public keys and 1088 byte private keys. SPHINCS-256 was claimed to have 256-bit security against classical computers and 128-bit security against quantum computers in the standard model. However, the multi-target attack presented in [35] lowers this security but the exact reduction in bit security is not clear since SPHINCS is different to the original XMSS in many ways. For example SPHINCS uses a FTS scheme and a 512-bit message digest. It is however clear that security degrades with the number of messages signed.

The original implementation of SPHINCS was written in C and could be found in the eBACS benchmark suite [37]. A Java version of SPHINCS had also been implemented in the Bouncy Castle Java Cryptography API [38].

3.4 Lattice-based signature schemes

To understand lattice-based cryptography it is important to first understand the basics of lattices. A lattice is a set of points in an n-dimensional space with a periodic structure [23]. The lattice is generated by n linearly independent vectors $b_1, ..., b_n \in \mathbb{R}^n$. These vectors are known as the basis of the lattice and it is straightforward to see that several different bases can be used to produce the same lattice. For basic lattice-based digital signature schemes, short and fairly orthogonal vectors are usually denoted as "good" bases and act as private keys, while long and far from orthogonal vectors are denoted as "bad" bases and act as public keys. Good bases can then find solutions to presumed hard problems while bad bases can only verify that the solution is correct.

Lattice-based cryptosystems are based on the presumed worst-case hardness of lattice problems [23]. Some of these problems are the Shortest Vector Problem (SVP), Closest Vector Problem (CVP) and the Shortest Independent Vectors Problem (SIVP). More recently proposed mathematical problems used for constructing lattice-based cryptosystems, used in for example the BLISS [39] and ring-TESLA [40] algorithms, are the Short Integer Solution and Learning With Errors problems over rings, (R-SIS) and (Ring-LWE) respectively.

Attempts to solve lattice problems using quantum algorithms have been made since the discovery of Shor's algorithm but so far there have been no major breakthroughs and lattice-based cryptosystems are still considered safe from quantum attacks [23, 3]. This means that there are currently no known quantum algorithms for solving lattice problem that perform significantly better than classical algorithms. It should however be noted that lattice-based cryptosystems have not been studied as much as for example RSA, and their security is uncertain even on classical computers.

In the remainder of this section the BLISS signature schemes will be explained in more detail. BLISS was chosen for its high efficiency and small keys. BLISS has also been called a bridge between theoretical and practical lattice-based schemes and for being a good candidate for integration into constrained systems and devices [41].

3.4.1 BLISS

The BLISS signature scheme is a recently proposed lattice-based scheme and its security relies on the presumed hardness of the generalised Short Integer Solution (SIS) problem. The BLISS scheme builds upon the failures of several previous lattice-based signature schemes such as NTRUSign and GGU that were broken by Nguyen and Regev [42] due to information about the private key leaking for each signature made [39]. BLISS uses rejection sampling, a method to sample from an arbitrary target probability distribution, with Bimodal Gaussian distributions to try to better hide the structure of the private key. Compared to previous lattice-based algorithms using rejection sampling, the method used by BLISS has a smaller number of average rejections in the rejection sampling step, which accelerates the total running time of the signing algorithm.

Public and private keys A BLISS private key is a matrix S in a ring R and the public key is a matrix A such that:

$$AS = A(-S) = qI_n \pmod{2q},$$

where I_n is the identity matrix of dimension n, the value of q is prime and $q = 1 \pmod{2n}$ [39, 43].

Signature generation A signature for message digest μ is generated by sampling a vector y from a discrete Gaussian distribution and computing the hashed value:

$$c \leftarrow H(Ay \mod 2q, \mu)$$

A bit $b \in \{0, 1\}$ is then sampled and the potential output:

$$z \leftarrow y + (-1)^b Sc$$

is computed. The output z is distributed according to a bimodal discrete Gaussian distribution. Rejection sampling is then performed and if successful, a signature (z, c) is returned as output. If the rejection sampling step is unsuccessful, the algorithm is restarted with a new value y. The expected number of iterations is generally small, between 1.6 and 7.4 for the different parameter sets proposed in the original paper.

Signature verification To verify a signature, the verifier must ensure that z is no larger than should be expected from a discrete Gaussian variable with width parameter σ [43]. The max-norm of z must also be verified to be small. The signature is then verified by checking if:

$$c = H(Az + qc \mod 2q, \mu)$$

The verification is valid because:

$$Az + qc = A(y + (-1)^b Sc) + qc = Ay + ((-1)^b AS)c + qc = Ay + (qI_n)c + qc = Ay \mod 2q$$

Attacks, parameter sets and implementations There are several different parameter sets proposed for BLISS offering different target security levels as well as trade-offs between speed and size [39]. The four parameter sets proposed in the original paper are BLISS-I, II, III and IV with targeted 128, 128, 160 and 192 bits of classical security in the random oracle model respectively. A more efficient implementation, compatible with BLISS, was later proposed called BLISS-B [44]. An instantiation of the BLISS-B-IV parameter set produces 6.5 kb signatures, 7 kb public keys and 3 kb private keys.

The quantum security of BLISS and BLISS-B was not discussed in the original papers. A recently published paper by Saarinen [45] has shown that the quantum security level is roughly halved by using Grover's algorithm to find a hash collision. The author proposed a new, less efficient but more secure, version of BLISS called BLZZRD with a supposed 128-bit quantum security.

Furthermore, Staffas [43] proposed another version called REBLISS to try to fix the problems pointed out in [45] as well as other weaknesses in the original BLISS scheme. The result is a presumably more secure scheme with comparable performance in regards to speeds and sizes. However, the author does not recommend using REBLISS or any other BLISS variant until more research has been done in the field since the security is still uncertain.

Academic implementations of BLISS and BLISS-B are published and made available under an open source license [46]. The authors are however clear that the academic implementation should not be used. BLISS and BLISS-B have later also been implemented in the *libstrongswan* cryptographic library included in strongSwan [47], an open source IPsec-based VPN Solution. By using libstrongswan it is possible to generate public key pairs, create and verify signatures and generate X.509 certificates. An educational implementation of BLZZRD is available in a public Github repository [48] while there seems to be no publicly available implementation of REBLISS.

3.5 Multivariate-based signature schemes

Multivariate-based cryptosystems are based on multivariate polynomials over a finite field [23]. In multivariate-based cryptosystems, the main security assumption is backed by the NP-hardness of the problem to solve non-linear quadratic equations over a finite field. The problem can be described as:

> **MQ Problem:** "Solve the system $p_1(x) = p_2(x) = ... = p_m(x) = 0$, where each p_i is a quadratic in $x = (x_1, ..., x_n)$. All coefficients and variables are in $\mathbb{K} = \mathbb{F}_q$, the field with q elements." [23]

The MQ Problem is NP-hard, meaning that not even the signer would be able to solve it efficiently if the set of equations were truly random. The way multivariate-based signature schemes work is that the public key is created with a trapdoor function which acts as the private key. This consequently means that effective attacks against the scheme might exist if there is an attack against the trapdoor function.

Multivariate signature schemes usually have a very large public key but in turn, produce very small signatures [23]. The basic idea of multivariate-based signature schemes is to choose a multivariate system F of quadratic polynomials which can be easily inverted [49]. This F is commonly referred to as the *central map*. Two affine linear maps S and T are then chosen to hide the structure of the central map. The public key is the composed map $P = S \circ F \circ T$, which is difficult to invert. The private key is the three maps S, F and T which makes it possible to invert P.

There are a several multivariate-based signature schemes available today. The remainder of this section will explain the Rainbow and the HFE$^{\text{v-}}$ scheme Gui in more detail. Rainbow was chosen mainly for its availability in a cryptographic library while Gui was chosen for its supposedly higher security and efficiency. Other HFE$^{\text{v-}}$ schemes exist as well but the recently proposed Gui is able to offer performance on par with RSA and ECDSA [50].

3.5.1 Rainbow

The Rainbow signature scheme, first proposed in [51], is a multivariate signature scheme based on the principle of Oil and Vinegar variables. More specifically, Rainbow uses a multi-layer Oil-Vinegar system. The principle of Oil and Vinegar variables is one way to create easily invertible multivariate quadratic systems [49].

To understand Oil and Vinegar schemes, let o and v be two integers such that $n = o + v$ [49]. For the n variables $x_1, ..., x_n$ in an Unbalanced Oil and Vinegar (UOV) scheme, where $v > o$, the Vinegar variables are $x_1, ..., x_v$ and the Oil variables are $x_{v+1}, ..., x_n$.

The quadratic polynomials of an Oil and Vinegar scheme are generated in such a way that the central map F can be easily inverted for the signer by first choosing the values of the v Vinegar variables at random. This produces a system of o linear equations in the o variables $x_{v+1}, ..., x_n$, which can be solved by Gaussian Elimination. If the system has no solution, new values for the Vinegar variables are chosen and the process can be repeated until a solution is found.

Public and private keys A Rainbow private key consists of the central map F, consisting of the multi-layer Oil and Vinegar system, and the two randomly chosen, invertible affine linear maps L_1 and L_2 [51]. The public key is the field structure of K and the $n - v_1$ polynomial components of the composed map $\bar{F} = L_1 \circ F \circ L_2$.

Signature generation To sign a message of arbitrary a length, a message digest Y' can be calculated using a hash function. In order to sign the message digest:

$$Y' = (y'_1, ..., y'_{n-v_1}),$$

a solution to the following equation must be found:

$$L_1 \circ F \circ L_2(x_1, ..., x_n) = \bar{F}(x_1, ..., x_n) = Y'$$

In order to solve this equation, the maps L_1, L_2 and F must be inverted. The Rainbow signature scheme uses multiple layers of Oil and Vinegar constructions [51]. To invert the map F we start by choosing values for $x_1, ..., x_{v_1}$ randomly and plugging them into a first layer of equations. This produces a system of o_1 linear equations in the o_1 unknowns $x_{v_1+1}, ..., x_{v_2}$, which can be solved by Gaussian Elimination. The values obtained are then inserted into the next layer of equations and the process is repeated until a solution is found for all variables $x_1, ..., x_{v_1}, ..., x_{v_2}, ..., x_n$. If one of the linear systems does not have a solution, the whole process is repeated for new values of $x_1, ..., x_{v_1}$.

By applying the inverse maps of L_1, F and L_2 to Y' we obtain the signature:

$$X' = (x'_1, ..., x'_n)$$

Signature verification Verifying a signature is a simpler operation. The verifier only needs to calculate a message hash Y' and verify that $\bar{F}(X') = Y'$.

Attacks, parameter sets and implementations Several attacks against Rainbow and other multivariate-based signature schemes have been found since Rainbow was first proposed. In 2010, a large number of parameter sets thought to be secure against the known attacks were provided in [49]. The security levels of the proposed parameter sets were calculated using the Lenstra-Verheul equation format that defines a year at which the scheme provides adequate security. The security of the different parameter sets ranges from 2010 to 2050. A translation to bit security and equivalent RSA security is given in [52], where a parameter set aimed at offering adequate security in the year 2040 is said to be equivalent to a 101-bit symmetric key or a 3214-bit RSA key. That parameter set, over the field GF(256), has 122.6 kB public keys, 87.7 kB private keys and 592-bit signatures.

Rainbow over the field GF(256) is implemented in the Bouncy Castle Java API [38] with adjustable parameters.

3.5.2 HFE^{v-} scheme Gui

The HFE^{v-} multivariate-based signature scheme was introduced in 1995 and has remained unbroken since then [53]. HFE^{v-} schemes exploit the fact that there are general methods to solve *univariate* polynomial equations over finite fields $\mathbb{F}_q$ efficiently. In order to exploit this property, HFE^{v-} schemes makes the multivariate polynomials in the central map secretly related to a univariate polynomial. This makes it possible to easily invert the central map, as is required in multivariate-based signature schemes.

Public and private keys A private key in the HFE$^{\text{v-}}$ signature scheme consists of the central map F, which is related to a univariate polynomial, and two secret invertible affine maps S and T [50]. The public key is the composed map $P = S \circ F \circ T$.

Signature generation A signature z on message digest h is generated by finding a solution to the equation:

$$P(z) = h$$

This can be done since the inverse of S, F and T can all be computed by the signer. The affine maps are easily invertible and the relationship to a secret univariate polynomial of HFE$^{\text{v-}}$ polynomials allows the signer to invert the central map F.

Signature verification A signature z is verified by hashing the message to obtain h' and verifying that $P(z) = h' = h$ [50].

Attacks, parameter sets and implementations Attacks against HFE$^{\text{v-}}$ schemes exist but so far none have been able to completely break the security of the scheme. One of the more recent variants of HFE$^{\text{v-}}$ schemes is called Gui [50] and offers better performance and possibly better security compared to previous HFE$^{\text{v-}}$ schemes such as QUARTZ. This was achieved by analysing design criteria for HFE$^{\text{v-}}$ schemes and finding that its possible to use HFE polynomials of very low degree while supposedly maintaining a high level of security.

There are three different parameter sets proposed for the Gui scheme [50]. The most secure parameter set, Gui-127, has an estimated 120-bit classical security and the quantum security is said to be 120 bits for a long time despite Grover's algorithm. This is because an attack using Grover's algorithm would require a quantum computer with a million qubits in comparison to the one required to break RSA and ECC, which supposedly only needs a few thousand qubits. The authors also state that although a quantum computer with a million qubits seems improbable, the scheme can be further protected by doubling the number of polynomials and variables. This would however increase the public key size by a factor 8 and also decrease performance. If a quantum computer with a million qubits exists, the quantum security is reduced to 60 bits.

For the Gui-127 parameter set, public keys are 143 kB, private keys are 5.3 kB and signatures are 163 bits. No public implementations of Gui seemed to be available.

3.6 Code-based cryptography

Code-based cryptography relies on error-correcting codes and the NP-hardness of the general decoding problem [23]. Code-based cryptography has shown to be a good candidate for post-quantum encryption, with the most promising candidate being the McEliece cryptosystem with hidden Goppa codes [54] that has withstood serious attacks for several decades from classical computers and has no efficient quantum attack [23]. One apparent downside is the size of the public key that becomes roughly 1 MB when aiming for 128-bit quantum security [55].

The use of code-based cryptography for digital signature schemes has had less success. Schemes proposed have either been broken, as in [56, 57], or offer poor performance in comparison to other post-quantum schemes with their large public keys and highly inefficient signature generation [3]. The advantage of code-based signature schemes is, similarly to multivariate-based signature schemes, that they have small signatures and efficient verification. For this reason, code-based signature schemes continue being an interesting research topic.

3.7 Isogeny-based cryptography

Isogeny-based cryptography is based on supersingular elliptic curve isogenies. This should not be confused with the elliptic curve cryptography used in ECDSA, which instead relies on the elliptic curve discrete logarithm problem and thus is broken by Shor's algorithm.

Supersingular elliptic curve isogeny-based cryptography is a relatively new field of research and therefore there has not been any digital signature schemes available until recently. Two independent papers [58, 59] presenting isogeny-based digital signature schemes have recently been published but both schemes suffer from relatively poor performance compared to other post-quantum schemes. For reference, the scheme presented in [59] has 336 byte public keys, 48 byte private keys and produces 122,880 byte signatures.

Some might further argue that there is reason to be sceptical towards the security of isogeny-based schemes due to the fact that they are relatively new and therefore have not been under as much scrutiny as other cryptosystems. NIST has stated that there has not been enough analysis to have much confidence in their security [6]. Isogeny-based cryptography does however still continue being an interesting research topic for the future.

Chapter 4

Methodology

The methodology used to produce the results of this thesis was an extensive literature study combined with gathering empirical evidence through experimentation. The literature study results and the empirical evidence was used to determine the algorithms practical usability in PKI today by inductive reasoning. By studying the signature sizes, key sizes, security and performance of the algorithms when generating signatures or X.509 certificates, a general conclusion on their suitability could be made. The following sections describe the literature study and the algorithm evaluation in more detail.

4.1 Literature study

The literature study was carried out by first finding relevant sources to gain sufficient background knowledge about PKI, quantum computing and post-quantum cryptography in general. This was followed by finding some general requirements for digital signature algorithms in PKI in order to rank different properties of digital signature algorithms. Both current requirements and possible future requirements were considered.

For the background knowledge on PKI and post-quantum cryptography, the literature consisted of mostly books published in the last ten years. This was to ensure that enough in-depth information was available, as compared to published articles where the given background is generally brief. The reason behind focusing on recently published books was to ensure that the information was not outdated due to fast development in the field of quantum computing and post-quantum cryptography.

Finally, post-quantum algorithms for digital signatures were identified and examined while staying critical to their proposed security levels. The main source of information was the original papers published on the algorithms as well as other published articles touching the same algorithm. Other articles could for example be practical implementation attempts, performance measurements, general discussions and cryptanalyses.

In cryptography it is important to look at the latest available information in order to not miss an important breakthrough such as a new attack against a cryptosystem. Therefore, most articles studied were from the last few years. However, it can also be important to not trust a newly proposed algorithms security measurements before it has had thorough analysis and peer-review. This meant that algorithms proposed in the last year were treated with more scepticism compared to older algorithms.

4.1.1 Literature sources and scepticism

Books, papers, articles and other sources of information were mainly gathered from three sources: The Primo search tool at KTH, that is connected to a large number of literature databases, Springer and Google Scholar. The websites of different cryptography conferences, such as the PQCrypto conference specialising in post-quantum cryptography, were also used to find the titles of newly published papers. From these sources it was deemed possible to gain access to a vast number of high-quality, peer-reviewed literature.

However, many articles regarding post-quantum cryptography were published quite recently. This could mean that, even though the articles had been peer-reviewed, they had not been subjected to as much scrutiny from the scientific community as older articles might have been.

Furthermore, since post-quantum cryptography is a relatively narrow field of research, many articles cited are published by the same universities, by the same project groups or by the same authors. This should not pose a problem as long as the articles hold a high scientific standard but it is still possible that some bias might have been present in those articles.

4.2 Algorithm evaluation

The evaluation of algorithms suitability for use in PKI was carried out in two steps. The first step was to study the results of the literature study to see which algorithms offered sufficient theoretical performance. This involved identifying key requirements for PKI and examining the algorithms in regards to those requirements. The key requirements for PKI were divided into performance requirements (running times) and size requirements (size of signatures and keys). The post-quantum algorithms were evaluated against the size requirements following the literature study. Important parameters that were considered for each algorithm were:

- Classical and quantum security level
- Signature size
- Public key size
- Private key size
- Code availability

Other interesting aspects considered were:

- Stateful/Stateless
- Limits on maximum number of signatures
- Similarity to currently used signature schemes

A signature algorithm with a proposed security level of at least 128 bits against both quantum attacks and classical attacks was seen as highly secure. Algorithms with less security were however also considered and compared since there can still be applications for less secure signature algorithms (short-lived certificates, constrained devices et cetera). Furthermore, the speed at which the performance of quantum computers will increase is uncertain, making a too strict security requirement counter-productive for a surveying study such as this thesis.

The second step was to gather empirical evidence by benchmarking some of the signature algorithms chosen from the first step. The goal of the thesis was to find algorithms that could be used in PKIs today. Therefore, only algorithms available in cryptographic libraries were benchmarked. The algorithms were evaluated to acquire empirical evidence of the running times in a practical setting. The algorithms were evaluated with regards to the following three properties in the benchmark:

- Key generation time
- Signature generation time
- Signature verification time

The algorithms were benchmarked by generating key pairs, generating signature for short messages and verifying the signatures. This was done in different libraries and programming languages, depending on the code availability. The average running time, median running time and sample standard deviation were measured for all operations. Self-signed X.509 certificates were also generated when the library allowed it, but since this was not possible for all algorithms this operation was not benchmarked in more depth than to confirm that X.509 certificate operations was about as fast as operations on arbitrary data.

To still have a fair time measurements between the different libraries and programming languages, the performance of an algorithm was evaluated against two common classical algorithms, 3072-bit RSA and ECDSA (P-256), which both offer 128-bit classical security. This evaluation could then produce a relative performance measurement to the classical algorithms, somewhat independent of programming language or library used. It is clear that this relative performance depends on how efficient the implementation of RSA and ECDSA was and how large the library overhead was. Therefore the relative performance evaluation could only give a rough estimate of how the algorithms might perform if they were implemented in the same library. The benchmark should therefore foremost be seen as a comparison between different practical implementations of algorithms and not as a measurement of the optimal performance of the algorithms.

4.3 Considered methodologies

Another methodology considered was to perform a literature study similar to the one conducted in this study but then choose only one post-quantum digital signature algorithm that was considered most suitable for digital signing in PKI. The chosen post-quantum algorithm would then be implemented in one or several cryptographic libraries in order to allow signing and verification of X.509 certificates. The implementations in different libraries could then be benchmarked to see which library offered the best performance and make a recommendation to the PKI community based on the benchmark results, while at the same time helping the community by making a post-quantum algorithm available in several different libraries.

This approach was however not chosen in favour of the methodology used in this thesis for several reasons. The first reason was that several promising post-quantum algorithms already had available implementations in cryptographic libraries. Comparing the implementations of those available implementations was deemed to have a higher value compared to implementing a single algorithm. The second reason was that the PKI community would most likely be more willing to integrate post-quantum algorithms available in moderately well-known cryptographic libraries that possibly was already being used in some PKI products.

Chapter 5

Literature study results

This chapter will describe the results obtained from the literature study. More specifically, signature algorithm requirements gathered from the literature study on PKI are first presented. Then, from the large set of post-quantum digital signature algorithms available today, a subset of algorithms are compared in order to find suitable candidates for use in PKI.

5.1 Signature algorithm requirements for use in PKI

For a PKI to be useful in practice it needs to be both secure and efficient. Since generating and verifying signatures is a vital part of the PKI, the signature algorithm must meet several requirements. The signature scheme must be secure for at least as long as the certificates they belong to are valid. For short-lived certificates the security could potentially be slightly lower compared to certificates with longer lifetimes, such as root CA certificates. Further requirements on the signature algorithm used in PKIs were divided into performance and storage requirements.

5.1.1 Performance requirements

The performance requirements of a signature algorithm can be broken down into key generation, signature generation and signature verification times.

Key generation of signature keys will either be performed by the end-entity or in rare cases by a CA. If the operation is performed by the end-entity, either in software or by hardware, the tolerable key generation time might be slightly higher. This is because the workload will be divided to all end-entities in contrast to the case of key generation on CAs where the CA need to generate keys for all end-entities. Furthermore, signature key generation is usually performed infrequently so the requirement of having short key generation times is not crucial as long as the key generation time is fast enough to not impact the end-entity's workflow severely.

Key generation on constrained hardware devices such as smart-cards might be problematic if key generation requires heavy computations. Generating keys on a more powerful device such as a server, PC or HSM (Hardware Security Module) and then transferring the keys to the constrained device is one way to solve the problem. This does however open up for security threats linked to the key transfer and the security of the machine generating the keys. Efficient key generation can therefore be seen as desired, but not necessarily crucial.

CAs need to sign certificates and CAs, RAs and end-entities might all need to sign and verify messages sent between them. It is therefore important that the signature generation and verification times are as small as possible. For example, CRLs and OCSP responses need to be signed by the CA and verified by whoever is trying to verify a certificate. Verification is performed more often than signing since certificates are only signed once but verified multiple times. Furthermore, verification is usually performed by end-entities who might have limited resources, while signing is usually done on the server side. Therefore, fast verification is deemed to be seen slightly more important than fast signing.

TLS, certificate-based logins and e-mailing using S/MIME and are just some applications that require signature generation and verification to be fast. The exact speed depends on the application and the hardware used to perform signature operations, but generally speaking the delay introduced by signature operations should not be noticeable to the end-user. Time-Stamping Authorities and other signing services will also need to provide signatures in a timely manner. Such services can also have regulations or customer demands when it comes to maximum signature times, usually less than one second.

The most restrictive performance requirement on a post-quantum signature algorithm might be to say that signature generation and verification should be at least as fast as classical signature algorithms. This might however be a bit too restrictive today since post-quantum cryptography is still a relatively new research area. A better requirement might be to say that the performance should be comparable to classical signature algorithms.

The performance requirements on a signature algorithm for key generation, signature generation and signature verification speeds will depend on the context in which it will be used. For end-entities, fast verification might be more important. For servers in the PKI, fast signature generation might be more important. In the general case for PKI the following ranking, from most important to least important, was deemed suitable due to the previously mentioned points in this section:

I. Fast verification of signatures

II. Fast signature generation

III. Fast key generation

This ranking coincides with the properties of RSA signing, where verification is faster than signing. This means that any post-quantum algorithm following the ranking will have similar properties to RSA. This was deemed to be a positive feature since RSA signing is commonly used for signing in PKIs today.

5.1.2 Size requirements

The storage needed by a PKI and its users will depend on the size of signatures, public keys and private keys. For X.509 certificates it is straightforward to see that the fields that change in size when switching algorithms are the signatureValue field and the subjectPublicKeyInfo field. The signatureValue field will change in size depending on the issuer signature algorithm and the subjectPublicKeyInfo field will change in size depending on the subject public key.

The size of a typical self-signed X.509 certificate excluding signature and subject public key was measured in order to fully understand how much impact a large signature size or a large subject public key size has on the total certificate size. The size was estimated to be roughly 300-400 bytes for a certificate encoded with DER. This size of course depends on the number of fields present, length of field inputs and the size of extensions, among other things.

Many protocols require signatures and certificates to be sent with requests or stored together with a file. For example, TLS requires certificates to be sent during the TLS handshake. If the signature of the certificate is much larger than the subject public key, a noticeable communication overhead will occur. To minimise this overhead, the issuing CA should use a signature algorithm that generates as small signatures as possible. If the CA is a root CA, a larger public key size can be tolerated due to the fact that root certificates are often distributed only once and together with an operating system, browser or other applications. However, if the CA is not a root certificate or some other certificate already known by the verifier, a certificate chain that includes the issuing CA's certificate must be transferred. This means that signature public key sizes also affect the size of transferred data.

For other protocols such as Long-Term Validation, the certificate might need to be stored together with a document. In this case, the total size of the signer's certificate will have an impact on the storage required to archive the signed document.

When downloading signed software or signed software updates, the size of signatures will determine the total download size. One interesting example is the software distribution in the Debian operating system where the median package size was 0.08 MB in 2014 [32]. A signature algorithm that generate large signatures will have a great impact if updates are small and frequently installed. In cases like this, having a signature algorithm with small signatures will be much preferred since the certificate used to verify the signatures is rarely updated.

The private key size is important when it must be stored on constrained devices such as smart-cards. In the general case however, having a small private key size is of low priority since most entities have enough storage available to store it.

The size requirements on a post-quantum signature algorithm depends on the application but for many applications, both signature and public key sizes are important. In PKIs, the following ranking, from most important to least important, was deemed suitable due to the previously mentioned points in this section.

I. Small signature size

II. Small public key size

III. Small private key size

5.2 Algorithm properties

The properties of the different post-quantum signature algorithms researched during the literature study are presented in Table 5.1. The algorithms are compared with regards to

Signature algorithm	Security level classic/quantum	Signature size (bytes)	Public key size (bytes)	Private key size (bytes)	Implementation
XMSS (SHA2-256_W16_H10)	256/128	2500	64	132	Botan
XMSS^{MT} (SHA2-256_W16_H20_D2)	256/128	4964	64	132	`https://huelsing.wordpress.com/code/`
SPHINCS-256	<256/<128	41,000	1056	1088	Bouncy Castle
BLISS-B-IV	159/96	832	896	384	strongSwan
REBLISS-I	128/128	809	870	166	-
Rainbow(256,31,21,22)	101/101	74	122,600	87,700	Bouncy Castle
Gui-127	120/120(60)	21	142,576	5350	-
Isogeny-based	192/128	122,880	336	48	`https://github.com/yhyoo93/isogenysignature`
RSA-3072	128/Broken	384	384	1728	Widespread
ECDSA (P-256)	128/Broken	64	64	96	Widespread

Table 5.1: Comparison of estimated security levels, signature and key sizes (in bytes), and available implementation of post-quantum and classical signature algorithms

security level, signature size, public key size, private key size and if they have an available implementation. All sizes in the table listed in bytes and might be slightly off due to rounding errors and misunderstandings between kilobits and kibibits. If an algorithm had several implementations available, the most high-grade implementation was listed. For educational and academic implementations, a link to the website where the code was available is listed.

The reference algorithms RSA-3072 and ECDSA over P-256, with 128-bit classical security but no quantum security, are also added to the table to show how the signature and key sizes of the post-quantum algorithms compare to a classical equivalent.

5.2.1 Security levels

The estimated security levels in Table 5.1 are expressed in bit security, meaning that for an attacker to attack a scheme with bit security b should require roughly 2^b operations to break the scheme. Both classical and quantum security levels are presented on the form:

"classical bit security"/"quantum bit security"

The security of SPHINCS-256 is reduced by the multi-target attack explained in [35] but the estimated reduction of bit security is not specified and is not entirely clear. Therefore, the bit security is specified as strictly less than the original security. It should be noted however that the attack does not completely break the signature scheme, in the same way that XMSS was not completely broken by the same attack. Furthermore, changing some parts of the SPHINCS algorithm would protect against the attack and at the same time decrease the signature size.

The security levels of BLISS-B-IV is reduced from 192/192 to 159/96 as shown in [43, 45]. Modifying the algorithm to obtain the REBLISS-I scheme would presumably increase quantum security and give slightly smaller keys and signatures.

No quantum security level was estimated for Rainbow(256,31,21,22) in [49, 52], but multivariate-based signature schemes are thought to be secure against quantum attackers since so far, no quantum algorithm exists to solve multivariate problems efficiently. Therefore, the quantum security level was set to the same value as the classical security level.

The quantum security of Gui depends on how a large-scale quantum computer is defined. In [50], the authors argue that the quantum security is 120 bits, the same as the classical security, as long as a quantum computer with a million qubits seems improbable. Otherwise, quantum bit security is reduced to 60 bits unless less efficient parameters are used.

5.2.2 Signature and key sizes

From the signature and key sizes presented in Table 5.1 it is clear that none of the post-quantum algorithms have both smaller signatures and smaller key sizes compared to RSA-3072 and especially not to ECDSA (P-256). There are however algorithms that offer either much smaller key sizes or smaller signatures. For example, the most storage efficient implementation of XMSS have keys that are smaller than RSA-3072 and on par with ECDSA (P-256). More performance efficient implementations of XMSS do however have larger private keys.

The multivariate-based signature schemes Rainbow(256,31,21,22) and HFE^{v-} scheme Gui-127 offer signatures significantly smaller than RSA-3072. Gui-127 signatures are even smaller than ECDSA-256, one of the most storage efficient signature algorithms used today. This makes Rainbow and Gui interesting signature algorithms when key sizes and security can be sacrificed to achieve small signature sizes. Compared to Rainbow, Gui offers both smaller signatures and smaller keys. Gui-127 private key sizes are significantly smaller than Rainbow, almost on par with RSA-3072 key sizes.

The lattice-based scheme BLISS-B-IV is the most balanced scheme with regards to signature and key sizes, with the only apparent downside being the questionable security.

5.2.3 Available implementations

From Table 5.1 it clear that the only algorithms with implementations in cryptographic libraries are XMSS, SPHINCS-256, BLISS and Rainbow.

XMSS has been available in the C++ cryptographic library *Botan* from version 1.11.34 where it is possible to both generate key pairs and sign/verify arbitrary messages. It is however not possible to generate X.509 certificates using XMSS.

SPHINCS-256 and Rainbow has both been available in the Java cryptographic library *Bouncy Castle* from version 1.55 and 1.48 respectively, where it is possible generate key pairs and sign/verify arbitrary messages. It is also possible to generate X.509 certificates for SPHINCS-256 and with minimal source code changes it is possible for Rainbow. SPHINCS-256 can be instantiated to use either SHA2 or SHA3. A Rainbow key pair can be instantiated with user-specified parameters over the field GF(256).

BLISS-B-I, III and IV has been available in the C cryptographic library *libstrongswan* included in strongSwan from version 5.3.0. It is possible to generate BLISS-B keys and generate/verify X.509 certificates using the command line pki tool. It is however not possible to sign or verify arbitrary messages using the pki tool. To do this, the *libstrongswan* library must be used directly.

The other algorithms presented in Table 5.1 either have no publicly available implementation or a basic educational implementation to display the performance and correctness of the algorithm.

5.2.4 Limitations of hash-based signatures

The hash-based signature schemes have two properties that are not relevant for other signature schemes. The first is that the hash-based signatures schemes have a limited number of signatures that can be generated. The second is that XMSS and XMSS^{MT} are stateful schemes, meaning that the private key is updated after every generated signature.

For XMSS and XMSS^{MT}, the maximum number of signatures that can be generated are determined during key generation. Using the parameter sets presented in [33], it is possible to generate between 2^{10} and 2^{20} signatures using XMSS and between 2^{20} and 2^{60} for XMSS^{MT}. Choosing a higher number affects running times and signature sizes negatively. For SPHINCS-256 the signature limitation is due to security being degraded for each generated signature. Using the same key pair for much more than 2^{50} signatures is not recommended since security is then severely degraded.

XMSS and XMSS^{MT} are both stateful schemes, making them unsuitable for systems that need to be backed up or where the signing should be distributed. SPHINCS is stateless but since security degrades with the number of signatures generated, distributed signing could still be problematic.

Chapter 6

Performance evaluation

In this chapter a performance evaluation is presented to show how the post-quantum algorithms performed in practice. The performance results of the post-quantum algorithms are compared against each other and against the two classical algorithms RSA-3072 and ECDSA (P-256).

The algorithms chosen for further evaluation following the results in Table 5.1 were XMSS, SPHINCS, BLISS-B and Rainbow. The reason for choosing these algorithms was that they had high-grade open source implementations in cryptographic libraries and relatively high security levels. The XMSS implementation offers the highest security, 256-bit classical and 128-bit quantum security. The SPHINCS, BLISS and Rainbow implementations offer slightly lower security. The implementation of Rainbow in Bouncy Castle does however allow for free choices of parameters so the security can be increased at the cost of decreased performance.

6.1 Benchmark descriptions

The benchmark for XMSS was written in C++ and utilised the Botan Cryptography API. The Botan API only allowed signing and verification of arbitrary messages when using XMSS. Time was measured using the *std::chrono::steady_clock* function included in the standard library.

The benchmark for SPHINCS-256 and Rainbow(256,31,21,22) was written in Java 8 utilising the Bouncy Castle Java Cryptography API via the JCA/JCE interface. Bouncy Castle allowed both signing and verification of arbitrary messages and certificate generation when using SPHINCS and Rainbow. Time was measured using the *System.nanoTime()* method, which offered nanosecond precision.

The benchmark for BLISS-B was written in C and utilised the libstrongswan cryptography library included in the strongSwan VPN Suite. The libstrongswan library allowed signing and verification of arbitrary messages while X.509 certificate were most easily generated and verified using the *strongSwan pki tool*, a command line interface tool. Time was measured using the *clock_gettime* function in the *time.h* library, which offered nanosecond precision.

All benchmarks were performed on a laptop with an Intel Core i7-3610QM CPU @ 2.30 GHz with Turbo Boost deactivated. The operating system used was Linux Mint 18.1 64-bit with Linux kernel 4.4.0-67-generic. Benchmarks were run with a minimal number of other processes running in the background to reduce system overhead. The average and median running times were calculated from a variable number of iterations depending on the algorithm tested. The general rule was that the benchmark should run for at least 250 iterations (for operations taking several seconds each) or at least tens of seconds (for operations taking time in the order of milliseconds or nanoseconds). Linux's non-blocking source of randomness */dev/urandom* was used in all benchmarks and cryptography APIs, in favour of blocking calls to */dev/random*, which was the default for some libraries.

All post-quantum algorithms were measured against RSA-3072 and ECDSA (P-256) implemented in the same library as the post-quantum algorithm. Due to the fact that Botan did not allow X.509 certificate operations for XMSS, all benchmarks measured signing and verification of arbitrary messages. The same wrapper classes or functions were used and the data signed was small and identical. In the strongSwan benchmark, *openssl* was utilised for RSA and ECDSA operations while the BLISS-B implementation was a native libstrongswan plugin. The average and median running times for key generation, signatures generation and signature verification were measured in all benchmarks. Additionally, the sample standard deviation was computed to show how consistent the running times of different operations were.

The compilers used were *g++/gcc* version 5.4.0 and *OpenJDK* version 1.8.0_121.

6.2 Benchmark results

In Table 6.1, Table 6.2 and Table 6.3, the benchmark results for the average running times are presented. The full benchmark data, including median time and sample standard deviations, can be found in Appendix A.

One thing to note about the results in Table 6.1 is that the XMSS algorithm implemented in Botan is optimized for storage by generating private keys from a seed to a pseudorandom function. As a consequence, the private key must be expanded whenever a message is signed which explains why signature generation is as costly as key generation.

The sample standard deviations were relatively small for all operations except for RSA key generation, Rainbow signing and BLISS signing. This is natural due to the mechanics of the respective algorithms. RSA key generation involves testing numbers to see if they are prime (with a high probability) until two prime numbers are found. This can take a variable amount of time which is shown by the large standard deviation. Signing in both Rainbow and BLISS has a chance to restart if an unsuitable random value is chosen at the start of the signing algorithm, which explains why the standard deviations were relatively high.

The Bouncy Castle benchmark had slightly higher standard deviations overall compared to the two other benchmarks. The exact reason for this was not investigated in too much detail, due to the small impact it seemed to have on the actual benchmark results, but it was possibly due to the unpredictiveness of the Java virtual machine. For example, the Java Just-In-Time Compiler and garbage collection occurring in the middle of a benchmarked operation could have made the results uneven.

The benchmark results show that BLISS-B-IV is the only algorithm with faster than or equal verification compared to RSA-3072 and ECDSA (P-256). It is also the only algorithm performing better than RSA-3072 for all operations.

Figure 6.1 summarises the results of Tables 6.1, 6.2 and 6.3 and shows how all the

Algorithm	Key generation	Sign	Verify
XMSS(SHA2-256_W16_H10)	4540 ms	4480 ms	2.69 ms
RSA-3072	4520 ms	12.5 ms	0.474 ms
ECDSA (P-256)	3.29 ms	18.2 ms	2.81 ms

Table 6.1: Average running times using the Botan Cryptography API

Algorithm	Key generation	Sign	Verify
SPHINCS-256	12.6 ms	236 ms	2.73 ms
Rainbow(256,31,21,22)	5770 ms	2.03 ms	1.85 ms
RSA-3072	2810 ms	25.1 ms	0.512 ms
ECDSA (P-256)	0.924 ms	0.553 ms	0.478 ms

Table 6.2: Average running times using the Bouncy Castle Cryptography API

Algorithm	Key generation	Sign	Verify
BLISS-B-IV	38.1 ms	1.27 ms	0.102 ms
RSA-3072 (openssl)	484 ms	4.84 ms	0.103 ms
ECDSA (P-256) (openssl)	0.101 ms	0.0941 ms	0.210 ms

Table 6.3: Average running times using the libstrongswan library (acting as openssl wrapper for RSA and ECDSA)

post-quantum algorithms performed in relation to each other. The fastest RSA and ECDSA times, obtained in the strongSwan benchmark, are also presented to show how the post-quantum algorithms perform in comparison. Note that the chart uses a logarithmic scale for its y-axis due to the large range of running times. The figure shows that BLISS-B-IV has the best performance out of the post-quantum algorithms for both signature generation and verification. Verification times for XMSS, SPHINCS and Rainbow are all relatively similar. The signing speed of Rainbow is almost on par with BLISS-B-IV, while XMSS and SPHINCS are both several times slower.

6.2.1 Relative performance

In Table 6.4 and Table 6.5, the relative performance speedup of the post-quantum signature algorithms are presented. The relative performance shows how the post-quantum digital signature algorithms performed in relation to RSA and ECDSA implemented in the same library and using the same wrappers and interfaces.

Table 6.4 shows that key generation speeds were generally higher for the post-quantum algorithms compared to RSA. The only exception is Rainbow due to the large number of computations needed to construct a key pair. Rainbow and BLISS-B-IV had significantly faster signing compared to RSA. The signing speed of SPHINCS-256 was roughly 8 times slower compared to RSA. The signing speed of XMSS was more than 300 times slower compared to RSA due to the costly key expansion mentioned earlier. Signature verification speeds were 4-5 times slower for all algorithms except for BLISS-B-IV where it was 1.35 times faster than RSA.

Table 6.5 shows that key generation speeds of the post-quantum algorithms were all far from the fast key generation speeds of ECDSA. Signing using the hash-based algo-

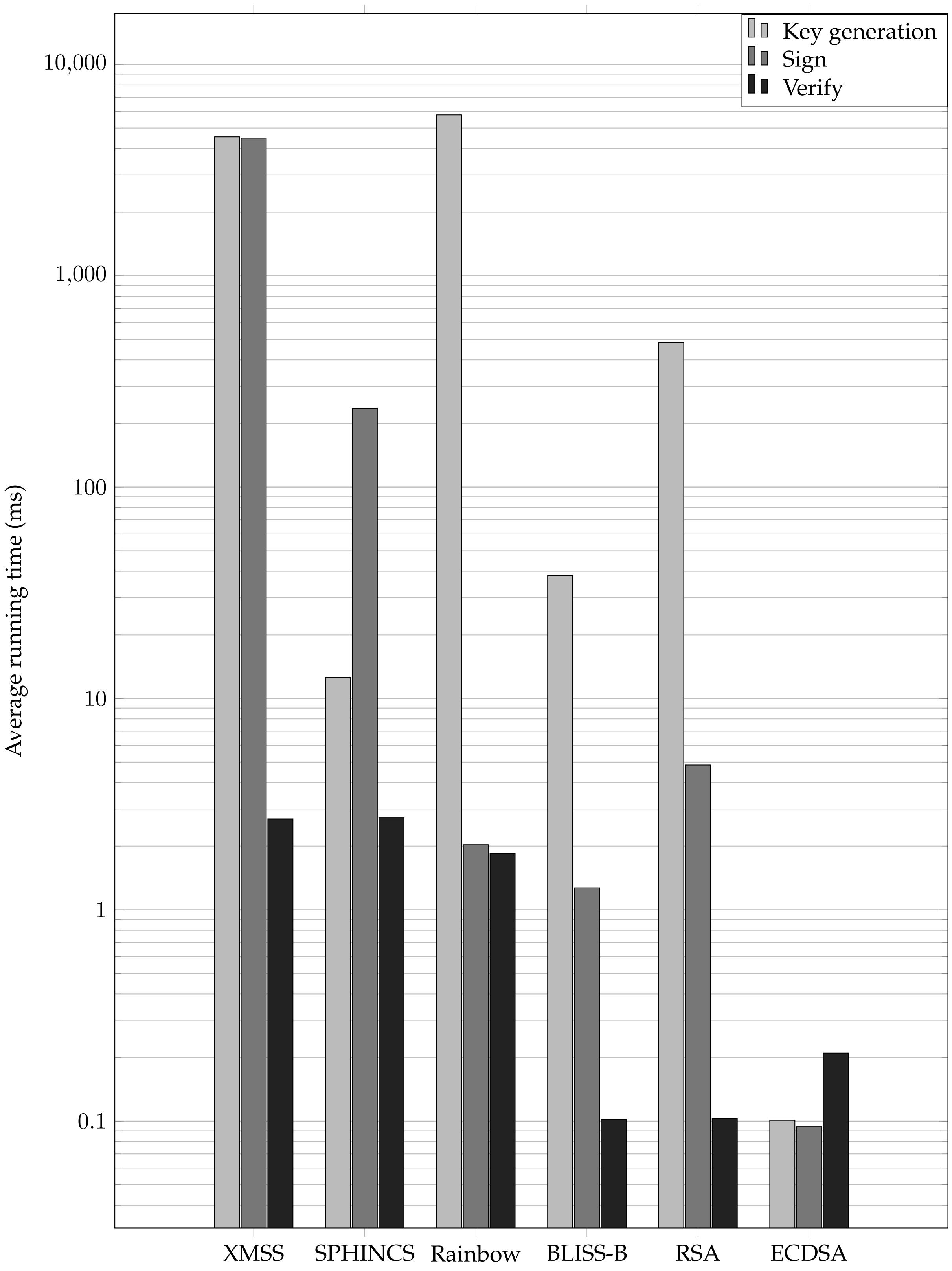

Figure 6.1: Comparison of average running times for all post-quantum algorithms and the fastest implementations of RSA and ECDSA (note that the y-axis is logarithmic)

Algorithm	Key generation	Sign	Verify
XMSS(SHA2-256_W16_H10)	1x	0.003x	0.2x
SPHINCS-256	200x	0.1x	0.2x
Rainbow(256,31,21,22)	0.5x	10x	0.3x
BLISS-B-IV	10x	4x	1x
RSA-3072	1x	1x	1x

Table 6.4: Average running time speedup compared to RSA-3072 for key generation, signature generation and signature verification

Algorithm	Key generation	Sign	Verify
XMSS(SHA2-256_W16_H10)	0.0007x	0.004x	1x
SPHINCS-256	0.07x	0.002x	0.2x
Rainbow(256,31,21,22)	0.0002x	0.3x	0.3x
BLISS-B-IV	0.003x	0.07x	2x
ECDSA (P-256)	1x	1x	1x

Table 6.5: Average running time speedup compared to ECDSA (P-256) for key generation, signature generation and signature verification

rithms was several hundreds times slower compared to ECDSA, while Rainbow and BLISS-B-IV were only 4 and 14 times slower respectively. Compared to ECDSA, signature verification speeds for BLISS-B-IV were roughly 2 times faster, XMSS was equally fast and the other post-quantum algorithms were between 4 and 5 times slower.

The post-quantum algorithm with the best relative performance for key generation was SPHINCS-256. Rainbow was best in regards to relative signature generation performance and BLISS-B-IV was best in regards to relative signature verification performance.

6.3 Possible error sources

It is clear that the benchmark was not ideal since it relies on measurements from libraries written in different programming languages and with different implementations. For example, the arithmetic library used for RSA operation can heavily influence the running times. In the strongSwan benchmark, where openssl is used for RSA operations, key generation and signing are several times faster compared to the RSA implementations in Botan and Bouncy Castle. Consequently, the relative performance speedup might become higher if the RSA implementation is slower, and vice versa. Therefore, the relative performance results are not conclusive.

Furthermore, the performance of the classical signature algorithms within the same libraries differed compared to each other. In general, ECDSA signing should be faster than RSA and ECDSA verification should be slower than RSA (as has been reported in for example [3] and measured in [37]). ECDSA performance in Botan is however significantly worse than in Bouncy Castle and strongSwan. In Botan, ECDSA signing and verification are both slower than RSA, while ECDSA signing is roughly 50 times faster than RSA in Bouncy Castle and strongSwan. Furthermore, ECDSA verification in Bouncy Castle is faster than RSA verification. These facts make the relative performance measurements somewhat misleading. Figure 6.1 helps negate this by showing the actual performance of the algorithms.

Implementation specific performance is relevant for the post-quantum algorithms as well. A good example of this is the signature generation speed measurement of XMSS. In the Botan cryptography API, XMSS private keys are minimised to cryptographically secure seeds that are used to seed pseudorandom functions to generate the full keys on demand. One major drawback with this approach is that in order to generate a signature, the entire tree must be generated again making signature generation as slow as key generation.

Another example is SPHINCS, which can be highly vectorised using the Intel AVX2 instruction set that allows for 8-way parallel computations on 32-bit integers [32]. In the eBACS benchmark suite, this fast implementation is roughly 4 times as fast as the reference implementation [37].

Furthermore, the benchmarking environment was not ideal since running background processes could have impacted the measurements. The running time of the benchmarks were however deemed long enough to ensure that any short peak in CPU usage from background processes should have had negligible impact.

The use of libraries written in different languages was also not ideal. This was noted by the relatively high sample standard deviation observed in the Bouncy Castle benchmark written in Java. A better benchmark result could have been obtained by writing implementations for the post-quantum algorithms for the same cryptographic library. That approach does however also have some pitfalls, namely that the performance can be influenced by the author.

Even though the benchmark presented in this thesis was not optimal, it still gives some valuable results and observations. The benchmark gives concrete measurements for how the post-quantum algorithms implemented in the different libraries performs in comparison to each other. The relative performance measurements can also be used as guidance for anyone making a choice to switch from a classical algorithm to a post-quantum algorithm in the same library.

6.4 Post-quantum X.509 certificates

In addition to the performance benchmarks where arbitrary messages were signed, self-signed X.509 certificates were generated for the post-quantum algorithms where the libraries allowed it. This was done as a proof-of-concept to show that post-quantum certificates could be generated and theoretically used in practice by PKIs today. Certificates were generated for SPHINCS-256, Rainbow(256,31,21,22) and BLISS-B-IV. Naturally, since none of the post-quantum algorithms were supported in more than one library, the generated post-quantum certificates could only be properly used by entities with access to the same cryptographic libraries.

The most time-consuming part of generating an X.509 certificate should be to generate the signature and the overhead from other operations should be negligible. This notion was confirmed for SPHINCS, BLISS and RSA by comparing signature generation and certificate generation times. However, generating and verifying certificates using the Rainbow algorithm in Bouncy Castle was several times slower compared to generating and verifying signatures for arbitrary strings. Some minor changes to the Bouncy Castle source code was required to allow X.509 certificate generation using Rainbow which indicated that full support for the algorithm was not present, which might explain the poor performance. Further reasons for the slowdown were not investigated.

Chapter 7

Discussion

In this chapter, the results from the literature study and the benchmarking are discussed in order to answer the research question of this thesis. The benchmark results form the basis for the recommendation on which post-quantum algorithm could be used today, with support from the literature study. Some more general discussions about the future of PKI and digital signatures are also backed by the results of the literature study. Some generalisations of the different categories of post-quantum algorithm are made in order to make better recommendations for the PKI community.

7.1 Discussion on post-quantum algorithms

It is clear from the literature study and the benchmarking that even though several post-quantum signature algorithms exist, they are all lacking in some aspects when compared to algorithms used in PKI today. Hash-based signature schemes seem to have the most robust security but they come at the price of having a limited number of signatures, large signature sizes and in the case of SPHINCS, degraded security for each new signature generated. Multivariate-based signature schemes provide efficient signing and very small signatures but the key sizes are very large. Lattice-based signature schemes seem to provide the overall most efficient schemes with relatively small key sizes, small signatures and efficient signature operations. However, the estimated security levels of multivariate-based and lattice-based schemes are not clear and have been reduced on a regular basis. The security also relies on mathematical problems that have not been studied for very long and could therefore be susceptible to attacks even from classical computers. Other post-quantum digital signature schemes exist but the performance of these schemes are so far even worse.

7.1.1 Hash-based algorithms

The verification speeds of XMSS and SPHINCS were good, when taking into account that the Botan and Bouncy Castle libraries seemed to be slower in general compared to strongSwan. The relative performance compared to RSA-3072 was about 5 times as slow for both algorithms. For many applications, such a slowdown could most likely be tolerated if it means that robust quantum security is achieved. In the future, more efficient hash functions could also be developed, which could speed up the performance of all hash-based signature schemes. Furthermore, if a hash function used in a hash-based signature scheme is shown to have a vulnerability it is easy to replace it.

The signing speed of the XMSS implementation in the benchmarked version of Botan was however very slow. This slow signing could be tolerated for signers where signing is not time critical, such as signing of legal documents or code signing, For PKIs however, a 4 second signing operation on for example OCSP responses would be a considerable overhead. Making the Botan implementation as efficient as the optimised reference implementation found at `https://huelsing.wordpress.com/code/`, that signs messages in about 20 ms, would make it more usable. The fact that an Internet Draft exists for XMSS also make paves the way for standardisation in a foreseeable future.

The large signature sizes of hash-based signature schemes, especially XMSS^{MT} and SPHINCS, could be a problem for applications where signatures are generated often and for relatively small messages. In the case of X.509 certificates, a 41 kB SPHINCS signature is roughly 100 times as large as the combined size of other fields in the certificate, excluding the public key. If a large public key size is included in the certificate, for example a McEliece encryption key or Rainbow key, the signature size will have less importance. Public keys for hash-based or lattice-based signature scheme have however been shown to be small in general.

The property of having a limit on the number of maximum number of signatures that can be generated from a singe key pair is another negative aspect of hash-based signature schemes, including the stateless SPHINCS. Being able to generate a limited number of signatures will most likely not pose a problem for most CAs and PKI users since the total number of signatures generated is generally low. If a CA needs to issue and sign a large number of certificates, it is more likely to distribute the load on intermediate CAs, who will use separate key pairs.

The degradation of security for each signature generated might however still pose a problem when it comes to standardisation. For example, NIST's requirements for post-quantum digital signature schemes to be standardised include the following criteria:

"For the purpose of estimating security strengths, it may be assumed that the attacker has access to signatures for no more than 2^{64} chosen messages; however, attacks involving more messages may also be considered." [7]

For SPHINCS-256, the security would be severely degraded by attackers of the above type. One way to protect against too many signatures being generated is to make SPHINCS stateful in the sense that a counter counts the number of generated signatures. The counter would not need to be as exact as the index in XMSS and the private key can be backed up or used for distributed signing. The difficulty lies in synchronising all signers with the counter. A simple way would be to make an interface for reading and updating the counter available online to all signers who could synchronise with it daily or maybe even more seldom. The computationally heavy signing operation does however help by making it difficult to reach a high number of generated signatures too quickly.

The overall security of hash-based signature schemes does however feel relatively well understood compared to other possibly post-quantum secure schemes. The fact that the security of hash-based signature schemes relies solely on the properties of cryptographic hash functions is a strong argument for using them in favour of schemes relying on less understood mathematical problems. If an attack against a specific hash function used in the signature scheme is found, then it is simple to replace it. For other types of schemes, such as lattice-based and multivariate-based schemes, an attack against the underlying mathematical problem would most likely lead to a complete breakage of the scheme. Furthermore, hash functions are necessary for other types of signature schemes as well meaning that if an efficient attack exists against hash-based signature schemes then the same attack could possibly be used against other schemes relying on hash functions as well.

7.1.2 Lattice-based algorithms

In the performance benchmark, BLISS-B-IV outperformed the other post-quantum signature algorithms and was even faster than RSA for all operations. The signature and key sizes are also on levels comparable to RSA. This would make BLISS a suitable successor if only the security was as clear as for hash-based signature schemes. Unfortunately, the security remains an open question due to the recent cryptanalyses of BLISS that found several problems with the BLISS scheme. Even though the flaws can be corrected, the confidence in the security of the scheme is somewhat damaged.

Lattice-based cryptography is interesting due to efficient algorithms such as BLISS but it seems that more research needs to be carried out in the field to increase confidence in the security of lattice-based schemes.

7.1.3 Multivariate-based algorithms

The performance benchmark showed that Rainbow had very good signing performance and verification speeds were also relatively fast. The only downside was the long key generation times, which is not a big problem for most applications, and lower estimated security compared to other post-quantum schemes. However, other studies have claimed that more efficient parameter sets than the one used in the benchmarks achieves an estimated quantum security of 128 bits, for example [3]. It is unclear how they have come to that conclusions and if it is correct, but it might indicate that the parameter set used in the benchmark could be more secure than previously stated.

The characteristics of multivariate-based schemes, with very small signature but large key sizes, offer an interesting alternative to other post-quantum algorithms. In applications where key sizes are of little importance, multivariate-based signature algorithms such as Gui, with 21 byte signatures, can even be more efficient than ECDSA (P-256) that has 64 byte signatures. The security of multivariate-based schemes is however still uncertain.

7.2 Post-quantum algorithms for certificate signing

The process to use an arbitrary post-quantum algorithm for certificate signing is relatively simple. First, an algorithm identifier for the signature and the subject public key needs to be agreed upon by all parties planning on using the post-quantum algorithm. The algorithm identifier consists of an object identifier (OID) and any optional parameters. Secondly, the format of the signature and the subject public key must be agreed upon. Lastly, all parties must be able to generate and verify signatures using the post-quantum algorithm. If all these steps are achieved, the post-quantum algorithm can be fully used in the X.509 certificate standard. The biggest factor in achieving the steps seems to be standardisation. Once standardisation institutes decide on algorithms to standardise, cryptographic libraries will most likely be quick to implement the algorithms in the library.

The flexibility of the X.509 standard allows for a smooth transition to post-quantum certificates. Furthermore, post-quantum signature algorithms available today operate in a similar way to their classical counterparts, that is the signer has a private key for generating signatures and the verifier uses the signer's public key to verify the signature. The exception is XMSS and other stateful hash-based signature schemes since the statefulness can be problematic in many ways.

For an organisation to make the transition from classical algorithms to XMSS, they would not only need to update all systems in the PKI to handle the new algorithm but they would also need to make major changes to how private signature keys are handled. This includes setting up rigorous rules for how backup and distribution of private keys can be performed, if possible at all. Some ideas, like reserving partitions of the index space, have been presented in papers such as [60] and could make XMSS easier to use. However, companies looking to make a transition to post-quantum cryptography in the near future will most likely look for alternatives more similar to classical signature algorithms to avoid having to make large structural changes.

To this end, SPHINCS seems to be a more attractive candidate for a hash-based post-quantum signature schemes being adopted by companies and organisations. The fact that SPHINCS is available in a widely used cryptographic API makes it likely that some companies and organisations might start offering support for it soon. However, for widespread deployment the algorithm would need be standardised.

7.2.1 Recommendations for the PKI community

The need to use post-quantum algorithms for signing certificates in PKIs is perhaps not as crucial as finding post-quantum encryption and key-exchange schemes. This is because most certificates are only valid for a couple of years. However, some signatures and also some certificates, such as root CA certificates or TSA certificates, might need to be valid for 10 years or more. Therefore, some recommendations for the PKI community can be helpful to guide preparations. It should however be noted that with the current uncertainty of both classical security and quantum security for most post-quantum schemes, switching to a post-quantum digital signature algorithm should only be done when using classical algorithms is not a choice. For example, if it becomes known that a quantum computer actually capable of breaking RSA and ECDSA exists.

CAs aimed towards the general public, such as CAs issuing TLS certificates, can not use post-quantum algorithms today due to the fact that all entities using a PKI must be able to handle the signature algorithm used. PKI vendors could however start offering support for post-quantum algorithms for everyone wanting to use them in more closed settings where they know that all entities in the PKI will be able to handle the post-quantum algorithm.

Making a transition to using XMSS, SPHINCS or Rainbow can be relatively simple if the widely used Botan or Bouncy Castle cryptographic library is already used in the PKI. Switching to BLISS is however not as easy since the libstrongswan library, although available under the GNU General Public License 2.0, is most likely currently not used by many applications outside of the strongSwan VPN suite.

For the purpose of certificate signing, the most secure post-quantum alternative with regards to estimated bit security seems to be XMSS or XMSS^{MT}. However, if the statefulness of the schemes is not handled correctly, the security effectively becomes zero. XMSS can therefore not be recommended for use in PKIs today unless the risks of having a stateful signature scheme are understood and handled correctly. One way to handle the risks might be to keep the XMSS private key in a strictly confined HSM environment. The stateless scheme SPHINCS can however be recommended for all PKIs where the number of generated signatures from a single key pair is somewhat limited and where large signature sizes and relatively long signature generation times are not a problem. This is because the security of hash-based algorithms seems to be quite well-understood.

Rainbow and other multivariate-based signature schemes could potentially be used for signing root certificates or be used in other protocols that do not require the public key to be distributed often. In the case of root certificates, the certificate is usually only distributed once during initial set-up of a system. A certificate with a public key size of over 100 kB is in this case not a problem, compared to certificates that are transmitted often. However, since root certificates need to be secure for a long time and the security of Rainbow is somewhat uncertain it can not be recommended for general use in PKIs today.

The security of BLISS is also somewhat uncertain even though recent cryptanalyses has helped patching some problems with the scheme. However, if a lower and more uncertain security level can be tolerated in exchange for a more efficient scheme, BLISS is a promising alternative that can be recommended. It must however be noted that even the classical security of BLISS and other lattice-based systems is uncertain and it will most likely take time until the security estimates of lattice-based schemes are trusted.

7.3 Transitioning to a post-quantum PKI

The post-quantum signature algorithms available today could all theoretically be used for signing in PKIs. There is no need make any significant changes to existing infrastructures other than to transition to using new algorithms, except in the case of stateful hash-based signature schemes. To make the transition to a post-quantum PKI easier it might be good to start moving towards it as soon as possible by support post-quantum algorithms for everyone interested in using them. This should be easier than to start transitioning when large-scale quantum computers are already available.

Some of the biggest hindrances today seem to be the fast development of new post-quantum algorithms and lack of standardisation. With new, promising algorithms being presented every year, spending development time to test and implement the algorithms can seem wasteful considering the fact that quantum computers still do not exist.

One way to promote the transition towards a post-quantum PKI, without necessarily breaking current functionality, might be to start using multiple signing keys and certificates for some applications. For example, code and document signing allows multiple signers. This feature can be used to generate signatures using both a classical algorithm and a post-quantum algorithm. By doing this, the verifier could choose to either ignore the post-quantum signatures (thus not breaking nor requiring any extra functionality) or verify both the classical signature and the post-quantum signatures to have a stronger verification that the code or document is authentic. For legal documents for example, it could be a good idea to start signing documents with an additional post-quantum algorithm as soon as possible to avoid being in a position where the validity of the classical signatures, and thus the authenticity of the documents, is questioned. Blockchain techniques is another interesting option that could be considered.

One thing identified in this thesis is that post-quantum signature sizes are in many cases larger compared to classical signatures. This will become less of an issue as processing power and available storage keeps increasing in the future, but it could still be important to find ways to make a post-quantum PKI more efficient.

One idea to make communication in the post-quantum PKI more efficient is to start caching more certificates. This should not pose a problem as long as the revocation status of a certificate is still checked every time it is to be used for verification. This change would require changes in several different protocols. However, protocol changes will most likely be needed anyway due to restrictions on public key and signature sizes. Public key and signature sizes have often been presumed to be relatively small, as is the case with RSA and ECDSA. Post-quantum algorithms with larger sizes might therefore not be usable unless the protocol specifications are updated.

7.3.1 Ethical and environmental aspects

The development of a post-quantum PKI will ensure that people and businesses can communicate securely even if large-scale quantum computers are available. Large-scale quantum computer will most likely be available to governments before the general public has access to them, given the great amount of research and funds required to build one. This raises an ethical question since governments with access to a large-scale quantum computer would be able to effectively spy on criminals if they use classical cryptography.

The same type of espionage used on people with criminal intent could however of course be used on law-abiding citizens and companies as well. This would be a huge threat to peoples' rights to privacy and freedom of speech on the Internet. Even the United Nations has stated that encryption and anonymity deserve strong protection since it enables individuals to exercise their right to freedom of opinion and expression in the digital age [61]. The development of a post-quantum PKI can therefore be seen as a necessary step to protect the rights of people worldwide and to help promote free, democratic societies free from excessive surveillance.

The environmental impact of post-quantum PKIs in comparison to PKIs today, will most likely not be significantly higher. It is however clear that the energy consumption of devices in the PKI will increase if the computation times needed for the signature algorithm increases. In the case of PKI for the IoT, this increased power consumption could make a somewhat noticeable environmental impact if the number of IoT devices is very large. In order to limit the impact, efficient signature algorithms should be used. This goes hand in hand with scientific interests and company interests, which means that the industry will be adopting more efficient algorithms as soon as they are deemed secure enough to be used.

Chapter 8

Conclusions

Quantum computers pose a threat to cryptography as we know it and thus PKIs are threatened as well. Luckily, there has been much research on post-quantum cryptography in the last decade and several interesting alternatives to RSA and other classical digital signature schemes have been proposed.

The survey conducted in this thesis found several post-quantum digital signature algorithms that could potentially be used in PKIs today. Hash-based XMSS and SPHINCS, multivariate-based Rainbow and lattice-based BLISS-B were all available in cryptographic libraries. Furthermore, SPHINCS, Rainbow and BLISS-B already had support for X.509 certificate operations. The post-quantum algorithms did however suffer from either highly uncertain security, lack of efficiency or both. The benchmarks conducted in this study showed that BLISS-B had the most efficient implementation. Security-wise, the hash-based signature schemes are better alternatives since they seem to provide stronger security due to the fact that security only relies on the properties of cryptographic hash functions. The most practical hash-based scheme SPHINCS could be recommended for use but then there is a need to handle 41 kB signatures that take a relatively long time to generate.

In conclusion, this thesis has found that although several post-quantum algorithms exist and have working implementations, there are still some to obstacles to widespread deployment and use in PKI. The perhaps biggest obstacle is the lack of trust in the security of post-quantum algorithms. Much more research will need to be done before individuals and companies will have much trust in their security. Lack of standardisation is another obstacle that needs to be dealt with before post-quantum algorithms can start being widely used in the PKI industry and other fields. Standardisation institutes have started looking at post-quantum algorithm standardisation and they will hopefully find suitable candidate algorithms in the next couple of years.

8.1 Future work

Some interesting future work could be to implement interesting post-quantum algorithms without high-grade implementations in a cryptographic library to allow X.509 certificate operations. This could help bring attention to the threat of quantum computers to anyone using the cryptographic library and showing that practical post-quantum algorithms exist.

Another interesting aspect to study could be to measure and compare the amount of entropy needed for randomness during key generation, signing and verification for different post-quantum and classical signature schemes. This could help guide product development for environments where the entropy pool is small.

There is also still need for more research in the field of post-quantum cryptography. Mainly, further cryptanalysis on the security of existing post-quantum algorithms is needed and more efficient algorithms with robust security need to be developed.

Bibliography

[1] Bruce Schneier. *Applied cryptography*. John Wiley & Sons, 1996. ISBN 0-471-11709-9.

[2] Johannes A Buchmann, Evangelos Karatsiolis, and Alexander Wiesmaier. *Introduction to public key infrastructures*. Springer Science & Business Media, 2013.

[3] M Campagna, L Chen, Ö Dagdelen, J Ding, JK Fernick, N Gisin, D Hayford, T Jennewein, N Lütkenhaus, M Mosca, et al. Quantum safe cryptography and security. *ETSI White Paper*, 8, 2015.

[4] IBM. IBM Makes Quantum Computing Available on IBM Cloud to Accelerate Innovation. `https://www-03.ibm.com/press/us/en/pressrelease/49661.wss`, May 2016. Accessed: 2017-02-22.

[5] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[6] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. *National Institute of Standards and Technology Internal Report*, 8105, 2016.

[7] NIST. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf`, December 2016. Accessed: 2017-04-12.

[8] Matt Braithwaite. Experimenting with Post-Quantum Cryptography. `https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`, July 2016. Accessed: 2017-02-23.

[9] SHS Vries. Achieving 128-bit security against quantum attacks in OpenVPN. Master's thesis, University of Twente, 2016.

[10] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, RFC Editor, May 2008. URL `https://www.rfc-editor.org/info/rfc5280`.

[11] M. Nystrom and B. Kaliski. PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986, RFC Editor, November 2000. URL `https://www.rfc-editor.org/info/rfc2986`.

[12] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[13] K. Moriarty Ed., B. Kaliski, J. Jonsson, and A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, RFC Editor, November 2016. URL `https://www.rfc-editor.org/info/rfc8017`.

[14] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[15] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, RFC Editor, June 2013. URL `https://www.rfc-editor.org/info/rfc6960`.

[16] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008. URL `https://www.rfc-editor.org/info/rfc5246`.

[17] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751, RFC Editor, January 2010. URL `https://www.rfc-editor.org/info/rfc5751`.

[18] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling. RFC 5750, RFC Editor, January 2010. URL `https://www.rfc-editor.org/info/rfc5750`.

[19] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161, RFC Editor, August 2001. URL `https://www.rfc-editor.org/info/rfc3161`.

[20] TS ETSI. 102 023 V1. 2.2 (2008-10), Electronic Signatures and Infrastructures (ESI). *Policy requirements for time-stamping authorities*.

[21] CSS Technical Team. 2016 Public Key Infrastructure (PKI) and Internet of Things (IoT) Security Predictions. `https://blog.css-security.com/ctoblog/2016-public-key-infrastructure-pki-and-internet-of-things-iot-security-predictions`, December 2015. Accessed: 2017-05-16.

[22] Tao Zhang and Luca Delgrossi. *Vehicle Safety Communications Protocols, Security, and Privacy*. Information and Communication Technology Series. Wiley, Hoboken, 2012. ISBN 1-118-13272-6.

[23] Daniel J Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-quantum cryptography*. Springer Science & Business Media, 2009.

[24] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[25] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.

[26] Arjen K Lenstra. Key lengths. Technical report, Wiley, 2006.

[27] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 387–394. ACM, 1990.

[28] Denis Butin, Stefan-Lukas Gazdag, and Johannes Buchmann. Real-world post-quantum digital signatures. In *Cyber Security and Privacy Forum*, pages 41–52. Springer, 2015.

[29] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

[30] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[31] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[32] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. Sphincs: practical stateless hash-based signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 368–397. Springer, 2015.

[33] Denis Butin, Andreas Hülsing, Aziz Mohaisen, and Stefan-Lukas Gazdag. XMSS: Extended Hash-Based Signatures. Internet-Draft draft-irtf-cfrg-xmss-hash-based-signatures-09, Internet Engineering Task Force, March 2017. URL `https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-xmss-hash-based-signatures-09`. Work in Progress.

[34] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal parameters for xmss mt. In *International Conference on Availability, Reliability, and Security*, pages 194–208. Springer, 2013.

[35] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography–PKC 2016*, pages 387–416. Springer, 2016.

[36] Botan. Botan: Crypto and TLS for C++11. `https://botan.randombit.net/`, January 2017. Accessed: 2017-03-07.

[37] eBACS. eBACS: ECRYPT Benchmarking of Cryptographic Systems. `https://bench.cr.yp.to`, July 2016. Accessed: 2017-04-20.

[38] Bouncy Castle. The Legion of the Bouncy Castle Java Cryptography APIs. `https://www.bouncycastle.org/java.html`. Accessed: 2017-03-07.

[39] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013.

[40] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In *International Conference on Cryptology in Africa*, pages 44–60. Springer, 2016.

[41] James Howe, Thomas Pöppelmann, Máire O'Neill, Elizabeth O'Sullivan, and Tim Güneysu. Practical lattice-based digital signature schemes. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(3):1–24, May 2015. ISSN 1539-9087.

[42] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures.(report). *Journal of Cryptology*, 22(2), April 2009. ISSN 0933-2790.

[43] Rebecca Staffas. Post-quantum lattice-based cryptography. Master's thesis, Royal Institute of Technology, 2016.

[44] Léo Ducas. Accelerating bliss: the geometry of ternary polynomials. *IACR Cryptology ePrint Archive*, 2014:874, 2014.

[45] Markku-Juhani O Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, pages 1–14, 2017.

[46] BLISS: Bimodal Lattice Signature Schemes. `http://bliss.di.ens.fr/`. Accessed: 2017-03-14.

[47] strongSwan. Bimodal Lattice Signature Scheme (BLISS). `https://wiki.strongswan.org/projects/strongswan/wiki/BLISS`, 2016. Accessed: 2017-03-14.

[48] Saarinen, Markku-Juhani O. Reference implementation of the BLZZRD variant of the BLISS Ring-LWE Signature Scheme. `https://github.com/mjosaarinen/blzzrd`, 2016. Accessed: 2017-03-14.

[49] Albrecht Petzoldt, Stanislav Bulygin, and Johannes A Buchmann. Selecting parameters for the rainbow signature scheme-extended version-. *IACR Cryptology ePrint Archive*, 2010:437, 2010.

[50] Albrecht Petzoldt, Ming-Shing Chen, Bo-Yin Yang, Chengdong Tao, and Jintai Ding. *Design Principles for HFEv- Based Multivariate Signature Schemes*, pages 311–334. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-48797-6. doi: 10.1007/978-3-662-48797-6_14. URL `https://dx.doi.org/10.1007/978-3-662-48797-6_14`.

[51] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International Conference on Applied Cryptography and Network Security*, pages 164–175. Springer, 2005.

[52] Albrecht Petzoldt, Stanislav Bulygin, and Johannes Buchmann. *Selecting Parameters for the Rainbow Signature Scheme*, pages 218–240. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-12929-2. doi: 10.1007/978-3-642-12929-2_16. URL `https://dx.doi.org/10.1007/978-3-642-12929-2_16`.

[53] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography—dealing with the fallout of physics success. Cryptology ePrint Archive, Report 2017/314, 2017. `https://eprint.iacr.org/2017/314`.

[54] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244: 114–116, 1978.

[55] Daniel J Bernstein, Tung Chou, and Peter Schwabe. Mcbits: fast constant-time code-based cryptography. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 250–272. Springer, 2013.

[56] Aurélie Phesso and Jean-Pierre Tillich. An efficient attack on a code-based signature scheme. In *International Workshop on Post-Quantum Cryptography*, pages 86–103. Springer, 2016.

[57] Dustin Moody and Ray Perlner. Vulnerabilities of "mceliece in the world of escher". In *International Workshop on Post-Quantum Cryptography*, pages 104–117. Springer, 2016.

[58] Steven D. Galbraith, Christophe Petit, and Javier Silva. Signature schemes based on supersingular isogeny problems. Cryptology ePrint Archive, Report 2016/1154, 2016. `https://eprint.iacr.org/2016/1154`.

[59] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. Cryptology ePrint Archive, Report 2017/186, 2017. `https://eprint.iacr.org/2017/186`.

[60] David McGrew, Panos Kampanakis, Scott Fluhrer, Stefan-Lukas Gazdag, Denis Butin, and Johannes Buchmann. State management for hash-based signatures. In *Security Standardisation Research*, pages 244–260. Springer, 2016.

[61] David Kaye. Report on encryption, anonymity, and the human rights framework. *Report of the special rapporteur on the promotion and protection of the right to freedom of opinion and expression*, 22, 2015.

Appendix A

Full benchmark results

The full results of the benchmarks, including median running times and sample standard deviations, are presented in Tables A.1, A.2 and A.3. The average running times are highlighted for easier comparison between different algorithms.

	Key generation			Sign			Verify		
Algorithm	**Mean**	Median	Std Dev	**Mean**	Median	Std Dev	**Mean**	Median	Std Dev
XMSS(SHA2-256_W16_H10)	**4540 ms**	4540 ms	8.75 ms	**4480 ms**	4483 ms	8.66 ms	**2.69 ms**	2.68 ms	0.0841 ms
RSA-3072	**4520 ms**	4070 ms	2780 ms	**12.5 ms**	12.5 ms	0.177 ms	**0.474 ms**	0.473 ms	0.00214 ms
ECDSA (P-256)	**3.29 ms**	3.30 ms	0.0265 ms	**18.2 ms**	18.2 ms	0.174 ms	**2.81 ms**	2.81 ms	0.0204 ms

Table A.1: Average, median and sample standard deviation of running times using the Botan Cryptography API

	Key generation			Sign			Verify		
Algorithm	**Mean**	Median	Std Dev	**Mean**	Median	Std Dev	**Mean**	Median	Std Dev
SPHINCS-256	**12.6 ms**	11.9 ms	2.11 ms	**236 ms**	231 ms	11.9 ms	**2.73 ms**	2.64 ms	0.410 ms
Rainbow(256,31,21,22)	**5770 ms**	5750 ms	94.0 ms	**2.03 ms**	1.74 ms	0.909 ms	**1.85 ms**	1.85 ms	0.0550 ms
RSA-3072	**2810 ms**	2490 ms	1700 ms	**25.1 ms**	25.0 ms	0.948 ms	**0.512 ms**	0.499 ms	0.210 ms
ECDSA (P-256)	**0.924 ms**	0.687 ms	0.645 ms	**0.553 ms**	0.523 ms	0.144 ms	**0.478 ms**	0.458 ms	0.171 ms

Table A.2: Average, median and sample standard deviation of running times using the Bouncy Castle Cryptography API

	Key generation			Sign			Verify		
Algorithm	**Mean**	Median	Std Dev	**Mean**	Median	Std Dev	**Mean**	Median	Std Dev
BLISS-B-IV	**38.1 ms**	36.5 ms	7.70 ms	**1.27 ms**	0.811 ms	0.762 ms	**0.102 ms**	0.102 ms	0.00267 ms
RSA-3072 (openssl)	**484 ms**	396 ms	351 ms	**4.84 ms**	4.79 ms	0.227 ms	**0.103 ms**	0.103 ms	0.00128 ms
ECDSA (P-256) (openssl)	**0.101 ms**	0.100 ms	0.00853 ms	**0.0941 ms**	0.0937 ms	0.00164 ms	**0.210 ms**	0.209 ms	0.00598 ms

Table A.3: Average, median and sample standard deviation of running times using the libstrongswan library (using openssl for RSA and ECDSA)

www.kth.se