UiO : Department of Informatics
University of Oslo

Quantum Safe Cryptography Based on Hash Functions: A Survey

Mateusz Zych

Master's Thesis Autumn 2018

Abstract

The use of public key cryptosystems range from securely encrypting emails and files to creating digital signatures for non-repudiation. The security of public key cryptosystems is based on computationally "unsolvable" complex mathematical problems. This tends to change with the introduction of quantum computers that undoubtedly pose a threat to the current schemes. In response to that extensive research has been conducted that resulted in several cryptosystems that are believed to be quantum resistant. This thesis presents a concise overview of multiple hash-based signature schemes and provides a comparative description and analysis of them. The comparisons are based on different signature scheme properties, such as key sizes, signature sizes and security level provided. The proposed quantum resistant schemes are also compared to the standard schemes used today, such as RSA and ECDSA.

Acknowledgements

I would like to thank my supervisor, Leif Nilsen, for the help I got from him during the thesis. I have been fortunate to make many new friends during my time at the University of Oslo. There are too many to mention, you know who you are, and you people are great. Additionally, I would like to thank my family for believing in me during the tough time of writing this thesis. Finally, I would like to express a special appreciation to my Mom, who always encouraged me through the process and supported me with great conversations when there was nobody to respond to my queries.

Contents

List of Figures

List of Tables

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Motivation

The advancements in technology and particularly in digital communications is one of the technological foundations of the modern society. Confidentiality, integrity, authenticity, and non-repudiation in data transmission and data storage are properties that have made cryptography an essential discipline within information technology.

In recent years, quantum computing has become attractive as a research topic due to the acceleration of technology. This development has led to the increased exposure of quantum computers, which make use of quantum physical phenomena to perform calculations. Two types of quantum computers exist are universal and non-universal. Their difference is that universal quantum computers are developed to perform any given task, whereas non-universal quantum computers are limited to a specific purpose.

Quantum computers pose a significant risk to both present public key algorithms (such as RSA, Diffie-Hellman, and DSA) and symmetric key algorithms (like 3DES, AES). Each year it seems that we are getting closer to create a fully operational universal quantum computer that can utilize strong quantum algorithms, such as Shor's and Grover's. In conventional cryptography the public key algorithms are used for key exchange and digital signatures. The consequence of the technological advancement is the absolute collapse of the present public key algorithms that are considered secure (e.g, RSA and Elliptic Curve Cryptosystems). Consequently, all protocols using either key exchange or digital signatures will be broken resulting to insecure network communications.

In August 2016, the National Institute of Standards and Technology (NIST) published the following information [24]: *"The advent of practical quantum computing will break all commonly used public key cryptographic algorithms. In response, NIST is researching cryptographic algorithms for the public key - based key agreement and digital signatures that are not susceptible to cryptanalysis by quantum algorithms."*

The goal of post-quantum cryptography (also known as quantum-resistant cryptography) is to develop cryptographic systems that are secure against both conventional and quantum computers and can inter-operate with existing communication protocols [26]. Many post-quantum public key candidates have been actively investigated during the last few years that are believed to be resistant to quantum-based attacks. There are several ways to perform post-quantum secure cryptography, such as lattice-based, multivariate, hash-based, code-based, super-singular elliptic curve isogeny cryptography and symmetric key quantum resistance [23]. Not all of the aforementioned provide key encapsulation mechanisms and digital signatures. However, if a given algorithm covers at least one of these two properties, it would be valuable for post-quantum cryptography. This thesis will focus on hash-based digital signature schemes.

1.2 Description of the problem

Hash-based signatures have been investigated and researched since the late 70s. Information about hash-based signatures is scattered around the world. This makes it difficult to find a reasonable source with a good overview of this field. The last overview of this topic has been provided in 2009 by ETSI which covered only the basic of post-quantum cryptography, including hash-based signatures. There has been a lot of changes since ETSI's overview of the Post Quantum Cryptography (PQC) field. Over time, new algorithms have been introduced, and existing algorithms have been modified and improved.

This thesis will investigate and address the following question:

1. What kind of hash-based post-quantum digital signatures schemes do exist?

This thesis can also be considered an up-to-date concise "encyclopedia" of post-quantum hash-based algorithms for digital signatures.

1.3 Methodology

This thesis intends to describe hash-based signatures schemes and provide a comparative description and analysis of some of the proposed schemes. The description will be based on the aspect of the signature schemes such as key sizes, signature size and security level. The proposed schemes will also be compared to the standard schemes used today, such as RSA and DSA. Also, by describing and comparing hash-based signatures. The thesis aims to make hash-based signatures a bit more accessible, as a lot of the information on this topic is not aggregated and the history of hash-based signatures stretches back to 1978.

The work in this master thesis has also resulted in a publication in the International Journal of Advanced Computer Science and Applications (IJACSA) in March 2018. The paper entitled "The impact of quantum computing in present cryptography" is attached as an appendix at the end of this thesis.

1.4 Structure

The thesis will first, chapter 2, describe basic security concept, conventional cryptography and the usage of digital signatures in standardized protocols. Chapter 3 covers information about quantum computing and its impact on conventional cryptography. Chapter 4 will then present the first appearance of One-time hash-based signatures and its evolution. Chapter 5 describe some of the most important Few-time hash-based signatures. Chapter 6 describes stateful Many-Time hash-based signatures. Chapter 7 describes stateless Many-Times hash-based signatures which are proposed for the standardization process. All chapters 4, 5, 6 and 7 describe hash-based signatures by first presenting the protocols of key generation, signing and verifying process, then describes the security of the system. Chapter 8 discus the difference between stateful and stateless signature schemes. Chapter 9 concludes the thesis with a summary and possible ideas for future work in this field. At the end of this thesis, there is an appendix to find with the published paper derived from this work.

Chapter 2

Theoretical background

This chapter is meant to be a comprehensive collection of information which should provide the reader with the basic definitions needed to understand the second part of the thesis. Definitions which seem relevant to security, conventional cryptography, quantum computing, and protocols will be described in their entirety in this chapter.

2.1 Security Concepts

Security is a wide field of study with many concepts, principles, services, and mechanisms. However, the term itself has no single definition. In computer science, security strongly depends on security objectives and in which degree the security services with the help of security mechanism can fulfill them. For instance, a system might provide or need secure communication, which would be an objective. The latter is dependent on services like TLS that encrypts, which is a mechanism, the communication. How secure a system is, strongly depends on proper security requirements and the security measures that fulfill these under all proper circumstances.

The CIA (Confidentiality - Integrity - Availability) principle is often used when describing the system security. However, it does not cover the whole specter of security. Thus, more concepts like accountability, authentications and other are added to provide better security. Subsections below presents these security concepts.

2.1.1 Confidentiality

Confidentiality (privacy, secrecy) says that the information should be secure and only accessed by the authorized parties. Privacy refers to protection of personal data, whereas secrecy refers to protection of data belonging to an organization. Additionally, confidentiality possesses properties such as unlinkability and anonymity. Here, the former means that two or more items of interest cannot sufficiently be distinguished whether they are related or not. The latter term ensures that a subject cannot be identified within a set of subjects [46, 34-35].

2.1.2 Integrity

Integrity is to prevent unauthorized modifications or destruction of data. In other words, to secure that the data has not been tampered with. Two types of integrity exist data and system integrity. Data integrity refer to the integrity of all digital information. Whereas, system integrity refers to a state of a system where it is performing its intended functions without being degraded by disruptions in its environments [46].

2.1.3 Availability

Availability ensures that resources, services or data are accessible and usable upon demand by authorized entities. The importance of availability is illustrated by for instance in a DoS (Denial of Service) attack, which hinders legitimate users from using a system [90].

2.1.4 Accountability

Accountability ensures that the actions of an entity can be traced uniquely to that entity. In other words, all actions affecting security are traceable back to the responsible party. Accountability depends on identification, authentication, and auditing of a user [46].

2.1.5 Non-Repudiation

The security term non-repudiation is strongly related to accountability. Non-repudiation provide undeniable evidence that an action has occurred. This term is usually divided in non-repudiation of [46]:

- origin - an entity cannot deny to having send a message,
- delivery - an entity cannot deny to have received a message.

2.1.6 Identity and Access Management

Identity and access management (IAM) is a framework which denotes the security technologies that enables the intended authorized individuals to access the right resources at the right times for the right reasons. This framework consists of three phases: configuration-, operation- and termination- phase [61].

Configuration phase contains:

1. Registration: Creation of an identity and new account.
2. Provision: Issuing credentials and unique name.
3. Authorization: Granting of rights by the authority.

Operation phase contains:

1. Identification: Identification is an assertion of who or what something is. This information is used to recognize an individual user. In computer systems, where log in appears, the username corresponds to the identity, where it claims to be a known entity. D.Gollman in his book [46, 60], says that identification is a $1 : n$ comparison that tries to identify the user from a database of n person.
2. Authentication: The entity has to prove that the claims about the identity are correct by providing some credential to the system. The system has then to compare the credential with the information stored in the database for that specific entity. This process is called verification and is an $1 : 1$ comparison [46, 60]. The entity is authenticated only when the passwords for that entity matches.
3. Access control: Grating access by the system. Access control determines policies for which information and computing services can be accessed, by whom,

under which conditions and which actions they are allowed to perform [46, 66-68].

Termination phase contains:

1. Authorization revocation: Removing of rights by the authority.

2. Credentials deactivation: Disabling the possibility of getting the resources by providing the credentials.

3. Account deactivation: The account deactivated and not deleted to maintain the user information.

2.2 Conventional cryptography

Cryptography is the science of providing information security. It can be referred as the process of securing data in transit or stored, against third-party adversaries. Cryptography is etymologically derived from Greek words "kryptós" and "graphein" which means hidden and writing respectively. Cryptography possesses of two important properties; encryption and decryption. Where encryption is a transformation of any plaintext to ciphertext. Whereas, decryption is the opposite process. Both encryption and decryption have to be performed under control of a key. Cryptography has been known and used for thousands of years for securing a message from illegitimate third parties. Nowadays, cryptography has become a fundamental part of modern security systems. In this section, we are going to explain cryptographic primitives like symmetric and asymmetric cryptography as well as hash functions since they are crucial for topic considered in this thesis.

2.2.1 Cryptographic Notions

Cryptographic notations are important to be familiar with since these are crucial for the full understanding of the meaning in particular descriptions. Therefore, it is necessary to emphasize which notations will be consistently used from now of and to the end of this thesis.

Algorithmic Notations

Following are frequently used in this thesis, this important to be familiar with:

- m - message of arbitrary length in form of a bit string $\{0,1\}^*$,
- i - index,
- sk - secret/private key containing all sk_i's,
- sk_i - part of secret/private key sk at index i,
- pk - public key containing all pk_i's,
- pk_i - part of public key pk at index i,
- σ - signature containing all σ_i's,
- σ_i - part of signature σ at index i,
- H - hash function with collision resistance $H : \{0,1\}^* \rightarrow \{0,1\}^n$,
- F - pseudo random function $F : \{0,1\}^n \rightarrow \{0,1\}^n$,
- F_k - pseudo random function from functions family with key k, $F : \{0,1\}^n x \{0,1\}^k \rightarrow \{0,1\}^n$,
- d - message digest, result of $d = H(m)$.
- n - security level in bit, meaning that it is necessary with 2^n operations to break the security.
- $\log n$ - $\log_2 n$

These notations are common for most of algorithms described in this work. Algorithm specific notation will be additionally described in particular subsections.

2.2.2 Symmetric cryptography

In symmetric cryptography, when to parties want to securely communicate with each other they have to share a secret key. That secret key will be used for both encryption and decryption. So, the sender can encrypt a plaintext message using the shared secret key and the receiver can decrypt the message by using the same cryptographic algorithm the sender used and the same shared secret key. The secret key should only be known by the two parties that are communicating with each other. Therefore, an efficient way for exchanging secret keys over public networks was demanded. These are described in Subsection 2.2.3 below. Furthermore, symmetric cryptography is divided into two groups; stream ciphers and block ciphers. These are covered in this Subsection.

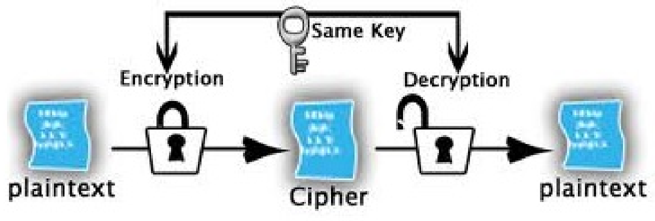


Figure 2.1: Example of classical symmetric cryptography [114].

Stream Cipher

A stream cipher is an encryption method where plaintext message is encrypted bit-wise with a random or pseudo random key stream, normally by an XOR operation. An example of stream cipher is One-Time Pad, and it is known to be theoretically secure. This means that even source of unlimited computing power is not able to break this encryption. Particularly, "theoretical secure" is known to be crypt-analytically unbreakable [98]. However, One-Time Pad does not guarantee integrity and gets insecure if secret keys are reused.

The advantage of stream ciphers is that both encryption and decryption are the same operations which are easy and fast to compute. However, there are some disadvantages as well. Since each bit from plaintext consumes one bit from key stream, it turns out that key streams have to be at least as long as the plaintext. Additionally, the key should also be truly random, this is complex and expensive in large quantities. A stream cipher is Pseudo Random Number Generator (PRNG) which takes a short truly random secrete seed and expand it into a long sequence which looks random. Moreover, key communication is another obstacle. The secret key can be very big, and it has to be transmitted securely [62].

Block Cipher

Block cipher is an alternative to the stream cipher. In a block cipher, the encryption happens on a bigger bulk of data, typically on 128-bit blocks. The procedure of secure communication looks similar to stream ciphers. However, the secret key size is much smaller, so the biggest disadvantages of stream ciphers are eliminated.

In a usual case, where the sender wants to send an encrypted message to the receiver with a block cipher. The sender first has to choose a cryptographic secure algorithm and then establish a secret key shared with the receiver only. Then, the sender may encrypt the message with this cryptographic algorithm by providing both the plaintext message and the secret key to it. The chosen algorithm will slice the message into blocks and encrypt it with the secret key. This procedure happens automatically and creates a secure ciphertext. Now, the ciphertext can be safely sent to the receiver over an insecure channel. The receiver, knowing the secret key can decrypt the ciphertext using the same cryptographic algorithm that the sender used to encrypt the message.

There are mainly two ways for an attacker to decrypt the ciphertext. Namely, have the knowledge of the secret key or break the encryption algorithm, so that one is able to revert the encryption. However, breaking the current standardized symmetric algorithms is still infeasible. Therefore, an attacker willing to steal the information that is encrypted is trying to get the knowledge of the secret key to be able to decrypt it. In block ciphers, the size of keys determinates the security level of the encryption.

Nowadays, in symmetric cryptography, standardized block ciphers are 3DES and AES. Data Encryption Standard (DES) is an older standard from 1977 [43]. DES encrypts plaintext blocks of 64-bit with 56-bit keys. This gives the security level of 56-bit which nowadays is considerate as insecure. In 1978, a more secure variant of DES called 3DES, pronounced triple DES, or TDEA (Triple Data Encryption Algorithm, was introduced. 3DES use $56 * 3 = 168$-bit keys, providing higher security level than original DES. However, the security level of 3DES is not 168-bit as one may think. This is due to *Meet-in-the-middle* attack, which decreases the security level of 3DES to 112-bit.

More recently, in 2001, NIST standardized another block cipher called AES [40]. Advanced Encryption Standard has the advantages that it fast in both hardware and software. This makes the algorithm speed up in different applications. Additionally, AES is even more secure than 3DES. This algorithm comes in three different key sizes: 128, 192 and 256 bit, which is adequate to their security level. The recommendation from NIST is that AES-128 is good enough for daily use. However, for top secret information AES-256 should be used [87].

Recently published paper from 2016 [10] shows the insecurity of 3DES. This is done by performing extended versions of meet-in-the-middle attack, which decrease the security level of 3DES from 112-bit to only 90-bit.

The threshold of what is considered as secure is increasing from year to year. In the last years, a security level of 80-bit is considered as the minimum for being secure enough to use. Consequently, both standardized block ciphers 3DES and AES are considered secure. In November 2017, NIST published an update with a recommendation of 3DES usage [41], where they among other says that 3DES "may be used by federal organizations to protect sensitive unclassified data".

Although, block cipher in its standard ECB (Electronic Codebook) mode is not secure for nowadays purposes. It is recommended to uses block ciphers in other modes like:

- CBC (Cipher Block Chaining),
- CFB (Cipher Feedback),
- OFB (Output Feedback),
- CTR (Counter),
- GCM (Galois/Counter Mode).

Figure 2.2 illustrate the tremendous difference in encrypting a picture with a block cipher in ECB mode versus block cipher in any other modes listed above. The difference comes from the fact that ECB modes encrypt block by block independently with the same key. Whereas, other modes provide some additional data into the next block when encrypted. This emphasizes the importance of choosing the proper mode when using block ciphers. Nowadays, the most commonly used mode is the GCM (Galois Counter Mode) mode which provides both integrity and confidentiality [84]. This mode is most often used in combination with AES.

2.2.3 Asymmetric cryptography

Asymmetric cryptography or Public Key Cryptography (PKC) is a form of encryption where the keys come in pairs. This pair consists of a private key and a public key which are mathematically connected. The private key should be kept secret, and

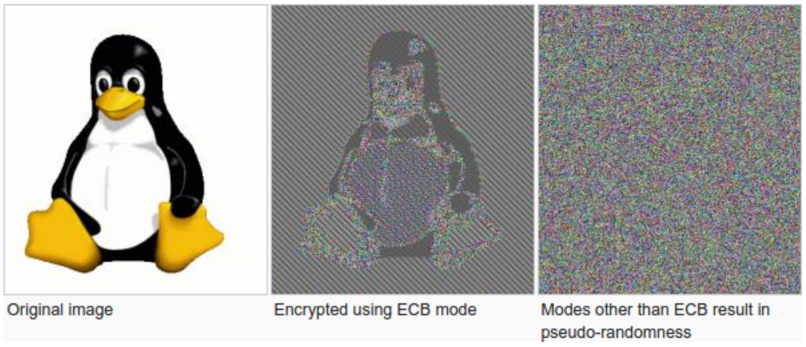


Figure 2.2: Presents AES Encryption in different modes [106].

public key may be exposed to the public. Each party should have its own private and public key. Asymmetric cryptography can be used for encryption, key exchange, and digital signatures.

Security Assumptions

RSA, Diffie-Hellman (DH) and Elliptic Curve Cryptography (ECC) are asymmetric cryptography algorithms that are primarily used today. Their security relies on mathematical problems, which are considered hard for conventional computers to solve. In current conventional standardized algorithms, discrete logarithmic problem and factorization problem are used. Such kind of algorithms are called one-way functions because they are easy to compute in one direction, but the inversion is difficult [39].

Asymmetric cryptographic algorithms such as Diffie-Hellman (DH), ElGamal encryption and Digital Signature Algorithm (DSA) are based on the Discrete logarithmic problem (DLP). The difficulty of breaking these cryptosystems is based on the difficulty in determining the integer r such that $g^r = x \mod p$. The integer r is called the discrete logarithm problem of x to the base g, and it can written as $r = \log_g x$ mod p. The calculations in DLP are executed in algebraic groups. This problem is known to be very hard solved especially if the parameters are large enough since there are no logarithmic operations in algebraic groups. Recently, in 2017, keys equal or larger than 2048 bits are recommended for secure key exchange [87].

Elliptic Curve cryptosystem (ECC) uses a generalization of the discrete logarithm problem [23]. ECC uses a pair (x, y) that fits into the equation $y^2 = x^3 + ax + b$ mod p together with an imaginary point Σ (sigma) at infinity, where $a, b \in Z_p$ and

$4a^3 + 27b^2 \neq 0 \mod p$. ECC needs a cyclic Group G and the primitive elements to be of order G [87].

Factorization problem relies on the hardness of big integers factorization. Given one 2000-bit product of two primes, there are computationally infeasible to find the origin two primes of this product. There are many algorithms that use the hardness of factorization problem, the most known is RSA [87, 169].

Encryption

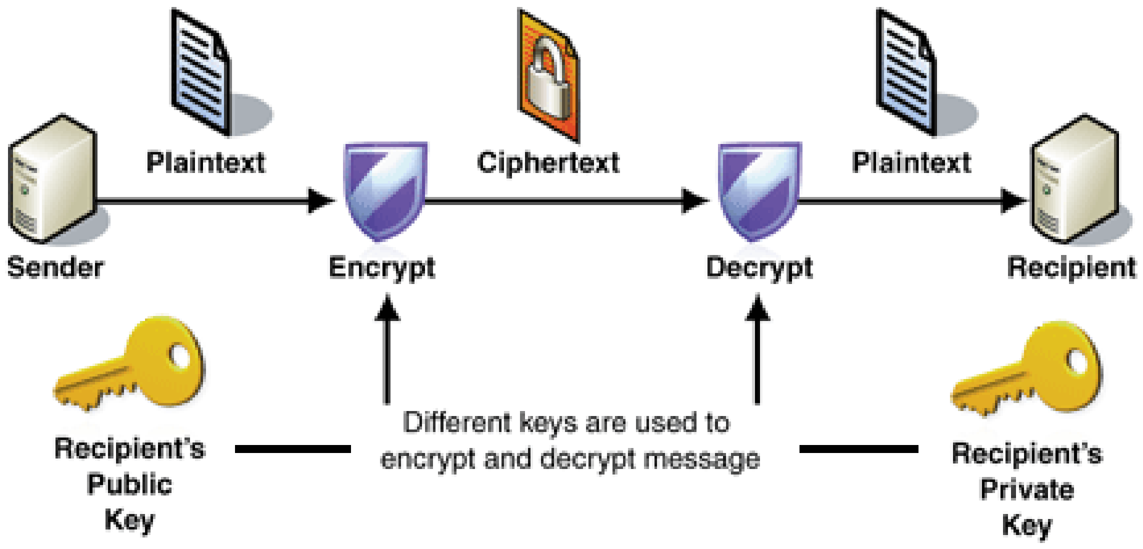


Figure 2.3: An overview of asymmetric cryptography [107].

RSA is a standardized algorithm for asymmetric cryptography that among other provides encryption. As mentioned before and shown in Figure 2.3 on Page 21 two different keys are used for encryption and decryption of a message. Since the keys have a mathematical connection between them, it means that decryption is possible to perform with receivers private key only if the message was encrypted with the receivers corresponding public key.

For instance, if the sender wants to encrypt a message to the receiver. The sender would ask the receiver about his public key or get the public key from the receiver's public repository. Then, the sender would use his public key to encrypt the message. Next, the sender would transmit the encrypted message to the receiver who is able to decrypt the message with his private key.

RSA was invented in 1977 by Ronald Rivest, Adi Shamir, and Leonard Adleman [93]. This algorithm became one of the most important public-key schemes. Its security is based on the difficulty of factorizing product of two primes, and the scheme goes as follows.

Before any encrypted message sending may happen, the receiver has to compute a public key (n, e) and a private key (d). To do so the receiver has to:

1. Choose two large primes p and q, nowadays they should be at least of 1024-bit each. These should also be tested with primality test for example Miller - Rabin [1].
2. Compute $n = p \cdot q$, which usually is bigger than 2048 bits.
3. Use Euler's Phi function: $\varphi(n) = (p-1) \cdot (q-1)$ to select a public exponent $e \in \{1, 2, ..., \varphi(n) - 1\}$ such that $gcd(e, \varphi(n)) = 1$.
4. Compute the private key d by $d \cdot e = 1 \cdot (\text{mod } \varphi(n))$ which can be computed with the Extended Euclidean Algorithm.

After these steps the encryption can be perform using public key with following equation

$$y = x^e \pmod{n}. \tag{2.1}$$

Whereas, the message can be decrypted with equation

$$x = y^d \pmod{n}. \tag{2.2}$$

Where x is a plaintext message and y is the ciphertext. However, these calculations are computationally costly.

Key Exchange

According to Paar and Pelzl [87], RSA and asymmetric algorithms, in general, are not meant to replace symmetric algorithms since they are computationally costly. Therefore, RSA is mainly used for secure key exchange between two parties and often used together with symmetric algorithms such as AES, where the symmetric algorithm does the actual data encryption and decryption. Another widely used standardized algorithm in asymmetric cryptography is the Diffie-Hellman key exchange. With Both mentioned scheme, the sender and the receiver are able to agree on a secret key although the long distance between them. In addition, another family of public key algorithms known as Elliptic Curve Cryptography is extensively used. ECC provides the same level of security as RSA and DLP systems with shorter key operands which makes it convenient to be used in systems with low computational resources [87].

The Diffie-Hellman algorithm was developed particularly for the purpose of key exchange. Here, the sender and the receiver can agree on a common secret without a leak of information and without the need of meeting each other. Some parts of this scheme were already described above in the paragraph about Security Assumptions. However, the full algorithm will be presented for better understanding. In Diffie-Hellman, there are two public parameters called domain parameters. They are p and α, where p is a big prime and α is a generator in the multiplicative group modulo p. Then, each party has to generate a secret number such that only the generated parties have the knowledge of it. Let's assume that the sender generates number a and the receiver generates number b. Next, they both take the generator g and raise it to their generated number and send it to each other. Particularly, the sender calculates $g^a modp = Key_A$ and the receiver calculates $g^b modp = Key_B$. Then, Key_A is send over to the receiver, and Key_B is sent over to the sender, both over the insecure public network. Last, the sender raises the value received from the receiver with her generated number and the receiver does the same with the value received from the sender. So that they both come to the same answer which will become their secret key. Mathematically it look like this, the sender calculate $Key_B^a = Secret\ key$ and the receiver calculate $Key_A^b = Secret\ key$ [34]. A potential adversary will only get the values of g, p, Key_A, Key_B. However, from this information, the adversary is not able to obtain the shared secret key of the sender and the receiver.

A small example with real numbers will give a better understanding.

$$Domain\ parameters : p = 23\ (prime)\ and\ g = 11 (generator) \tag{2.3}$$

$$\textbf{sender:}\ generates\ a = 6 \xrightarrow{calculates} g^a\ mod\ p = 11^6\ mod\ 23 = 9 \tag{2.4}$$

$$\textbf{receiver:}\ generates\ b = 5 \xrightarrow{calculates} g^b\ mod\ p = 11^5\ mod\ 23 = 5 \tag{2.5}$$

$$\textbf{Eve's}\ knowledge :\ g = 11, p = 23, g^a\ mod\ p = 9\ and\ g^b\ mod\ p = 5 \tag{2.6}$$

$$sender \xrightarrow{g^a\ mod\ p=9} receiver,\ receiver \xrightarrow{g^b\ mod\ p=5} sender \tag{2.7}$$

$$\textbf{sender:}\ 5^6\ mod\ 23 = 8,\ \textbf{receiver:}\ 9^5\ mod\ 23 = 8 \tag{2.8}$$

These generated numbers a and b are private keys of sender and receiver respectively. Whereas,

$$g^{ab}\ mod\ p = g^{ba}\ mod\ p \tag{2.9}$$

is their shared secret key, which can be used as key in for example AES encryption.

Digital Signature

Digital signatures may be compared to handwritten signatures. However, they are more secure. Digital signatures provide authentication, non-repudiation, and integrity. For instance, the sender can sign a document digitally by hashing the document and encrypt it with her private key. Then, to verify the signature, the receiver has to decrypt the received document with the sender's public key (authentication and non-repudiation) and hash the original document. Whenever these two hash values match (integrity), then the signature is to trust. Current Digital Signature Standard (DSS) is Digital Signature Algorithm (DSA) which was standardized by NIST in 1993 adopted from FIPS 186 [44]. DSA is a variant of El-Gamal signature scheme [42]. The last update on DSA was provided by NIST in 2013 [66].

Structurally, a digital signature scheme is divided into tree sub algorithms: key generation Kg, signing $Sign$ and verification $Vrfy$ - algorithm.

1. **Kg**: takes as input a security parameter n and generates both secret key and public key (sk, pk).
2. **Sign**: provided with a private key and a message (sk, m) generates a signature σ of given message.
3. **Vrfy**: Takes a signature, message and public key (σ, m, pk) as input and returns "accept"/"reject" or 1/0 in boolean value. A valid signature need to hold the following equation:

$$\mathbf{Vrfy}(pk, Sign(sk, m), m) = 1 \tag{2.10}$$

Security of Digital Signatures

In 1988, Goldwasser et al. [45] provided with a hierarchy of attack models against digital signature schemes. The authors, describes two kinds of attacks, whereas the second consist of four variants.

- Key-only attacks: where the adversary knows only the public key of a signature scheme.

- Message attacks: where the adversary possesses a signature of a known or chosen message.
    - *Known-message attack:* the adversary possess the information of a set of messages along whit their signatures. However, the messages were not chosen by the adversary.
    - *Generic chosen-message attack:* is an attack which does not depend on the public key of the signature scheme. The adversary may choose a fixed number of messages to be signed by the signature scheme. However, the adversary will not be able to change any message after he sees the signatures.
    - *Directed chosen-message attack:* the adversary has knowledge of the public key of the signature scheme and can decide a list of messages to be signed. A message cannot be changed after seeing the first signature.
    - *Adaptive chosen-message attack:* the adversary knows the public key of the signature scheme, he also has the possibility to ask for the signature of any messages and can adapt the messages to already seeing signatures. This is the strongest attack an adversary can mount to a signature scheme.

The authors of [45], have also provided the definitions of breaking a signature scheme. They said that a signature scheme is broken when the adversary is able to perform one of the following attacks in a non-negligible probability

- *A total break*: the adversary learns the secret trap-door of the scheme. (Ex. in RSA it would be the information about the primes p and q.)
- *Universal forgery*: when the adversary finds functionally similar signing algorithm with the same trap-door and is able to forge a signature.
- *Selective forgery*: when the adversary can forge a signature of the message decided by him.
- *Existential forgery*: when the adversary is able to forge a signature of at least one message. The message is not known to the adversary, and it may be a random string.

The strongest security property of these above for a signature scheme is the last one. It is desired to a signature scheme to be immune for existential forgery of an adversary. Then, the signature scheme is Existential Unforgeable (EU).

However, combining the notions of both "breaking the system" and known attacks listen above. The desired security of a digital signature scheme is to be Existentially Unforgeable under Chosen-Message Attacks (EU-CMA) in the standard model. Where the security in the standard model is based on the fact that the adversary is limited by the computational power and the amount of time.

When a signature scheme is proven to be secure using only complexity assumptions (like for example preimage-resistance, second preimage-resistance or collision-resistance), then the scheme is secure in the standard model. Nevertheless, there is difficult to prove the security in the standard model. Therefore an idealized version is used, where the cryptographic function is replaced with a random function. When such a changed scheme is proved secure, then it is said that the scheme is secure in the random oracle model. The random oracle model was first described in 1993 by Beralle et al. [6].

In addition, a property of a good signature scheme is to be forward secure [5]. This means, that when such a scheme gets compromised, for example by leaking the private key, the signatures that already was signed with this key remains secure. To fulfill this criterion, the signature scheme should be a key evolving scheme. This means that the private key of that signature scheme should change after a period of time.

2.2.4 Hash functions

Hash functions are one of most important cryptographic primitives. Hash functions are divided into two main branches, keyed and unkeyed. These have many applications in real wold usage. For instance, one of most popular keyed hash functions applications is MAC or Message Authentications Code. This provides a cryptographic checksum based on secret symmetric keys, as well as authentication and integrity of the message. On the other hand, the unkeyed hash functions are used as MDC or Manipulation Detection Codes.

Hash functions which possess these six properties provide a good standard and security [87].

1. A hash function compress a message of arbitrary length to a fixed length bit string. In practice, it brakes the message into blocks with padding in the last block if needed, and then hash these blocks with the concatenation of previous hashed block. The result of such hash function can be seen as the fingerprint

or a digest of the given message. The output of a hash function may also be called a hash value or a checksum.

2. Hash functions have to be easy to compute. However, this property should not be applied to all hash functions. Hash functions used for password storing should be a bit slower so that brute force attack would not become practical.

3. Fixed length output, hashing a message m with a hash function h then a hash value $z = h(m)$ need to have a specific length.

4. Preimage resistance also known as one-wayness. It has to be computationally infeasible to find x when $z = h(x)$ and z is the output of a hash function when hashing message x.

5. Second preimage resistance or weak collision resistance. This requirement says that given $h(m_1)$ it should be computationally infeasible to find another message m_2 which maps to the same hash value. Thereby it should be computationally infeasible to find message m_2 which maps to the same hash value as the given message m_1.

6. Collision resistance means that there are computationally infeasible to find two different messages that maps to the same hash value. This property is more difficult to achieve than the previous one, since the attacker has access to both variables which can be modified.

These are the most important requirements. When all of them are satisfied, then the security of a hash algorithm depends on its output length. Where longer output yields better security level.

In 1989, both Ralph Merkle and Ivan Damgård proved that whenever a one-way compression function is collision resistant, then the whole hash function will be collision resistance as well. This led to a well known and widely used construction of hash functions called Merkle-Damgård construction [80]. In more detail, to be able to compress messages of any length, the hash function has to slice the messages in fixed length blocks. Then, for each iteration a function f is taking two inputs, an initialization vector (IV) and a block from the message. In the first round, the IV is given, then in the following round the functions f takes the output of the previous iteration of f as an IV in the current round. In the last round of f iterations, when the sliced message is computed, the padding is added. The padding starts with an 1 follows with the necessary amount of 0's and ends with the length of the origin message. The result of all these internal f function iterations is fed into the finalization function that may compress the internal, bigger, state to expected smaller output.

For instance, compressing 1024-bit internal state into 256-bit output. All hash functions from MD family and hash functions from SHA family up to SHA-2 including, are based on Merkle-Damgård construction. Nevertheless, this is not the only cryptographically secure construction. The last standardized hash function SHA-3 from 2015 does not use the construction of Merkle-Damgård. SHA-3, originally Keccack, is using a sponge construction [69].

There are many applications where hash functions are useful, one of them are digital signature schemes. When using hash functions in hash-based digital signatures, there is a need for two more properties as pseudorandomness and undetectability [59]. Undetectability gives the property where an attacker cannot detect whenever a bit is an output from a hash function itself or if it is just a random value. Whereas, pseudorandomness gives a property that random oracles posses. The requester gets a value from a black box based on an input value and an initialization bit. The black box generate a output $g(x)$ from an input x and an initialization bit b. When the bit b is equal to 1, then the black box choose a value from the hash function. Otherwise, it generates a random value (for instance from lazy sampling). The black box needs to remember the previous answers such that it is consistent.

In 2006, Halevi and Krawczyk [49] presented a method to "Strengthening Digital Signatures via Randomized Hashing". The authors, says that by performing bitwise exclusive OR operation on the message with salt before hashing, will free practical digital signature from relying on strong collision resistance. Consequently, yielding better security on lower (second pre-image resistance) security assumption. The same year, the authors had a short presentation of this topic in NIST hash workshop [48]. Three years later, NIST published a recommendation for using randomized hashing in digital signatures [33]. In 2012, NIST published an updated version with recommendations for using randomized hash in different applications [32].

The current standard in the hash function is SHA-3, which was announced in August 2015 [69]. SHA-3 offers arbitrary output length in order to fit to any applications needs. However, SHA-2 family is not outdated and is still secure for use, especially these variants with longer outputs. Both SHA-2 and SHA-3 family provide good and lasting security.

2.2.5 Attacks on Hash Functions

Important to emphasize is to be aware of different attacks on a hash function, especially when using these in security applications. In this subsection, some of the most common attacks on hash functions are briefly described.

Generic Security

The security of hash functions, is often analyzed as a black box testing. It is because in black box testing there is no need to look at the internals, such that many different hash functions may be tested. The results are often expressed in query complexity, namely the number of queries that was needed to break a specific property. The Table 2.1 shows the generic security for hash functions in classical cryptography [59]. As well as, the number of queries needed to break a certain property of a hash function, where n represents the output length of given function.

	One wayness	**Second Image Resistnace**	**Collision Resistance**	**Untraceability**	**Pseudo Random Function**
Classical	$O(2^n)$	$O(2^n)$	$O(2^{n/2})$	$O(2^n)$	$O(2^n)$

Table 2.1: This table shows generic security for hash function in classical cryptography.

A hash function that does not meet the generic security is considered as insecure and should not be used in security applications.

Brute Force

Brute force attack is one of the oldest existing attacks. It can be seen more like an exhaustive search in a database. With brute force attack, an attacker tries all possibilities inputs until he gets expected output. This kind of attack is very unpractical and time-consuming due to the enormous amount of possibilities [87]. Taking into consideration a hash function with an output of 256-bit and the task to break a second pre-image resistance. It will take $2^{256} + 1$ tries to do so. To set the number into a perspective and give it more meaning, it has to be said that the known observable universe consists of $10^{80} \approx 2^{268}$ particles.

Birthday (Collision) Attack

Birthday problem or birthday paradox is often explained with a riddle which goes as follows: How many people need to be in a room so that at least two people have birthday exactly on the same day? Of course, since it is 365 (or 366) days in a year, the safe answer to this question will be 367 people. This gives 100% chance that at least two people have the birthday on the same day. However, taking into consideration the probability theory from math, every answer that has the probability above 50% is correct. Since then it is more likely that given answer is accurate than the opposite case. Consequently, following this logic and probability calculations, leads to the answer of 23 people. This yield the probability of 50.7% of being true. In other words, given 23 people in a room, it is more likely that there are two of those who share their birthday than there is no one that is sharing a birthday with any other in the room. As it can be observed the answer is roughly above the square root of days in a year $\sqrt{365} = 19.1 \rightarrow 1.17 * \sqrt{365} \approx 23$ [87, 301]. Moreover, slightly increasing the number of people to 30 or 50, rises the probability to 70.6% and 97.0% respectively.

Birthday paradox is also used in the cryptographic attack called the birthday attack, which is a real threat to hash functions. Birthday attack reduces the complexity of finding a collision in a cryptographic hash function by a square root. Meaning that a cryptographic hash function with b-bit output length, have the security level of only $b/2$-bit. Since the security drops by a square root. So with a hash function having 80-bit output, there is need of only 2^{40} tries for a collision to occur [87, 293-314].

Having in mind that digital signatures make use of hash functions with collision resistance, this attack may cause a forgery of a signature.

Chosen Prefix Collisions Attack

Collisions attacks are divided into two kinds of attack. First is a plain finding of collision by applying birthday paradox as described above. Second is to find prefixes for two different messages such that they map to the same hash value. In other words, find p_1 and p_2 for messages m_1 and m_2 so that $p_1||m_1$ maps to the same hash value as $p_2||m_2$.

This attack was used by Stevens et al. [100] to find a collision in SHA-1. SHA-1 was standardized by NIST in 1993 and has been widely used since then. Unfortunately, in many years it has been known that SHA-1 has some weaknesses. In 2011, NIST has officially announced that SHA-1 should not be used anymore due to theoretical attacks. However, SHA-1 is still widely used and creates a threat, since the collisions are now practical. The authors was able to find a prefix to colliding messages in such way that makes a user have two arbitrarily chosen distinct visual contents. They concluded that this attack was 100 000 times faster than a brute force attack.

Any applications relying on collision resistance hash functions are threatened by this attack. However, applications that requires second-preimage resistance will not suffer from this attack.

Length extension attack

In this attack, an attacker is trying to determinate the hash value of the concatenation of two messages $H(m_1||m_2)$ [105]. By having the knowledge of both the hash value $H(m_1)$ and length $|m_1|$ of the first message and having control over the second concatenated message m_2. The attacker may be able to extend the origin message m_1 and compute a new hash value yielding a valid signature. Important to emphasize is that the attacker does not have the knowledge of the actual content of the origin message m_1.

In practice, this attack utilizes the way Merkle-Damgård construction [80] is build up. The attack works in following way. When Merkle-Damgård construction is used to hash a secret concatenated with a message $H(secret||message)$. The secret will appear in the first block(s) of a sliced message. Therefore, the internal state after the final iteration of function f will yield a valid signature of given message. However, having the internal state of the Merkle-Damgård construction after the last iteration of function f. Then, adding a second message to the construction and making it iterate through the second as well, will also yield a valid signature. Consequently, as aforementioned, an attacker having knowledge of the length of a message, its hash value, and having control over the second concatenated message is able to create a valid signature.

All hash functions based on Merkle-Damgård constructions like MD0-MD5 and SHA-0-SHA-2 algorithms are exposed to this attack. Nevertheless, both SHA-3 and HMAC are immune to length extension attack, since they do not use the construction of $H(secret||message)$. One of the prevention technique for this attack is to flip the order of these messages to $H(message||secret)$ [38].

2.3 Digital signatures in current protocols

In the time of internet expanse, the use of current standardized protocols as well as creating new protocols grows continuously. Most of the protocols need to be secure and may have to provide a sort of authentication. Nowadays, digital signatures are used to fulfill such security requirements. In a post-quantum world, all of the digital signatures need to be replaced, since they will be directly or indirectly affected by a quantum computer. Therefore it is important to have an overview of what role does a digital signature have in these protocols as well as in which way the signatures are used. In this section, some of most commonly used internet protocols that are containing a digital signature are presented.

2.3.1 PKI

PKI is a framework used to distribute, administrate and use public keys over a network. PKI issues X.509 certificates to authenticate the relationship between a public key and an entity on a network.

To trust an entity over a network, it is necessary to verify its certificate, by verifying the certification path. The certification path starts with the end entity's certificate and proceeds through some intermediate CAs up to a trust anchor or root CA (certificate authority). A trust anchor is a part which trust is assumed and not derived from other sources. The trust anchor is a root CA or a CA which can derive its trust to an underlying intermediate (subordinate) CA or end entity (subscriber) [73].

A variation of this structure exists, where there are several root CAs, in a trust list. In this structure, a subscriber can be verified by any of these root CAs. This structure is used in web browsers,

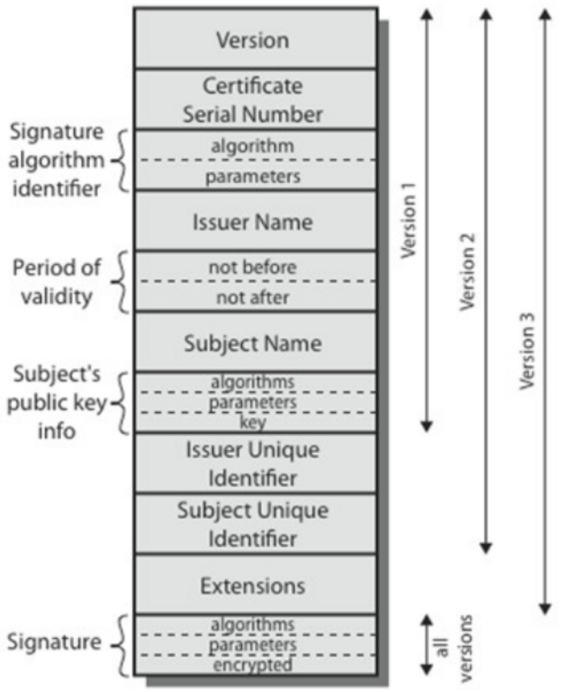


Figure 2.4: Presents 3 versions of X.509 certificate structure [109].

where they are initialized with hundreds of root CAs from the beginning.

Figure 2.4, presents a X.509 certificate and data fields it consist of in different versions [63]. There are 3 versions of X.509 certificate available, where version 3 is the last version of the standard which provides the most information about the end entity and the purpose of the certificate.

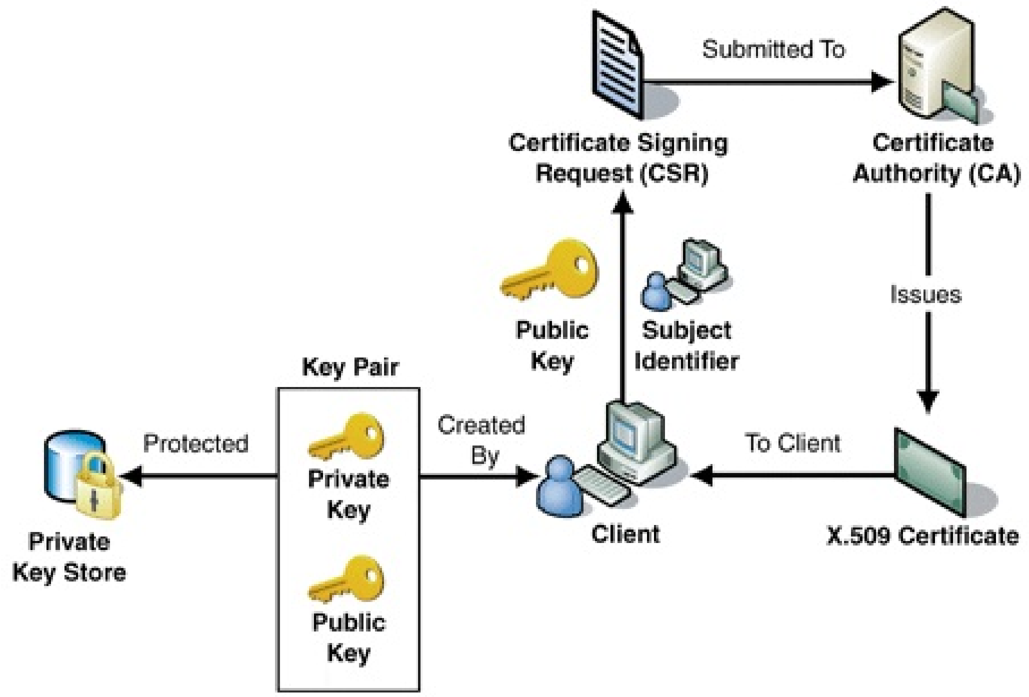


Figure 2.5: The flow of getting an X.509 certificate in Public Key Infrastructure [108].

To get a certificate, the end entity has generated a Certificate Signing Request (CSR) containing information about the end entity, signed by entity's private key. Thereafter the CA will validate the request by verifying the signature and will generate a new X.509 certificate for the end entity if validation was succeeded [74].

A PKI structure also consists of a Certificate Revocation List (CRL). CRL is a signed data structure which continuously keeps control over invalid certificates. Each CA posses a CRL list which keeps track of issued certificates that are revoked. This is an append-only list and is available to the public [28]. However, due to the problem of big CRL list, the OCSP was taken to use. OCSP is an Online Certificate Status Protocol which returns one line digitally signed response instead of CRL list. OCSP creates less overhead on the network while checking the certificate, and have the advantage of requesting the status of a single certificate [96].

A good example of public key infrastructure in Norway is BankID, Buypass, and ID-porten.

2.3.2 TLS

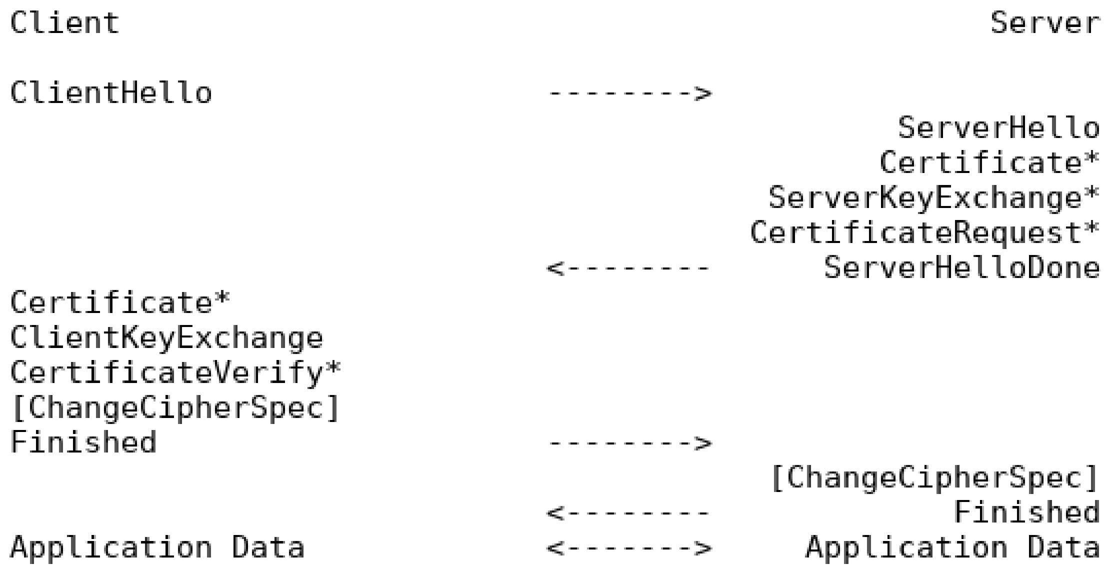
TLS is a widely used protocol to secure data traffic through the internet. A good example of TLS usage is HTTPS protocol used by web browsers. HTTPS protocol is a secure version of HTTP, namely by securing the communication by TLS. The TLS protocol encrypts all traffic between a server and the client with a symmetric cipher due to the performance.

To provide the security, TLS always starts with a handshake between the client and the server. The handshake protocol consists of four alternating messages between the parties, where the client start the conversation. Among other information, a certification exchange message, containing X.509 certificates, is sent. The purpose of this message is to authenticate both parties so that they can trust each other. However, most of TLS implementations often use server authentication only, called one-way authentication. Furthermore, the secret key is established through a key agreement exchange. When both parties share the secret key, then the TLS handshake protocol is completed, and both parties may freely and securely send the application data between them.

Figure 2.6 present entirely handshake protocol of TLS, where fields labeled with * are not required [104]. However, then no authentication is provided and an anonymous session is established.

2.3.3 IPsec

IPsec is a network protocol suite that provides with authentication of an entity, key management, and confidentiality. This protocol uses IKE to perform mutual



```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 2.6: TLS handshake protocol [103].

authentication, establishment and maintaining of Security Associations (SA) to use it with Authentication Header (AH) or Encapsulation Security Payload (ESP) [22].

IKE was originally built on three old protocols namely SKEME(1996), OAKLEY(1998) and ISAKMP(1998). SKEME is a simple and compact protocol which provide key exchange mechanism. SKEME have some flexible tradeoffs between security and performance without increasing complexity of the protocol. As key exchange mechanism, it uses Diffie-Hellmann with fast and frequent key refreshment [68].

OAKLEY protocol is also a key agreement protocol that use Diffie-Hellmann key exchange algorithm to exchange keys across an insecure network connection securely. This protocol supports Perfect Forward Secrecy, key updates and is compatible with ISAKMP [86].

ISAKMP is a protocol for establishing SA and cryptographic keys on a network but does not define the actual key exchange technique [29].

IKE protocol has been updated over time to be more simple and reliable and in 2014 IKEv2 became an internet standard. To establish SA, there is a need for four IKE initial exchanges messages [22].

In IKE_SA_INIT messages, a Diffie-Hellman exchange is performed for establishing the common secret key. This key is used to derive all further keys for IKE SA. Then IKE_AUTH messages ensures that messages are authentication and encrypted expect the headers. It will also authenticate the previous message that was sent. When IKE is done with establishing the SA, IPsec can be executed in AH or ESP mode.

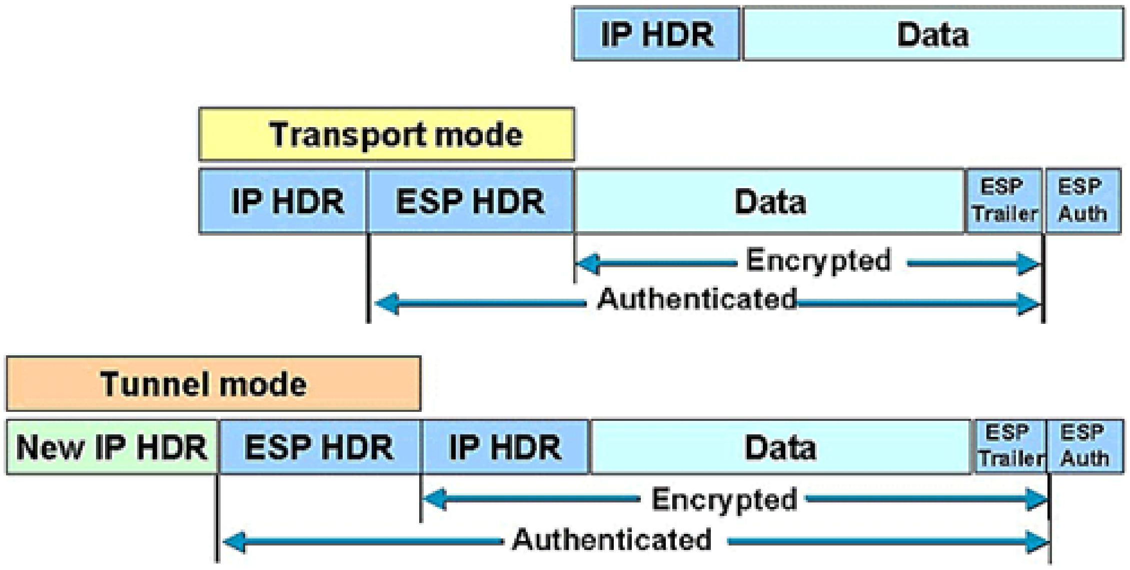


Figure 2.7: AH and ESP presented in both transport and tunnel mode [14].

IPsec in AH mode provides message authentication of payload and immutable header fields by extending IP header [64]. While IPSEC in ESP mode provide encryption and optionally authentication of the payload by extending header and trailer of the originally IP [65]. Both AH and ESP mode can be used in transport or tunnel mode. The tunnel mode adds extra protection of the origin IP header datagram by adding a new IP header in front of the datagram.

2.3.4 DNSSEC

DNSSEC provides a security extension to DNS queries on an IP network. However, DNSSEC is not an internet standard and can be added to servers as an additional security layer. DNSSEC has been invented in 2005 but first in 2014 was allowed to use in .no domains. DNSSEC provides origin authentication and data integrity but does not provide confidentiality or availability. The security in DNSSEC based on public key cryptography and chain of trust.

Chain of trust makes it easier to use an unknown server based on the fact that the server trusts another server which again trusts the root server. Since everybody trusts the root server, then the unknown server can now, based on this fact, be trusted by validation chain. DNS root server is the starting point of the chain and is also known as the trust anchor. This is identical to the PKI infrastructure. First in 15. july 2010 the root anchor was created by ICANN and its "DNSSEC Root Signing Ceremony" [60, 101].

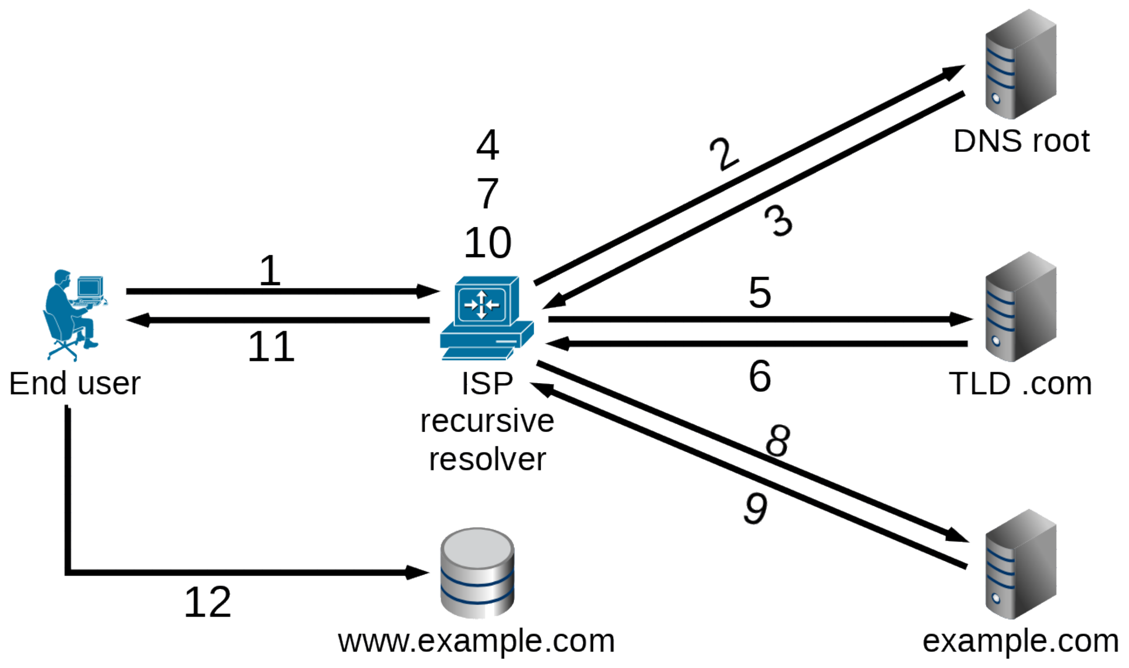


Figure 2.8: Presents a DNSSEC query to www.example.com

When an end user sends a DNSSEC request to a web page with URL www.example.com (step 1). The query goes to ISP recursive resolver which has to find out the A or AAAA record/IP address for given URL. ISP sends then a request to DNS root name sever and ask about the IP address of TLD of the URL which is the .com name server (step 2). Since, DNSSEC flags in on, the DNS root name server answers with a bigger data pack that to regular DNS query. The response contains five parts (step 3):

1. Non secure referral to .com name server.
2. RRset of DNSKEYs: DNS root ZSK_{pub} and KSK_{pub} keys.
3. RRsig of RRset from 2., which is digitally signed by roots KSK_{pvt}.
4. DS record for the .com name server, which is a digest of .com $KSKpub$ key.
5. RRsig of DS from 4., signed by roots $ZSKpvt$ key.

The ISP recursive resolver can now access the .com name server and validate its authentication. For validation it have to verify the root zones records and the root zone it self (step 4).

1. Record verifying
   (a) Root zones KSK_{pub} is used to verify the signature of DNSKEY RRset.
   (b) Root zones $ZSKpub$ is used to verify root zones DS record for .com zone.
2. Zone verifying
   (a) ISP recursive resolver has already a copy of DNS root zones KSK_{pub} key from another source that the DNS root name server itself. Now the root zone can be verified by comparing the local copy of KSK_{pub} key with the KSK_{pub} key send by DNS root zone.

Once again, it can be mentioned that digital signatures are essential for secure usage of a public network.

2.3.5 S/MIME

S/MIME standing for Secure/Multipurpose Internet Mail Extension is an important protocol for signing and encrypting data from MIME protocol. This protocol is able to transmit different types of data between two distinct systems which uses different formats. Formats like: text, audio, images, videos, applications and many others are supported by S/MIME [50].

S/MIME provides with authentication, message integrity and non-repudiation of origin by using digital signatures. As well as, privacy and data security by using encryption. In S/MIME, the certificates are required to protect the authenticity and integrity of public key, thus protecting against man in the middle attack.

Chapter 3

The Impact of Quantum Computing

3.1 Quantum Computing

Quantum computing theory firstly introduced as a concept in 1982 by Richard Feynman. Bone and Castro [12] stated that a quantum computer is completely different in design than a classical computer that uses the traditional transistors and diodes. Researchers have experimented with many different designs such as quantum dots which are basically electrons being in a superposition state, and computing liquids. Besides, they remarked that quantum computers could show their superiority over the classical computers only when used with algorithms that exploit the power of quantum parallelism. For example, a quantum computer would not be any faster than a traditional computer in multiplication. It can be concluded that a quantum computer is a computer that uses the effects of quantum mechanics to its advantage.

3.1.1 Quantum Phenomena

Quantum mechanics is related to microscopic physical phenomena and their specific behavior. In a traditional computer, the fundamental blocks are called bits and can be observed only in two states; 0 and 1. Quantum computers instead use quantum bits also usually referred as qubits. In a sentence, qubits are particles that can exist not only in the 0 and 1 state but both simultaneously, known as superposition. A particle collapses into one of these states when it is measured. Quantum computers take advantage of this property mentioned to solve complex problems. An operation on a qubit in superposition acts on both values at the same time. Another physical phenomenon used in quantum computing is quantum entanglement. When two qubits are entangled, their quantum state can no longer be described independently of each other, but as a single object with four different states. In addition, if one of the two qubits states changes the entangled qubit will change too regardless of the distance between them. This leads to true parallel processing power. The combination of the phenomena mentioned above results in an exponential increase in the number of values that can be processed in one operation when the number of entanglement qubits increases. Therefore, a n-qubit quantum computer can process 2^n operations in parallel [75].

3.1.2 Shor's Algorithm

In 1994, the mathematician Peter Shor in his paper "Algorithms for Quantum Computation: Discrete Logarithms and Factoring" [99], proved that the complexity of factoring large integers would be changed fundamentally with a quantum computer.

The following example will provide a better understanding of how Shor's algorithm factorizes large numbers. In this example, the prime origin factor of number 15 will be found by Shor's algorithm. To do so, there is in need a 4-qubit register. A 4-qubit register can be visualized as a normal 4-bit register of a traditional computer. Number 15 can be represented as 1111 in binary, therefore a 4-qubit register is enough to calculate the prime factor of this number.

According to Bone and Castro [12], a calculation performed on the register can be thought as computations done in parallel for every possible value that the register can take values between 0 and 15. This is also the only step needed to be performed on a quantum computer.

The algorithm does the following:

- n = 15, is the number we want to factorize
- x is a random number such as $1 < x < n - 1$
- x is raised to the power contained in the register (every possible state) and then divided by n. The remainder of this operation is stored in a second 4-qubit register. The second register now contains the superposition results. Let's assume that $x = 2$ which is larger than 1 and smaller than 14.

- Table 3.1 on Page 41 shows the following remainders when x is raise to the powers of 4-qubit register which is a maximum of 15 and divide by 15.

Register 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Register 2	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8

Table 3.1: Result of x raised to the powers of the 4-qubit register values and second results are remainders of dividing these numbers by 15.

In the result, a repeating sequence will appear. In this case, it can be observed that there is a repeating sequence of 4 numbers (1,2,4,8). Therefore, it can confidently stated that the period of this repeating sequence is $f = 4$ when $x = 2$ and $n = 15$. The value f can be used to calculate a possible factor with the following equation: $P = \text{GCD}(x^{f/2} - 1, n)$

In a case where the result of the factor is not a prime number, then the calculation needs to be repeated with different x values.

Shor's algorithm can be used additionally for computing discrete logarithm problems. Vazirani [110] explored in detail the methodology of Shor's algorithm and showed that by starting from a random superposition state of two integers, and by performing a series of Fourier transformations, a new superposition could be set-up to give, with high probability, two integers that satisfy an equation. By using this equation, it can calculate the value r which is the unknown exponent in the DLP.

3.1.3 Grover's Algorithm

Lov Grover created an algorithm that uses quantum computers to search unsorted databases [47]. The algorithm can find a specific entry in an unsorted database of N entries in $\sqrt{N}$ searches. In comparison, a conventional computer would need $N/2$ searches to find the same entry. The algorithm goes as follows.

1. Initialize the system of each N state to be the distribution of equal amplitude of $1/\sqrt{N}$. This can be obtained in $O(log\ N)$ steps.

2. Following operations should be performed $\sqrt{N}$ times.

   (a) The system may be in any state S. If the check of that state gives 1, then rotate the phase by π radians. Otherwise, leave the system unaltered.

(b) Apply diffusion transformation D defined as matrix: $D_{ij} = 2/N$ if $i \neq j$ & $D_{ii} = -1 + 2/N$. This can be written as $D = WRW$. Where, R is the rotation matrix defined as:

$$R_{ij} = 0 \; if \; i \neq j; \tag{3.1}$$
$$R_{ii} = 1 \; if \; i = 0 \; and \tag{3.2}$$
$$R_{ii} = -1 \; if \; i \neq 0 \tag{3.3}$$

Whereas, W is the Walsh-Hadamard Transformation matrix defined as follow:

$$W_{ij} = 2^{-n/2}(-1)^{\bar{i} \cdot \bar{j}} \tag{3.4}$$

Here, $\bar{i} \cdot \bar{j}$ denotes bitwise product of binary representation of i and j.

3. This algorithm, like many other quantum algorithms, is probabilistic. Such that the answer is correct with the probability of at least 1/2.

In this manner a quadratic speed up over an unsorted database can be achieved when a appropriately large quantum computer is available.

3.1.4 Quantum Computing

A company named D-wave works on quantum computers since 1999. In the past several years, D-wave has successfully implemented non-universal quantum computers each year, and for each time they were able to increase its power. Additionally, in 2016, they have successfully implemented a factorization algorithm on their non-universal quantum computer. It was possible by turning the factorization problem into an optimization problem [37]. In 2017, this algorithm was able to factorize factor of two primes up to 200 000, without prior knowledge. Later in this year, D-wave managed to develop a quantum annealing processor with over 2000 qubits, making it the state of the art of non-universal quantum computers [113]. Big companies like NSA, Google, NASA and other, cooperate with D-wave and have got their non-universal quantum computer for research purposes. This kind of quantum computers as D-Wave has made, can be used for purposes like optimization, machine learning, pattern recognition, anomaly detection, financial analysis and many more.

Nevertheless, universal quantum computers come with more benefits due to greater computational power. Universal quantum computers are able to use the full potential of quantum physical phenomena. Combining this with the ability to perform any given task, it provides the potential of being exponentially faster than a traditional computer. This aspect will be used for a number of applications in science and businesses, in fields such as secure computing, machine learning, cryptography, quantum chemistry, material science, optimization problems, sampling, quantum dynamics searching and more.

Meaning that any information that has been, or will, be transmitted on public channels are vulnerable to eavesdropping. Even if the data is encrypted and is safe against current attacks, it can be stored for later decryption, once a practical universal quantum computer becomes available [23].

Many companies are currently working on a solution to be able to make a universal quantum computer. IBM is one of the companies that succeeded in this venture. In March 2017, IBM announced their 17 qubits universal quantum computer with proper error correction [111], which is currently the state of the art of universal quantum computers. They claim to build a universal quantum computer with 50 quantum bits in few years [70]. This number may look small, however such quantum computer with this amount of quantum bits will be more powerful than any existing conventional supercomputers [36].

Both D-Wave and IBM have their quantum computers accessible online for research purposes. It provides the public with the opportunity to participate in the development of quantum algorithms. This has shown to be an efficient way to get insights of quantum computing development.

3.1.5 Challenges in Quantum Computing

Quantum computers come not only with tremendous computing power but also with a lot of difficult technical challenges. Many researchers are working on these problem trying to solve them.

- Quantum algorithms are mainly probabilistic. This means that in one operation a quantum computer returns many solutions where only one is the correct. This trial and error for measuring and verifying the correct answer weaken the advantage of quantum computing speed [67].
- Qubits are susceptible to errors. They can be affected by heat, noise in the environment, as well as stray electromagnetic couplings. Classical computers are susceptible to bit flips where zero can become one or the opposite. Qubits

suffer from bit-flips as well as phase errors. Direct inspection for errors should be avoided as it will cause the value to collapse, leaving its superposition state [21].

- Another challenge is the difficulty of coherence. Qubits can retain their quantum state for a short period of time. Researchers at the University of New South Wales in Australia have created two different types of qubits (Phosphorous atom and an Artificial atom) and by putting them into tiny silicon (*silicon 28*) they were able to eliminate the magnetic noise that makes them prone to errors. Additionally, they stated that the Phosphorous atom has 99.99% accuracy which accounts for 1 error every 10,000 quantum operations [82]. Their qubits can remain in superposition for a total of 35 seconds which is considered a world record [30]. Moreover, to achieve long coherence qubits need not only to be isolated from the external world but to be kept in temperatures reaching the absolute zero. However, this isolation makes it difficult to control them without contributing additional noise [67].

IBM in 2017, introduced the definition of *Quantum Volume*. Quantum volume is a metric to measure how powerful a quantum computer is based on how many qubits it has, how good is the error correction on these qubits, and the number of operations that can be done in parallel. Increase in the number of qubits does not improve a quantum computer if the error rate is high. However, improving the error rate would result in a more powerful quantum computer even on the same amount of qubits [11].

3.2 Consequences of quantum computing

Quantum computing theory has been researched extensively and is considered the destructor of the present state of modern asymmetric cryptography. Quantum computers threaten the main goal of every secure and authentic communication because they are able to do computations that classical (conventional) computers cannot. The power of quantum computing will allow an eavesdropper to intercept the communication channel between authentic parties. This task is considered to be computational infeasible by a conventional computer [15]. In addition, it is a fact that symmetric cryptography can also be affected by specific quantum algorithms. Furthermore, algorithms that can break the present asymmetric crypto schemes whose security is based on the difficulty of factorizing large prime numbers and the discrete logarithm problem have been introduced. It appears that even elliptic curve cryptography which is considered presently the most secure and efficient scheme is weak against quantum computers. The goal of post-quantum cryptography (also known as quantum-resistant cryptography) is to develop cryptographic systems that are secure against both quantum and conventional computers and can inter-operate with existing communication protocols and networks [26]. NSA has already announced plans to migrate their cryptographic standards to post-quantum cryptography [97]. Many post-quantum public key candidates are actively investigated the last years since it is believed that they are resistant to quantum-based attacks. There is no doubt that quantum computers will provide a lot of good invention. However, there is a need to be aware of the consequences it will bring along.

3.2.1 Symmetric cryptography

Quantum computing is considered a minor threat for symmetric cryptography. The main consequence come along with Grover's algorithms. This algorithm is able to perform a search on an unsorted database in $\sqrt{N}$ steps, where N is the length of the database. Therefore, brute force on a 128-bit AES key, which have 2^{128} different possibilities, can be obtain in $\sqrt{2^{128}} = 2^{64}$ steps. Unfortunately, this leads to lower security on all existing symmetric cryptographic algorithms by half.

As mentioned in 2.2.2 the security level of 80-bit is considered secure. Therefore, to prevent weak ciphers, there is a need of doubling the key length to compensate for the square root in Grover's algorithm, to maintain the same security level of symmetric ciphers. Nevertheless, AES originally comes with larger keys support like 192-bit and 256-bit keys. This makes AES with bigger keys have the security level big enough even in a post-quantum era [23]. NSA allows AES cipher to protect classified information for security levels, SECRET, and TOP SECRET, but only with key sizes of 192 and 256 bits respectively [83]. Which can also be an indicator of AES being secure in the post-quantum era. In a more recent paper from 2016, NIST report [26] remarks that if the key sizes are sufficient (doubled), then symmetric cryptographic schemes (specifically the Advanced Encryption Standard-AES) are resistant to quantum computers.

On the other hand, Bone and Castro [12] remarks that a possible application of Grover's algorithm may crack DES, which relies its security on a 56-bit key, with only 185 searches to find the key. Taking into consideration 3DES, its security is enough against conventional attacks. However, this algorithm needs to be modified to be manageable in a quantum world. Since the security level of 3DES will decrease from 112-bit to 56-bit security level due to Grover's algorithm.

Buchmann et al. [7] stated that Grover's algorithm has some applications to symmetric cryptosystems, but it is not that fast as Shor's algorithm.

3.2.2 Asymmetric cryptography

As aforementioned in Subsection 2.2.3, security of modern information systems is fundamentally based on public key cryptographic mechanisms that rely on the difficulties of factoring large integers like RSA or computing discrete logarithms in cyclic groups as in DH or DSA. Recent algorithms based on elliptic curves (such as ECDSA) use a modification of the discrete logarithm. When the parameters are properly chosen, these problems are assumed to be computationally infeasible to solve. There is no known algorithm that may be able to solve these problems on conventional computers.

However, this tends to change due to the quantum computer with enormous computing power with specific algorithm exploiting quantum physics to its advantages. Shor's algorithm, described in Subsection 3.1.2, can make modern asymmetric cryptography collapse since is it known that this algorithm is able to solve factorization problem as well as the discrete logarithm problem in polynomial time.

For many years it has been known that elliptic curve algorithm can achieve the same security level as widely used RSA algorithm, but uses less bits to achieve the same outcome. This means that ECC has smaller memory usage and thus faster than RSA. For instance, to ensure 128-bit security, in RSA there is a need for 3072-bit keys, while ECC only needs 256-bit keys [85]. This tends to change with the introduction of quantum computers. Thus, Shor's algorithm poses a critical threat to asymmetric public key algorithms. A universal quantum computer with enough quantum bits cooperating together and required memory will be able to reveal the secret of both those algorithms. Despite that ECC is considered more secure than RSA, it is easier for a quantum computer to break ECC, since it uses smaller amount of bits. Proos and Zalka [89] explained that 160-bit elliptic curves could be broken by a 1000-qubit quantum computer, while factorizing 1024-bit RSA would require a 2000-qubit quantum computer. The number of qubits needed to break a cryptosystem is relative to the algorithm proposed. In addition, the Authors show in some detail how to use Shor's algorithm to break ECC over GF(p).

This will lead to lack of security, since to all asymmetric keys may be decrypted by this algorithm. Key exchange algorithm will no longer provide the intended security, which means that no symmetric key will be securely established over an insecure network. Along with that, digital signatures will no longer provide any verification of entity. An attacker will be able to generate another public key corresponding to the signature or private key, which will give a valid verification. Therefore, conventional asymmetric cryptography will be not secure anymore when a fully universal quantum computer becomes practical.

3.2.3 Hash functions

The family of hash functions suffers from a similar problem as symmetric ciphers since their security depends on a fixed output length. Grover's algorithm can be utilized to find a collision in a hash function in square root steps of its original length. In addition, it has been proved that it is possible to combine Grover's algorithm with the birthday paradox (Described in 2.2.5). Brassard et al. [13] described a quantum birthday attack. By creating a table of size $\sqrt[3]{N}$ and utilizing Grover's algorithm to find a collision, this attack is said to work effectively. By this to provide a $b-bit$ security level against Grover's quantum algorithm a hash function must provide at least a $3b-bit$ output. Due to Grover's algorithm, the generic security of hash functions has changed as well. Table 3.2 fulfill the Table 2.1 from Subsection 2.2.5 by the quantum information for generic security of hash functions. As a result, some of the present hash functions are disqualified for use in the post-quantum era. Nevertheless, both SHA-2 and SHA-3 family, with 224-bit variants and longer, remains post quantum secure. Taking into consideration hash-based digital signatures, the hash function called BLAKE by Aumasson et al. [3] is also qualified to use. BLAKE is a finalist of the SHA-3 competition organized by NIST [25] and it meets the requirements of generic security for secure hash functions. Subsequently, BLAKE has been optimized and updated to BLAKE2 which is faster than SHA hash function families.

	One wayness	**Second Image Resistnace**	**Collision Resistance**	**Untraceability**	**Pseudo Random Function**
Classical	$O(2^n)$	$O(2^n)$	$O(2^{n/2})$	$O(2^n)$	$O(2^n)$
Quantum	$O(2^{n/2})$	$O(2^{n/2})$	$O(2^{n/3})$	$O(2^{n/2})$	$O(2^{n/2})$

Table 3.2: Generic security for hash function in both classical and quantum world [59].

Part II

Survey of Hash Based Signatures

Chapter 4

One-Time Signatures

One-Time Signatures or OTS for short are used for signing and verifying a message across the internet. Signatures like this must never be used more than once due to the security. For each message to sign, the signer should generate a new set of both private and public keys in order to maintain security level.

The most significant difference between currently standardized digital signature schemes and hash-based signatures is that their construction does not include the use of asymmetric cryptography, or more specific Hidden Subgroup Problem (HSP), to ensure security. Instead, hash-based signatures rely on the security of the hash function itself to achieve the same functionality and security.

In this chapter, we are presenting some one-time signatures with a different approach of using hash functions.

4.1 Lamport Signature

In 1975, W. Diffie wanted to solve the problem to construct a digital signature. Lamport came immediately with a solution, though no optimal one. However, it was taken into consideration and was mentioned in paper [112] by W. Diffie and M. Hellman in 1976. Two years later, a more practical way of generating digital signatures was published [91] by M. Rabin, but still with some utility drawbacks. Lamport has eliminated those drawbacks and created Lamport One-Time Signature (L-OTS) in 1979 [72]. This One-Time signature is also interchangeably called Lamport-Diffie signature (LD-OTS) and is considered as the first hash-based signature scheme.

4.1.1 Parameters and Key Generation

In L-OTS the private key is generated by a Pseudo Random Number Generator (PRNG), and then the public key is obtained by applying a one-way function F on each private key element. Assuming a security level of n-bit. The private key sk will consist of $2n$ pairs of elements generated by PRNG, where each of elements has a length of $2n$-bit. This generation will result in private key sk of size equal to $4n^2$-bit.

$$sk = (sk_{(1,0)}, sk_{(1,1)}, ..., sk_{(2n,0)}, sk_{(2n,1)}) \tag{4.1}$$

The elements, $sk_{(i,0)}$ and $sk_{(i,1)}$ represents the i'th pair of generated random numbers from the private key.

Furthermore, the public key is generated by applying one-way function F on each element from the private key. The one-way function F need to provide with the output size of $2n$-bit. This will result in $2 * 2n = 4n$ new values. All these values together create the corresponding public key. This gives $4n^2$-bit as the size of the public key.

$$pk = (F(sk_{(1,0)}), F(sk_{(1,1)}), ..., F(sk_{(2n,0)}), F(sk_{(2n,1)})) \tag{4.2}$$

This will result with:

$$pk = (pk_{(1,0)}, pk_{(1,1)}, ..., pk_{(2n,0)}, pk_{(2n,1)}) \tag{4.3}$$

The public key should be publish, so that everyone could be able to verify the signature of a message.

4.1.2 Signing

To be able to sign a message of arbitrary length, we need a secure compression function H with collision-resistant. This hash function need to produce an output of $2n$ bit, to provide n-bit security level. Usually, a standardized cryptographic collision-resistant hash function is used. Output of such hash function H on input message m results in a message digest $d = H(m)$. Thereafter, d can then be used bitwise to create the signature. Whenever the bit from d on index i, going from left to right, is 0 then $sk_{(i,0)}$ element from the private key is used for the signature.

Otherwise, when the bit $d_i = 1$, then $sk_{(i,1)}$ is used. Consequently, the signature will contain of $2n$ elements, with total size of $2n^2$-bit. Since, only one of the pair elements from the private key is used to sign each individuality bit from the message digest.

$$\sigma_i = \begin{cases} sk_{(i,0)}, & \text{if } d_i = 0 \\ sk_{(i,1)}, & \text{otherwise} \end{cases} \tag{4.4}$$

$$\sigma = (\sigma_1, ..., \sigma_{2n}) \tag{4.5}$$

The sequence of random numbers from the private key creates the signature for a message digest. This signature can be now send to the receiver along with the original message, to be verified. The unused elements from the private key can be deleted to free the memory.

4.1.3 Verifying

On the receiver side, we have got both the signature σ and the message m from the sender. Then to verify the signature, we first have to hash the message with the hash function H to obtain the message digest d.

$$d = H(m) \tag{4.6}$$

Subsequently, by going through the message digest, we can choose the proper corresponding public key elements for verification. This is the same process as in signing procedure of the messages. Thereafter, apply the one-way function F on each element from the signature, independently. Lastly, compare values produces by F with chosen elements of the public key.

$$\forall \sigma_i \in \sigma : F(\sigma_i) \overset{?}{=} pk_{(i,d_i)} \tag{4.7}$$

If all of these numbers are equal, then the signature is valid. Otherwise the signature is invalid and should be rejected.

4.1.4 Security of L-OTS

In L-OTS, the same private key must never be used again to generate another signature. In a scenario where the signature is sent over an unsecured channel, an adversary has the opportunity to obtain half of the corresponding private key from the signature. Despite, the signature is still secure since the adversary does not know other elements from the corresponding private key. The only way to get knowledge of other private key elements would be to use the public key values and invert the one-way function, which is assumed to be computationally infeasible to perform. Therefore, the signature is secure if the underlying functions are secure and the private key is used only once. Then, having $2n$ non-repeatable random generated pairs in private key gives n-bit security level due to collisions attacks.

However, if a second signature is generated from the same private key, then the adversary may obtain more elements from the private key and will be able to forge a new valid signature. The more signatures are generated from the same private key, the easier it gets for an adversary to generate a new valid signature.

A cryptographic hash function that compresses the message adds a layer of security to the signature scheme. Since it makes it difficult for the adversary to manipulate the actual message, or in this case a message digest, being signed. Since hash function provides pseudo-randomness, a new signature produces with the same key pair will reduce the security level by half. Since in average the adversary may learn about 50% of the private key elements that are not overlapping with the previous signature.

However, if no hash function is used on the message, two messages bitwise complement of each other, are enough for a complete break of L-OTS. Since then the adversary will obtain all of the private key elements.

4.1.5 Reducing the Private Key Size

In order to sign several messages with Lamport signature scheme, one need to generate several instances of key pairs. Unfortunately, this means many key pairs to store. The public key is generated from the secret key, therefore storing it may be omitted. However, the secret key alone is $2n^2$-bit, and having many instances of it will be expensive due to the memory consumption. To avoid this flaw, one may use a deterministic pseudorandom function. Consequently, the signer instead of storing all the secret keys, will be able to reduce it to only one short secret seed and then generate the secret key when needed. This method reduce the memory requirement and is very efficient. A method of reducing the public key memory requirement will be described later in this thesis in Chapter 6.1.1.

4.2 Merkle One-Time Signature

In 1979, Ralph C. Merkle proposed an improved version of L-OTS [79]. This version of L-OTS reduces both key and signature sizes, by slightly changing the approach of message signings. This variant of L-OTS has later on also been called for Merkle OTS.

4.2.1 Parameters

In Merkle OTS, the key generation phase is equal to L-OTS key generation. However, with fewer elements. For the security level n, the private key consist of n elements generated by PRNG, where each element are of size n-bit:

$$sk = (sk_1, ..., sk_n) \tag{4.8}$$

The public key is then obtained by applying the one-way function F on each element from the private key:

$$pk = pk_1, ..., pk_n = F(sk_1), ..., F(sk_n) \tag{4.9}$$

Both private and public key has the length of n^2 bit.

4.2.2 Signing

In Merkle OTS, the signer need only to sign the ones from the message digest. Consequently, there is no need for as much private and public key elements as in original L-OTS. Thus, optimizing both signing and verification process.

To sign a message with Merkle OTS, the signer needs to hash the message m with the hash function H to obtain the message digest $d = H(m)$. Thereafter, whenever the bit on index i in the message digest d is equal to 1, the corresponding element from the private key is placed into the signature. Whereas, when the bit on index i in the message digest d is equal to 0, then the bit gets ignored and the signer proceeds to the next bit. This procedure will result in a signature with the size of $n/2 * n$-bit in average or more precisely the number of ones in message digest d times n. Assume d contains t ones in the positions $j_1, ..., j_t$ then,

$$\sigma = (\sigma_1, ..., \sigma_t) \text{ where } \sigma_i = sk_{j_i} \text{ for } i = 1, ..., t \tag{4.10}$$

4.2.3 Verifying

The verification of the signature is similar to the signing process. The receiver needs to compute the message digest d by hashing the message m with the function H.

$$d = H(m) \tag{4.11}$$

Then, by looking after ones in the message digest, the receiver can pick the right elements from signer public key. The receiver, stores these elements in a variable pk_ver for later comparison, this variable will become a verification key for the message.

$$pk\_ver : \begin{cases} pk_i, & \text{if } d_i = 1 \\ \text{nothing} & \text{otherwise} \end{cases} \tag{4.12}$$

Thereafter, the receiver needs to apply the one-way function F to all signature elements separately. Lastly, the receiver needs to compare these elements with the elements from the public key.

$$\forall \sigma_i \in \sigma : F(\sigma_i) \overset{?}{=} pk\_ver_i \tag{4.13}$$

If every elements of that comparison match, only then the signature is valid.

Verification of Merkle OTS consist of one iteration of H to produce message digest and several iteration of F. The number of iteration is equal to the number of ones from the message digest. In average it will be $n/2$ iterations of one-way function F.

4.2.4 Security

Merkle OTS has one major drawback. Namely, the signature is insecure. An adversary is capable of forging a new valid signature without knowing senders private key. This can be done by simply flipping one bit in the message digest, from 1 to 0 and pretend as it is the original message digest. This is more difficult in practice but still doable. Since a hash function is used on the message. The adversary needs to find a message digest which contains lower number of ones. However, every ones need to be at the same index as in the original signature. Then, the adversary will successfully create a valid signature. Thus, according to signing procedure of Merkle OTS, when a bit in d on index i is 0, then the signer does not have to include its private key from that index. Therefore, the receiver does not have to check it either. Thus, it is a valid signature.

4.2.5 Improvement of Merkle OTS

To overcome the security issues described in the section above, Merkle in his Ph.D. thesis [79] has adjusted the signature. Consequently, the signing process was changed to at first sign all 1's from the message digest, then signs all 1's from the complement of the message digest, which initially was the 0's in the original message digest. This change prevents the forgery of a signature. Since, when one bit in the original message digest flips from 1 to 0, the corresponding bit in the complement message digest will automatically change from 0 to 1. Then, the adversary would need to know sender's private key to be able to sign the flipped bit. The alternative way to forge the signature is to break one-ways of a cryptographic function, which is assumed to be computationally infeasible to perform. Thus the signature is no longer exposed to forgery.

Unfortunately, the improvement in security has also resulted in doubling both the key and signature size. This makes both keys and signature sizes equal to L-OTS sizes. However, the sizes can be optimized, and both keys and signature reduce almost to the same sizes as before in the original Merkle OTS.

The improved version of the algorithm uses both message digest and the complement of message digest to sign a document. This makes the signature scheme inefficient. However, the complement of the message digest can be considered as a checksum of the original message digest with a very bad performance. Nevertheless, the checksum can be constructed more efficiently. Merkle has proposed a checksum which adds only $1 + \log n$ bit to the original signature. Consequently both private and public key gets smaller by almost two-fold, whereas the security remains the same. The new checksum is a count of 0's from the original message digest. The number is represented in binary and added at the end of the message digest to be signed with the message. The new checksum adds only $1 + \log n$ bits to the signature since there is only need for $1 + \log n$ additional bits to represent a number between 0 and n.

Key Generation

The number of elements in both private and public key.

$$l = n + 1 + \log n \tag{4.14}$$

The private key is generated by a PRNG, where each element have size of n-bit.

$$sk = (sk_1, ..., sk_l) \tag{4.15}$$

The public key is obtained by applying one-way function F on all private key elements separately.

$$pk = (pk_1, ..., pk_l) = (F(sk_1), ..., F(sk_l)) \tag{4.16}$$

Signing

Furthermore, to create the signature, the sender needs to calculate the message digest d along with the checksum c for the message digest.

$$d = H(m) \tag{4.17}$$

$$c = \sum_{i=1}^{n} \bar{d}_i \tag{4.18}$$

The variable s represents concatenation of messages digest and the checksum.

$$s = d \mid\mid c \tag{4.19}$$

The following creates the signature with t elements of size n, where t is the number of ones in s.

$$\sigma = (\sigma_1, ..., \sigma_t) \text{ where } \sigma_i = \begin{cases} sk_i, & \text{if } s_i = 1 \\ \text{nothing} & \text{otherwise} \end{cases} \tag{4.20}$$

Verifying

To verify the signature, the receiver need perform similar calculations as when signing the message. First, calculate message digest and the checksum and concatenate them.

$$d = H(m) \tag{4.21}$$

$$c = \sum_{i=1}^{n} \bar{d}_i \tag{4.22}$$

$$s = d \mid\mid c \tag{4.23}$$

Then, by iterating through s, the receiver can pick right verification elements from senders public key. The receiver picks only the public key elements from index i where $s_i = 1$. Thereafter, apply function F on every element from the signature

σ. Lastly, compare all chosen elements from senders public key with the signature elements iterated with function F.

$$\forall \sigma_i \in \sigma : F(\sigma_i) \overset{?}{=} pk_i \tag{4.24}$$

Only when all of these elements are equal, the signature is valid. Otherwise, it need to be rejected.

4.3 Winternitz signature

Few months after L. Lamport published L-OTS, R. Winternitz from Stanford Mathematics Department has proposed another One-Time signature. Winternitz presented the solution to R. Merkle who published it in his paper [79] as Winternitz One-Time Signature (W-OTS). This algorithm improves the signature by decreasing its length by a factor between four and eighth. The main difference between L-OTS and W-OTS is that the second is able to sign several bits simultaneously achieving the same security level. However, the W-OTS scheme suffered from the same problem as L-OTS, particularly the signature forgery due to bit flip. Nevertheless, R. Merkle proposed the same solution to this problem as he did for the L-OTS, namely adding the checksum to the signature. Where the checksum is the number of 0's in the original message digest.

Merkle in his paper [79] provided only an overview of Winternitz's idea and a small example. The first time W-OTS was fully described by Dods et al. [35] in 2005.

W-OTS introduces new notation $F^i(x)$ which is used to describe i times repeatedly use of F on an input x, particularly $F^3(x) = F(F(F(x)))$ and $F^0(x) = x$.

4.3.1 Winternitz parameter

The key generation phase is somehow similar to L-OTS. The private key should be generated by a PRNG, and then the public key will be obtained from the private key. R. Winternitz introduced Winternitz parameter w, which can be freely chosen and describes the number of bits, bigger than 1, to be signed simultaneously.

Winternitz parameter w is a trade-off between the size of the signature and the running time of the scheme. Choosing a high value for w means to sign a greater number of bits simultaneously. This influences the signature scheme in such way that both keys and the signature sizes decrease, but the key generation, signing, and verification run slower because of the computational effort. On the other hand when w is smaller then the private key, public key, and the signature are longer, but the key generation, signing, and verification are faster.

Since the w parameter is crucial for the whole scheme, it has to be properly chosen depending on available resources. This is also one of the first steps in this signature scheme, before the key generation phase. According to G. Becker [4], when the signing process is fast, w can be increased to reduce the signature size. Worthy to mention is that when w increases, the signature generation time increases exponentially. Whereas, when w decreases, the signature size increases linearly. Therefore choosing a too big value for w is not recommended. Nevertheless, the scheme will be secure no matter what value of w is chosen. According to Dods et al. [35], W-OTS is most efficient with Winternitz parameter $w = 3$. However, it is recommended to use W-OTS with $w = 4$, since it is very fast, easy to implement and provides short signatures.

4.3.2 Key generation

When an appropriate value of w for the application is chosen, then l can be calculated. The variable l represents the number of elements in private key and has the following formula.

$$l_1 = \left\lceil \frac{|H(m)|}{w} \right\rceil,\ l_2 = \left\lceil \frac{\lfloor \log_2\ l_1 \rfloor + 1 + w}{w} \right\rceil,\ l = l_1 + l_2 \tag{4.25}$$

Where, l_1 represent the number of private key elements needed for signing the message digest, when signing w-bit simultaneously. Whereas, l_2 represent the number of private key elements to sign the checksum of the message digest.

All elements of the private key are generated from a PRNG and consist of n-bit each.

$$sk = (sk_0, ..., sk_{l-1}) \tag{4.26}$$

The public key is generated by applying F to each element from private key $2^w - 1$ times. The public key should be available to the public along with the parameter w and functions F and H.

$$pk_i = F^{2^w-1}(sk_i) \quad for\ i = 0, ..., l-1 \tag{4.27}$$

$$pk = (pk_0, ..., pk_{l-1}) \tag{4.28}$$

Table 4.1 on Page 59 presents correlation in numbers of how key elements decrease and evaluations of F increase while w grows.

Table 4.1: Dependence between w, private key elements and evaluations of F when $n = 256$

w	2	3	4	5	6	7	8
Keys and signature (l) elements	133	90	67	55	45	39	34
Evaluations of F to produce public key	399	630	1005	1705	2835	4953	8670
Signature size in KB	4,3	2,9	2,1	1,8	1,4	1,2	1,1

4.3.3 Signing

To sign a message the sender has to choose the parameter w and generate l private and public keys. Thereafter, hash the message m with function H to produce a message digest $H(m)$. Next, the sender needs to slice the message in pieces where each piece consists of w bit. If the message digest is not dividable by w, the sender should append additional zero's in the most left position. Resulting in

$$\boldsymbol{d} = (d_0, ..., d_{l_1 - 1}), \tag{4.29}$$

which is in base 2^w notation. After that, the checksum $\boldsymbol{C}$ of $\boldsymbol{d}$ can be calculated. To do so, the sender has to calculate the sum of all differences between $2^w = 8_{10}$ and each of the d_i from the sliced message digest. This is done with the following formula:

$$\boldsymbol{C} = \sum_{i=0}^{l_1 - 1} 2^w - d_i \tag{4.30}$$

The calculated checksum $\boldsymbol{C}$ also has to be sliced in pieces with w bit each, and as well add zero's in the most left position if the division does not go up.

$$\boldsymbol{C} = (c_0, ..., c_{l_2 - 1}) \tag{4.31}$$

Then, the sender needs to concatenate both the message digest and the checksum to a new variable $\boldsymbol{B}$. The slicing remains, therefore, more precisely $\boldsymbol{B}$ consist of l elements in 2^w notation.

$$\boldsymbol{B} = (b_0, ..., b_{l-1}) \tag{4.32}$$

The new variable $\boldsymbol{B}$ is the actual thing to sign. In other words, W-OTS sings both the message digest and the checksum of the message digest. Furthermore, to generate the signature, the sender needs to apply b_i many times the function F on input sk from index i.

$$\sigma_i = F^{b_i}(sk_i)\ for\ i = 0, ..., l-1 \tag{4.33}$$

$$\boldsymbol{\sigma} = (\sigma_0, ..., \sigma_{l-1}) \tag{4.34}$$

Thereby, the signature σ consist of l elements each of n-bit length. Which yields the signature size of ln-bit.

For example, let us assume that the sender has chosen Winternitz parameter to be $w = 3$, and a small message digest $H(m) = 16$-bit.

$$H(m) = 10011110101000110 \tag{4.35}$$

Then, the parameter l calculates to be 8, so that both the private and the public key will consist of 8 elements each. Next, the sender slice the message digest 3 by 3 and append additional 0's most left of the message digest if needed.

$$\boldsymbol{d} = (d_0, ..., d_5) = 010\ 011\ 110\ 101\ 000\ 110_2 \tag{4.36}$$

Thereafter the checksum C of message digest d can be calculated. Then, the sender has to calculate the sum of all differences between $2^w = 8_{10}$ and each of the d_i from the message digest. This can be done with the following formula:

$$\boldsymbol{C} = \sum_{i=0}^{l_1-1} 2^3 - d_i \tag{4.37}$$

Figure 4.1 on Page 61 show calculation of the checksum for this particular example, when $H(m) = 16$ and $w = 3$.

The checksum $\boldsymbol{C}$ need to be converted to binary, sliced 3 by 3 bit and extended with additional zero's from the most left position if needed.

$$\boldsymbol{B} = 26_{10} = 11010_2 = 011\ 010_2 \tag{4.38}$$

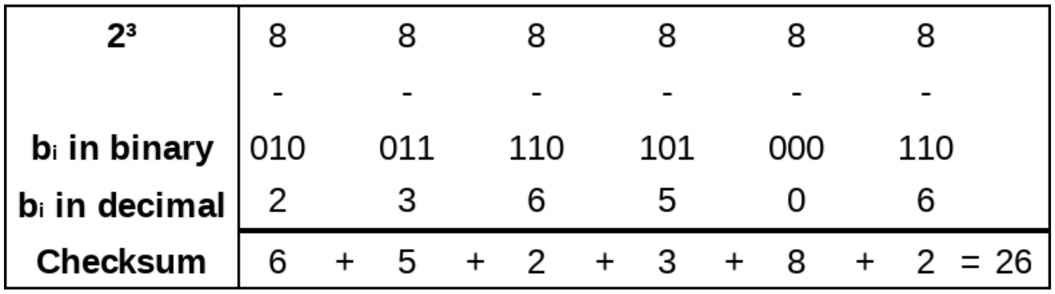
Thereafter, concatenating both message digest and checksum to create B consisting of $t = 8$ elements.

$$\boldsymbol{B} = \boldsymbol{d}||\boldsymbol{C} \tag{4.39}$$

Then, using formula from (Equation 4.33) gives following signature.

$$\boldsymbol{\sigma} = sig(B) = F^2(sk_0), F^3(sk_1), F^6(sk_2), F^5(sk_3), sk_4, F^6(sk_5)||F^3(sk_6), F^2(sk_7) \tag{4.40}$$

In this particular example the signature size is $tn = 8 * 16 = 128$ bit, since F is a length preserving function.



2^3	8		8		8		8		8		8	
	-		-		-		-		-		-	
b_i in binary	010		011		110		101		000		110	
b_i in decimal	2		3		6		5		0		6	
Checksum	6	+	5	+	2	+	3	+	8	+	2	= 26

Figure 4.1: Calculation of checksum in W-OTS, where $w = 3$ and b_i's are parts of the message digest to sign.

4.3.4 Verifying

To verify the signature (σ, m, pk), the receiver has to perform similar calculations as the sender while signing. Knowing both function F and H along with the parameter w, the receiver obtain all necessary information for signature verification. First, hash the message m to obtain message digest $H(m)$, and then append 0's in the most left position if needed. Next, calculate the checksum $\boldsymbol{C}$ and append 0's to the most left if needed as well.

$$\boldsymbol{C} = \sum_{i=0}^{l_1-1} 2^w - d_i \tag{4.41}$$

Them, the receiver concatenates the message digest $H(m)$ with checksum C to create B containing l elements:

$$\boldsymbol{B} = (b_1, ..., b_l) = \boldsymbol{d}||\boldsymbol{C} \tag{4.42}$$

Integer values from b_i in 2^w notation contain information about how many times a specific part of the signature has the function F been applied on. During the key generation phase, the sender has applied function F on private key elements $2^w - 1$ times to obtain the public key (Equation 4.27). Combination of these information implies that the receiver has to apply the function F on the parts of the signature $2^w - 1 - b_i$ times to reconstruct the sender's public key. Thereby, verify the signature.

$$\boldsymbol{\sigma} = (\sigma_0, ..., \sigma_{l-1}) \tag{4.43}$$

$$For\ all:\ F^{2^w-1-b_i}(\sigma_i) \stackrel{?}{=} pk_i \tag{4.44}$$

If the calculated value matches the sender's public key, then the signature is valid. Otherwise, the signature is rejected.

Considering the example from signing subsection above. Calculate message digest $H(m)$, slice it up and interpret it as integer values in 2^w notation. Appending one zero in the most left position to make $H(m)$ divisible by w.

$$H(m) = 010\ 011\ 110\ 101\ 000\ 110_2 \tag{4.45}$$

Then, calculate the checksum $\boldsymbol{C}$ as shown in Equation 4.30 and Figure 4.1 on Page 61. Convert it to binary and split in base 2^w notation. Appending one zero in the most left position to make $\boldsymbol{C}$ be divisible by w.

$$\boldsymbol{C} = 26_{10} = 11010_2 = 011\ 010_2 \tag{4.46}$$

Then, concatenate the message digest $H(m)$ with the checksum C to create B containing l elements:

$$\boldsymbol{B} = H(m)||C = 010\ 011\ 110\ 101\ 000\ 110\ 011\ 010_2 \tag{4.47}$$

Thereafter, interpret these values as integers, which gives.

$$\boldsymbol{B} = (2, 3, 6, 5, 0, 6, 3, 2)_{10} \tag{4.48}$$

Since parameter $w = 3$ then the sender needed to apply function F to all private key elements $2^3 - 1 = 7$ times to generate the public key. Therefore, we can use the formula from Equation 4.44 to reconstruct the sender's public key.

$$F^5(\sigma_0), F^4(\sigma_1), F^1(\sigma_2), F^2(\sigma_3), F^7(\sigma), F^1(\sigma_5), F^4(\sigma_6), F^5(\sigma_7) \tag{4.49}$$

Finally, we can compare all reconstructed public key elements from the signature with the sender's original public key. Only when each of these elements matches, then the signature is valid. Otherwise, reject the signature.

4.3.5 Security of W-OTS

The security of W-OTS relies mostly on hash functions and the checksum. Since the security of a hash function is know, if a cryptographic hash function is used, therefore there is a need to take a closer look at the W-OTS checksum. Forasmuch, the adversary having the intention of forging a new valid signature, may change something in the message, the checksum or the signature it self for his own profit. When the sender sign, for example, four bits simultaneously, then the sender takes the value b_i of these bits in base 10 and hash these four bits b_i times. If these bits were $1001_2 = 9_{10}$, then the sender would hash these bits 9 times to sign them. However, the adversary can take advantage of it and apply function F once more on part of the signature. Then claim that this was the original message (digest) on this place, namely $1010_2 = 10_{10}$ instead of $1001_2 = 9_{10}$. This action was allowed in the original proposal by Winternitz. However, Merkle improved the scheme by adding the checksum of the message. The checksum is a count of how many times a hash function has to be applied to the message to reconstruct the public key of the sender. Consequently, when the adversary changes the number of a hash function applied, then the signature will not be valid anymore. To compensate this disproportion the adversary has to reverse one iteration of a hash function on another part of the message. In other words, the adversary need to break the preimage resistant of a hash function. However, this is a computationally infeasible task to perform, and therefore the signature scheme remains secure against signature forgery.

Taking into consideration the security level of the W-OTS scheme. Since a collision resistant hash function is used to instantiate the scheme. To provide a security level of b-bit, a collision resistant hash function with an output of $n = 2b$ should be used. This is due to the birthday attack on collision resistance.

4.4 Variants of Winternitz Signature Scheme

4.4.1 W-OTS^{PRF}

One variant of W-OTS is called W-OTS$^{\$}$ and was presented by Buchmann et al. [20] in 2011. Authors have shown that W-OTS$^{\$}$ can be used with pseudorandom functions. A function F is pseudorandom when no efficient algorithm is able to determinate if the output is generated from the function F or if it is just a random bit string. This will lead to smaller output and therefore smaller signature while the security level remains the same. This scheme looks similar to its original, but it has some small changes especially when creating the public key and the signature of the message (digest). The authors have also proven that W-OTS$^{\$}$ is Existential Unforgeable under Chosen Message Attack (EU-CMA) in the standard model.

Differences between W-OTS and W-OTS^{PRF}

In this version of W-OTS, the authors have focused only on the core function of the algorithm. Therefore, signing a fixed length message of length m. However, the authors emphasize that an arbitrarily sized message can by signed bu utilizing a collision resistant hash function. Thereby, the authors do not consider the hash function H as part of the W-OTS^{PRF} signature scheme. Nevertheless, we are using $H(m)$ below to be consistent with previous algorithm descriptions.

The main difference between W-OTS^{PRF} and the original W-OTS is that W-OTS^{PRF} is using the output of F_k as the function key for the next iteration. The function applied to the same input x each time. While the original W-OTS uses the output of the function as input for the next iteration and the function key, remain fixed. For this purpose, the notation $F_k^i(x)$ is used to describe that function F_k is iterated i times on input x using function key k. Where the private key element is the function key k in the first iteration of function F_k. Thereafter, the output of the function F_k itself serves as the function key k for the next iteration of F_k.

$$F^i_{F_k^{i-1}(x)}(x), \text{ where } F^0_{sk_i}(x) = x \tag{4.50}$$

Furthermore, notation changes have been made. The Winternitz parameter $w = 2^e$ describes the compression level of the message, where e defines the number of bits signing simultaneously. This change yields some small differences in the formula for calculating the parameter l. However, the calculations give the same result, but at the same time, simplifies notations further in the algorithm.

Key Generation

Key generation algorithm $kg(1^n)$ generates both private key sk and public key pk given a security level n. To do so, first, the sender needs to choose an appropriate value of parameter w to fit the application W-OTS$^{\$}$ will be used within. Then, parameters l, l_1 and l_2 can be calculated by following formula.

$$l_1 = \left\lceil \frac{|H(m)|}{log_2(w)} \right\rceil, \; l_2 = \left\lfloor \frac{\log_2\,(l_1(w-1))}{log_2(w)} \right\rfloor + 1 \tag{4.51}$$

$$\boldsymbol{t = l_1 + l_2} \tag{4.52}$$

Additionally, the sender has to generate a random bit string x of length n. The random string x will be used to generate the public key as well as the signature of the message digest. Thereafter, the sender has to generate l private keys, where each private key element sk_i has the length of n-bit.

$$\boldsymbol{sk} = (sk_1, ..., sk_l) \tag{4.53}$$

To generate public key pk, the sender has to use function F_k and iterate it $w-1$ times on input x with sk_i as the function key k in the first iteration. Then, using the output of the previous iteration of the function F_k as function key k for the next iteration of F_k. The sender has to publish the random bit string x along with the whole public key. The random bit string x is added as a part of the public key pk on index 0.

$$\boldsymbol{pk} = (pk_0, pk_1, ..., pk_l) = (x, F_{sk_1}^{w-1}(x), ..., F_{sk_l}^{w-1}(x)) \tag{4.54}$$

Signing

Aside from the function F_k the signature generation phase is identical to the original W-OTS algorithm. The sender prepares a message m and hashes it with the function H to obtain message digest $H(m)$ of n-bit length. Then, slicing the message digest $H(m)$ into l parts called d_i, each of d_i have the length of e-bit which may contain values between 0 and $w-1$. In other words, the bit string from message digest is written in base w notation. Next, calculate checksum $\boldsymbol{C}$ of message digest $H(m)$ using d_i. Both message digest and checksum should be padded with 0's in the most left position if needed. To calculate the checksum $\boldsymbol{C}$, the sender sums all the differences between $w-1$ and d_i's with the lower and upper limit equals to 1 and l_1 respectively.

$$\boldsymbol{C} = \sum_{i=1}^{l_1}(w - 1 - d_i) \tag{4.55}$$

When all padding is added properly, then the message digest should be of length $l_1 e$ and the checksum of length $l_2 e$. Wherefore, the concatenated message digest and checksum should consist of l parts each with the size of e-bit, lets B represent these parts.

$$\boldsymbol{B} = (b_1, ..., b_l) = H(m) \mid\mid C \tag{4.56}$$

Integer values of b_i is the number of iteration of F_k on x with the private key from the i-th place sk_i as function key k for the first iteration of the function F_k. The output of function F_k becomes the function key k of the next iteration on function F_k. Proceeding like this for all l parts of B, the sender will produce a signature for this specific message digest and checksum.

$$\boldsymbol{\sigma} = sig(B) = (F^{b_1}_{sk_1}(x), ..., F^{b_l}_{sk_l}(x)) \tag{4.57}$$

Verifying

Verification process in also almost identical to the original version of W-OTS, apart from how the verifier applies the nest iteration of function F_k. The receiver has to perform similar calculations as the sender under signing. First, recreate the concatenation of both padded message digest and padded checksum in base w notation, the variable $\boldsymbol{B} = (b_1, ..., b_l)$, using Equation 4.55. Then iterate $w - 1 - b_i$ times the function F_k on input pk_0 with signature element sk_i as function key k for the first iteration. Then, the output of function F_k as the function key k for next iteration of function F_k.

$$\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_l) \tag{4.58}$$

$$(F^{w-1-b_1}_{\sigma_1}(pk_0), ..., F^{w-1-b_l}_{\sigma_l}(pk_0)) \overset{?}{=} (pk_1, ..., pk_l) \tag{4.59}$$

Thereafter, the receiver has to compare the result of his iterations of F_k with sender's public key. If every element matches, the signature is valid, otherwise the signature should be rejected.

4.4.2 W-OTS$^+$

The second variant of W-OTS from 2013 is called W-OTS$^+$ and was presented by A. Hülsing in [52].

Differences from previous versions

W-OTS$^+$ has changed the chaining function c and added randomization elements $\boldsymbol{r}$ to the scheme. Now, for every iteration i of the chaining functions c, the function key k remains the same. Whereas, the output from the last iteration of the chaining function is xor'ed with a randomization element r_i and is used as input for the next iteration of the chaining function. The function itself has been changed to non-compressing cryptographic hash function family.

$$c^i_k(x, \boldsymbol{r}) = f_k(c^{i-1}_k(x, \boldsymbol{r}) \oplus r_i) \tag{4.60}$$

The original W-OTS scheme relies on collision resistant hash function which is threatened to collisions attack. Therefore, there is a need for bigger functions output to provides the same security level. Thus, slower the whole scheme, since in general hash functions get slower when output size increases.

In [16], the authors provided a construction to create a function family by using two iterations of cryptographic hash function. Such a function family is used to instantiate W-OTS^{PRF}, which yields doubled runtime compared with W-OTS^{+}. However, when the function family is constructed of a block cipher, the run times are the same for both schemes. Considering the security, W-OTS^{PRF} contains parameter w as negative linear term on the security level $(n - w - q - 2\log(lw))$. Which creates a limitation in choosing of parameters, when a fixed security lever is requested. Whereas, W-OTS^{+} looses only $\log w$ on security level when parameter w increases. Therefore, W-OTS^{+} provides better security compared to its predecessors, with security level equals to $n - \log(w^2 l + w)$.

However, W-OTS^{+} consist of slightly bigger public key due to the additional randomization elements.

Key Generation

In this variant, choosing the security parameter n, Winternitz parameter w and calculating l is the same procedure as in previous W-OTS versions, as well as private key sk generation. The private key sk consist of l random generated bit strings.

$$\boldsymbol{sk} = (sk_1, ..., sk_l) \tag{4.61}$$

Additionally, the sender should also generate an additional $w - 1$ n-bit random strings. These will be used as randomization elements while producing public key pk, generating a signature σ and in the verification process. The notation for these random strings is $\boldsymbol{r} = (r_1, ..., r_{w-1})$. Furthermore, the sender has to generate one more n-bit random string k which will be used as the function key for all iterations of the chaining function c_k^i.

Then, to generate the public key pk, the sender needs to apply the chaining function c_k on his private key sk elements $w - 1$ times. Previously generated functions key k is used for every iteration i of chaining function c_k. At the first iteration of chaining function c_k, private key elements oxr'ed with randomization element r_1 are used as input. In the followings iterations of chaining function c_k, the output of previous iterations oxr'ed with randomization element r_i is used as input. Both r and k are included in the sender's public key on index 0 as first public key element pk_0. This gives the following public key.

$$\boldsymbol{pk} = (pk_0, pk_1, ..., pk_l) = ((\boldsymbol{r}, k), c_k^{w-1}(sk_1, \boldsymbol{r}), ..., c_k^{w-1}(sk_l, \boldsymbol{r})) \tag{4.62}$$

Signing

The procedure of signature creation is very similar to the previous versions of W-OTS. The sender has still to divide the message digest $H(m)$ in l_1 parts each of length e-bit, represented by $d = (d_1, ..., d_{l_1})$. Then calculate the checksum C in the same manner as previously and divide it in l_2 parts, each of e-bit length, represented by $C = (c_1, ..., c_{l_2})$. The sender should add padding in both d and C in most left position if needed, such that both are dividable by e without rest. Thereafter, the sender need to concatenate d and C to create $B = d \mid\mid C$, then slice B in l parts in base w notation, represented by $B_w = (b_1, ..., b_l)$. Base w representation means that each element b_i consist of e-bit and can hold a value between 0 and $w - 1$. After this, the sender can produce the signature in the following way:

$$\sigma = (c_k^{b_1}(sk_1, \boldsymbol{r}), ..., c_k^{b_l}(sk_l, \boldsymbol{r})) \tag{4.63}$$

By applying b_i times the chaining function c_k with function key k on bitwise xor of private key element and corresponding randomization element r_i from r as input.

Verification

To verify the signature, the receiver has to recompute $B = (b_1, ..., b_l)$ and iterate $w - 1 - b_i$ more times the chaining function c_k with function key k on bitwise xor'ed input between signature elements and corresponding randomization element r_i. The notation $r_{a,b}$ represents the subset of randomization elements between r_a and r_b. In case where a is bigger than b then $r_{a,g}$ is assumed to be a empty string.

$$\boldsymbol{pk} \stackrel{?}{=} ((r, k), c_k^{w-1-b_1}(\sigma_1, r_{b_1+1,w-1}), ..., c_k^{w-1-b_l}(\sigma_l, r_{b_l+1,w-1})) \tag{4.64}$$

If the compression of the sender's original public and the recomputed public key from the signature are equals, then the signature is valid. Otherwise, the signature is not valid and should not be trusted.

4.4.3 WOTS-T

In 2016, Hülsing et al. presented WOTS-T [58] an improved version of W-OTS$^+$. It turned out that, previous W-OTS versions were vulnerable to multi-target attacks. Meaning that when an adversary wants to break the scheme which is using a second-preimage resistant function with n-bit output. The complexity of breaking such function is $O(2^n)$. However, this is true when the adversary has only one try of requesting the function. Whereas in signature schemes as WOTS, the adversary will learn several hash function outputs. Thus, if the number of learned outputs is d, then the complexity of breaking the function reduces to $O(2^n/d)$. To compensate for the loss of security, the scheme needs to use longer functions output, which again leads to an increase in signature size and slows down the whole scheme. Therefore, the authors have changed the construction of the scheme to improve its security and mitigate the multi-target attack.

The most significant difference between W-OTS$^+$ and WOTS-T is the second uses new keys and bitmasks for each call to a hash function. Thereby, the chaining function has changed to be:

$$c^{i,j}(x, a_c, \mathrm{SEED}) = F(k_{i,j}, c^{i-1,j}(x, a_c, \mathrm{SEED}) \oplus r_{i,j}) \tag{4.65}$$

Where a_c is the chain address bit string. Zero iteration of the chaining function returns the input itself, $c^{0,j}(x, a_c, \mathrm{SEED}) = x$.

In order to adapt to the changes, the authors have introduced an addressing scheme for hash function calls. They suggest using a recursive addressing scheme that numbers sub-structures inside a structure. A function GenAddr a_{c_i} takes as input address of the structure and the index of the substructure and generates a unique bit string address for the substructure. The GenAddr function is used to generate both function key $k_{i,j}$ and bitmask $r_{i,j}$.

$$k_{i,j} = F_n(\mathrm{SEED}, \mathrm{GenAddr}(a_c, 2(j+i))) \tag{4.66}$$

This function creates the function key $k_{i,j}$ of length n-bit, used in the i'th iteration of chaining function $c^{i,j}$. Whereas, the following creates n-bit length bitmask to use in the i'th iteration of chaining function $c^{i,j}$.

$$r_{i,j} = F_n(\mathrm{SEED}, \mathrm{GenAddr}(a_c, 2(j+i)+1)) \tag{4.67}$$

The authors have proven that WOTS-T is Existential Unforgeable under Chosen Message Attack (EU-CMA) in the standard model.

Key Generation

In order to create WOTS-T keys, one needs to generate three variables.

- S - secret seed of n-bit length for private key generation,
- a_{OTS} - a unique address bit string,
- SEED - random n-bit string

Next, calculate variables $l = l_1 + l_2$ by choosing Winternitz compression level parameter w, (Equation 4.51). Then, the private key sk can be generated as follow.

$$sk_i = F_n(S, \text{GenAddr}(a_{OTS}, i)) \tag{4.68}$$

Giving l private key element each of length n-bit.

$$sk = (sk_1, ..., sk_l) \tag{4.69}$$

Next, the public key is obtained by applying $w-1$ times the chaining function $c^{i,j}$ on input consisting of the private key element, a unique address, and the public seed.

$$pk = (pk_1, ..., pk_l) = (c^{w-1,0}(sk_1, a_{c_1}, \text{SEED}), ..., c^{w-1,0}(sk_l, a_{c_l}, \text{SEED})) \tag{4.70}$$

Where $a_{c_i} = GenAddr(a_{OTS}, i)$. The public key has the same size as the private key. However, the secret seed S requires less storage than private key sk. Therefore, both private and public keys are generated on the fly when needed. Consequently, the private key sk consist of the secret seed S of length n-bit.

Signing

The signing process is the same as in W-OTS$^+$, however now signing of the message is performed with the changed chaining function $c^{i,j}$. Therefore, we skip the description and refer the reader to subsection above for the details. In short, calculate $\boldsymbol{B} = (b_1, ..., b_l)$ by concatenating both message digest $H(m)$ and its checksum $\boldsymbol{C}$ and interpret it as bases w notation. Since, only secret seed S is stored as private key, the private key elements need to be recalculated by the same formula as in key generation phase, $sk_i = F_n(S, \text{GenAddr}(a_{OTS}, i))$. Then, the signer generates the signature by the following formula.

$$\sigma = (\sigma_1, ..., \sigma_l) = (c^{b_1,0}(sk_1, a_{c_i}, \text{SEED}), ..., c^{b_l,0}(sk_l, a_{c_l}, \text{SEED}) \tag{4.71}$$

Where, $a_{c_i} = GenAddr(a_{OTS}, i)$ gives an unique bit string address for index i. The chaining function $c^{i,j}$ is applied b_i times starting from 0.

Verification

To verify the signature the receiver need message m, signature σ, unique address a_{OTS} and the public seed SEED. Next, calculate $\boldsymbol{B} = (b_1, ..., b_l)$ and apply $w-1-b_i$ many times the chaining function $c^{i,j}$ starting at index $j = b_i$ on input consisting of the signature element σ_i, an unique bit string address a_{c_i} and the public seed SEED. Where, $a_{c_i} = GenAddr(a_{OTS}, i)$.

$$pk = (pk_1, ..., pk_l) = (c^{w-1-b_i,b_i}(\sigma_1, a_{c_1}, \text{SEED}), ..., c^{w-1-b_l,b_l}(sk_l, a_{c_l}, \text{SEED}) \quad (4.72)$$

4.4.4 LM-OTS

Leighton-Micali One-Time Signature (LM-OTS) is yet another Winternitz based signature scheme. This one-time signature scheme was proposed along with Leighton-Micali Scheme (LMS) a many-time signature scheme, and the US patented by Leighton and Micali in [71] in 1995. Their work was inspired by Lamport, Diffie, Merkle, and Winternitz. LM-OTS, LMS and its Hierarchical version (HSS) were proposed for standardization in an IETF draft [77] in 2013. In this subsection, we describe LM-OTS.

Parameters and Variables

The parameters are similar to other W-OTS versions. Parameter w is the compression level, l_1 and l_2 represent number of elements when both message digest $h(m)$ and the checksum C are represented in base $w+1$ notation respectively.

$$w = 2^e - 1 \quad (4.73)$$

$$l_1 = \frac{n}{e},\ l_2 = \left\lceil \frac{\lfloor \log(l_1 w) + 1 \rfloor}{w} \right\rceil \quad (4.74)$$

$$l = l_1 + l_2 \quad (4.75)$$

The checksum is calculated from a hashed value of the message digest concatenated with other parameters. This hash value will be called h and in base $w+1$ notation will be described as $h = (h_1, ..., h_{l1}$. However, one can calculate the checksum with the same formula as previously.

$$\boldsymbol{C} = \sum_{i=1}^{l_1} (w - h_i) \quad (4.76)$$

In addition, one should also have the following:

- I - a string identifying the owner of the public key,
- Q - an indication of which instance of the scheme is being used,
- i - the number of times H has been applied.

All these parameters create variable s.

$$s = I \mid\mid Q \mid\mid i \tag{4.77}$$

The authors defines their chaining function F_s with function key s as repetitive iterations of hash function H. For $0 \leq b \leq f \leq w$

$$F_s(s; b, f) = \begin{cases} x & \text{if } b = f \\ F_s = H(x \mid\mid s \mid\mid b \mid\mid 00); b+1, f & \text{if } b \leq f \end{cases} \tag{4.78}$$

Key Generation

Given (n, w, I, Q), a sender can generate both private and public key. The private key sk consist of l elements each of n-bit length, chosen uniformly at random.

$$\boldsymbol{sk} = (sk_1, ..., sk_l) \tag{4.79}$$

Additionally, creating variable $s = I||Q||i$, where i is an index starting at 1 and going up l. Next, the sender may compute the public key by following formula:

$$pk_i = F_s(sk_i^0; 0, w) \tag{4.80}$$

Where the first value in function F_s is the private key element at index i, whereas the second and third values are from and to values respectively. Then, the public key pk consist of the single hashed value of concatenation of public key elements pk_i, parameter I, parameter Q and starting index 01.

$$\boldsymbol{pk} = H(pk_1 \mid\mid ... \mid\mid pk_l \mid\mid I \mid\mid Q \mid\mid 01) \tag{4.81}$$

Signing

To sign a message the signer need first to create a random string r of length n-bit, chosen uniformly at random and include it to the signature at index 0. Next, compute $h = H(M \mid\mid r \mid\mid I \mid\mid Q \mid\mid 02)$, and the checksum $\boldsymbol{C}$. Then concatenate h with the checksum C to obtain variable B and interpret it in base w notation.

$$B = (b_1, ..., b_l) = h \mid\mid C \tag{4.82}$$

Then to sing the message, the sender needs to apply b_i many times the function F_s with function key s on private key element sk_i.

$$\sigma_i = F_s(sk_i^0; 0, b_i) \tag{4.83}$$

$$\sigma = (r, \sigma_1, ..., \sigma_l) \tag{4.84}$$

Verification

To verify the signature, the receiver has to reconstruct the variable B by first recomputing h, then calculate the checksum C and concatenate the both h and C together. Then, the receiver has to apply $w - b_i$ many times, the function F with function key s on signature element σ_i.

$$pk_i = F_s(\sigma_i; b_i, w) \tag{4.85}$$

Next, concatenate all public key element together with identification I, indicator Q and 01 and the end. Then, hash all these values and compare the result with the sender's original public key. If the comparison holds, the signature is valid, otherwise, reject the signature.

4.4.5 WSS-N W-OTS Using Nonadjacent Forms

In June 2018, Dongyoung et al. [94] presented a new method of using W-OTS$^+$ with nonadjacent forms (NAF), which they call WSS-N. This method uses a biased distribution of 0,1 and -1, while W-OTS$^+$ is using a uniform distribution of binary representation.

This has further implications in signature creations. The authors show that WSS-N needs fewer hash functions call than Winternitz Signature Scheme based on Binary representation (WSS-B). For specific parameter and 256-bit classical security level, WSS-N managed to create the signature 8% faster than WSS-B with same parameters. However, with the cost of longer both key generation and signature verification time. The authors emphasize that the signature generation time is not the big bottleneck of OTS but the key generation time. Nevertheless, in devices like fire- and seismic -sensors, the keys can be generated in advance, whereas the signature needs to be generated fast.

The authors have also proved that WSS-N scheme is existentially unforgettable under adaptive chosen message attack (EU-CMA) in the standard model.

Chapter 5

Few Time Signatures

Few time signatures are signatures that can be used more than once. This is an improvement to be able to sign a few messages with only one key generation phase. However, the usage of these signatures is not unlimited. Forasmuch, after each uses the security level of such signature scheme decrease to some degree. Supposing that one use few-time signature scheme too many times, the private key will be compromised, and the scheme will be insecure. Hence the name, a signature scheme that can be used only a few times due to its security.

5.1 Bins and Balls

Bins and Balls (BiBa) is a few-time signature scheme introduced by Adrian Perrig in 2001 [88]. BiBa signature scheme was further extended to design a new protocol for broadcast authentication. In this section, only BiBa signature scheme will be taken into consideration. BiBa exploits the birthday paradox to achieve efficiency and security. This leads to very short signature and fast verification process.

An allegory to the name of this signature scheme is that the sender has many balls to throw (SEALs - private key elements) into few bins (output of G_h). Whenever two balls land in the same bin, a signature is generated.

5.1.1 Key Generation

The functions used in BiBa signature scheme are:

- $F = \{0,1\}^{m_2} x \{0,1\}^{m_1} \rightarrow \{0,1\}^{m_2}$ - a pseudo random function family.
- H - a hash function in the random oracle model.
- $G_h = \{0,1\}^{m_2} \rightarrow [0, n-1]$ - an instance of hash function family with function key h.

Where n describes the range of output values ($[0, n-1]$) of the hash function G_h, in where the collisions need to happen in order to create the signature. This number strongly depends on variables k and P_s. The parameter k describes the number of collisions that need to occur in G_h in order to generate a signature for a specific message. The parameter k determines the length of the signature. Whereas, variable P_s describes the probability of generating the signature in one trial. The expected number of tries that the sender should perform to find a signature is $1/P_s$. Where P_s is often set to 0.5, this means that the sender should be able to generate a signature no later than in the second trial ($1/0.5 = 2$). To increase the probability of finding the signature, the sender has to increase the number of private key elements t.

Proper values of n for requested security level P_f can be found in Table 5.1 on Page 76. Functions F, H and G, as well as parameters k and n, are publicly known.

k	n	P_f	k	n	P_f
2	762460	$2^{-19,5403}$	13	192	$2^{-91,0196}$
3	15616	$2^{-27,8615}$	14	168	$2^{-69,1001}$
4	3742	$2^{-35,6088}$	15	151	$2^{-101,3377}$
5	1690	$2^{-42,8912}$	16	136	$2^{-106,3119}$
6	994	$2^{-49,7855}$	17	123	$2^{-115,7250}$
7	672	$2^{-56,3539}$	18	112	$2^{-115,7250}$
8	494	$2^{-62,6386}$	19	104	$2^{-120,6079}$
9	384	$2^{-68,6797}$	20	96	$2^{-125,1143}$
10	310	$2^{-74,4851}$	21	89	$2^{-129,5147}$
11	260	$2^{-80,2237}$	22	83	$2^{-133,8758}$
12	222	$2^{-85,7386}$	23	78	$2^{-138,2788}$

Table 5.1: Security parameters of BiBa signature scheme when t=1024 [88].

Further, the private key elements sk_i are called SEALs which stands for SElf Authenticate vaLues. The amount of SEALs is described with parameter t and can vary each time. The parameter t has a direct influence on both private and public key length. Therefore, it should be chosen carefully. Often used value for parameter t is 1024, which yields both the private and public keys of 1024 elements. The private key elements sk_i, have to be generated at random, each of length m_2.

$$sk = (sk_1, ..., sk_t) = the\ SEALs \tag{5.1}$$

To obtain the public key, the sender has to apply function F on input 0 and the private key element sk_i as functions key. The calculations will result in t public key elements, each of length m_2.

$$pk = (pk_1, ..., pk_t) = (F_{sk_1}(0), ..., F_{sk_t}(0)) \tag{5.2}$$

5.1.2 Signing

To sign a message with BiBa signature scheme the sender needs first to concatenate a message m with a counter c, which initially contains the value of 0. Next, hash the concatenated parts with the function H so that $h = H(m||c)$. Then, calculate hash values of all private key elements sk_i separately with the function G with h as the function key.

$$G_h(sk_i)\ for\ i = 1, ..., t \tag{5.3}$$

The results produces t values in range $[0, n-1]$. Then, the sender needs to look after k private key elements which map to the same hash value. In other words, the sender is searching for k collisions to occur in G_h. If the sender is successful, then the signature consists of the private key elements which made the collisions happen along with the counter c.

$$\sigma = (\sigma_1, ..., \sigma_k, c) = (sk_{i_1}, ..., sk_{i_k}, c) \tag{5.4}$$

However, in the case where the sender is not able to find k collisions in G_h. Then, the sender increases the counter c and repeat the calculations of $G_h(sk_i)$. The whole procedure of finding signature (collisions) is repeated until success. When an appropriate number of collisions k is found, then the signature is created.

5.1.3 Verifying

BiBa signature scheme had the most efficient verification process at the time when published. Since the signature size is small, there is no need for much computation.

The verification process goes as follows. The receiver has already the knowledge of functions H, G, F, parameters k, n and public key pk since the sender has published them.

So to verify the signature $\sigma = (\sigma_1, ..., \sigma_k, c)$, the receiver has to first apply the function F with signature element σ_i as function key on input value 0. Thereafter, compare these values with sender's public key elements.

$$for\ each : F_{\sigma_i}(0) \overset{?}{=} pk_i \tag{5.5}$$

When all recomputed values matches, then the receiver may continue the verification. Next step is to check that all signature element σ_i, are different from each other.

$$\sigma_1 \neq ... \neq \sigma_k \tag{5.6}$$

Thereafter, recompute h by applying function H on concatenation of the message m and counter c to obtain h. Then, apply the function G with function key h on all signature elements σ_i and verify that all resulting values are equal. Thus, creating a k-way collision.

$$G_h(\sigma_1) = ... = G_h(\sigma_k) \tag{5.7}$$

5.1.4 Security

The security level P_f of BiBa signature scheme is defined by the number of hash function operations that an adversary has to perform to forge a signature, knowing at most $r = k$ elements from the private key sk. The minimum number of hash function operations to forge a signature is $2/P_f$.

However, the security level of BiBa signature scheme can be improved in a few ways. One of the possibilities is to increase parameters t and n which are the number of the private key elements and the range of output values of G_h respectively. The second way is to find multiple two-ways collisions and letting all these two-ways collisions be the signature for given message. This will increase the security. However, fewer signatures can sign since more private key elements are revealed with each signature. The last but not least way to improve security is to find one k-ways collisions, where $k > 2$.

Figure 5.1 on Page 79 shows a good example of the security of the signature as well as the probability of finding and forging one. In the figure, there are taken into consideration the probability of finding: one two-way, six two-ways and one 12-way

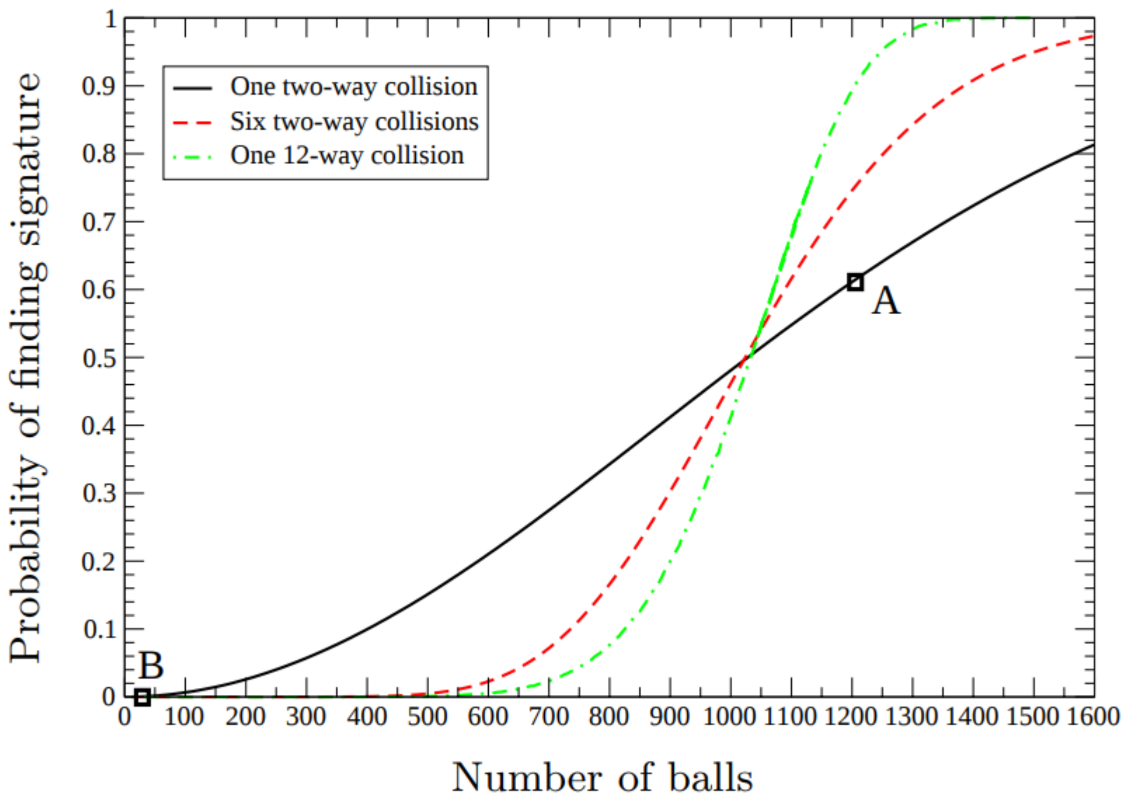


Figure 5.1: The probability of finding a signature when having x SEALs [88].

collision in BiBa signature scheme given x private key elements. As it can be observed in the figure, the signer as well as an adversary, need to know a big amount of private key elements in order to generate/forge a signature.

In one two-way collision the probability of finding a signature start to grow sooner than six two-ways and one 12-ways collision. This method provides a lower security level since it is easier for an adversary to forge a signature faster, with a lower amount of known private key elements.

Six two-ways and one 12-way collision can be seen appropriately. There is a small difference in the number of known private key elements from not being able to find a signature to have a high probability to find one. This is a good attribute of this scheme since even when an attacker has collected larger amount of revealed private key elements from the signature, he will still be unable to find a collision to forge a signature. Therefore, one 12-way signature with $n = 222$ gives better security level than six two-ways signature with $n = 2366050$.

As mentioned at the beginning of this section, BiBa exploits the birthday paradox to achieve efficiency and security. Birthday paradox was described in greater detail in Subsection 2.2.5 on Page 28. BiBa is leveraging the birthday paradox in such a way that it makes collisions more likely to happen. There is known that collisions are rare in hash functions, but there is no doubt that they exist.

The probability to find at least one collision in BiBa signature scheme can be calculated from the following equation:

$$P_c \approx 1 - e^{\frac{t(t+1)}{2n}}. \tag{5.8}$$

Here, t equals to the number of elements in private key and n is output range of G_h. Going back to the allegory of the original author of this scheme, t is the number of balls a signer have, and n is the number of bins that the signer throws these balls into. Then, this equation shows the probability of that at least two balls will land in the same bin.

Whereas the probability for an adversary to forge the signature after one trial is equal to:

$$P_f = \frac{\binom{r}{k}(n-1)^{r-k}}{n^{r-1}}. \tag{5.9}$$

Here, r is the number of the private key element known to the adversary, k is the number of private key elements used for signatures, and n is the output range of G_h.

Important to emphasize is that BiBa signature scheme is assumed secure when the sender does not disclose more than $\gamma = 10\%$ of private key elements. This means that when the sender has parameters $t = 1024$ and $k = 11$, then he is able to sign ten different messages with the same private key. After that, the security of this signature scheme decreases to an unsafe level.

In conclusion, BiBa signature scheme is very adjustable. So that anyone can make it suitable to its preferred use, by simply changing the parameters. However, this may affect the size of both keys and signatures, computational time and or the security level of the whole signature scheme.

5.2 Hash to Obtain Random Subset

Hash to Obtain Random Subset (HORS) is a signature scheme presented in April 2002, by Leonid and Natan Reyzin in [92]. HORS is deeply based on BiBa protocol and can be seen as its direct improvement. The Authors says that HORS maintains all the advantages of BiBa and removes its main disadvantage. Thus, keeping signatures small and fast verification and improves the time required for signing. Moreover, They were able to decrease the size of both private and public keys and signature, at the same time maintaining the same security level.

5.2.1 Key Generation

HORS signature scheme produces both private and public keys on parameter input n, k, t. Where the parameter n describes the security level of the scheme. The parameter k describes the number of elements a message is divided into in the signing process. This parameter also determinates the size of signature and its verification time. Whereas the parameter t describes the number of elements in both private and public key. HORS used two functions F and H. Where F is a length preserving the one-way function of length n, and H is a hash function taking arbitrary length input producing $k \log(t)$ - bit output. The parameters and functions are publicly known.

The private key sk consist of t random generated n-bit string.

$$sk = (sk_1, ..., sk_t) \tag{5.10}$$

To obtain the public key pk, the sender has to hash every private key element sk_i separately.

$$pk = (pk_1, ..., pk_t) = (F(sk_1), ..., F(sk_t)) \tag{5.11}$$

Consequently, the size of each private and public key is equal to tn-bit.

5.2.2 Signing

To create a signature with HORS, the sender needs to hash the message m with the function H to obtain the message digest $\boldsymbol{d} = H(m)$. Then, divide $\boldsymbol{d}$ in k parts and interpret them in base t notation holding values between 0 and $t-1$.

$$\boldsymbol{d} = (d_1, ..., d_k)_t \tag{5.12}$$

Integer values d_i determinate which elements from the private key should be used to sign the message. Then, to generate the signature, collect these private key elements.

$$\sigma = (\sigma_1, ..., \sigma_k), \text{ where } \sigma_i = sk_{d_i} \text{ for } i = 1, ..., k \tag{5.13}$$

No matter the parameters, the signature generation phase requires only one evaluation of the hash function H. The size of the signature is tk-bit.

5.2.3 Verifying

To verify the signature, the receiver has to hash the message m with function H to obtain the message digest $d = H(m)$. Then, slice the message digest d into k chunks, each with a length of $log\ t$. Subsequently, he has to interpret these chunks as integer values in base t notation.

$$d = (d_1, ..., d_k)_t \tag{5.14}$$

This will allow the receiver to pick the right elements from the sender's public key pk. Furthermore, the receiver has to apply the function F on all signature elements σ_i received from the sender. Thereby, by comparing all chosen elements from the sender's public key with the signature elements iterated through the function F, the receiver is able to determinate whether the signature is valid. Supposing, that at least one element is wrong, then the whole signature is not valid and should be rejected.

$$F(\sigma_i) \overset{?}{=} pk_{d_i} \text{ for } i = 1, ..., k \tag{5.15}$$

The verification process is very fast just like the Authors mentioned it in the paper [92]. To verify the validation of the signature, the verifier need only k evaluations of the function F and only one evaluation of hash function H.

5.2.4 Security

HORS signature scheme can be used r times, just like BiBa signature scheme. The parameter r denotes the number of messages that can be signed with one key pair before the scheme became insecure. The value is dependent on the choice of both parameters t and k. Nevertheless, the parameter r need to be a small number, since the security level of the scheme decreases as the value of the parameter r increases, in both algorithms. Furthermore, Authors emphasize that the security of HORS relies only on complexity-theoretic assumptions. Whereas, BiBa relies on the assumption that H is a random oracle, to provide sufficient security.

The authors proved that HORS is Existentially Unforgeable against r-time Chosen Message Attack (EU-CMA) if underlying hash function H is r-subset-resilient and the function F is a one-way function. Furthermore, the security level of HORS can be described by $k(\log t - \log k - \log r)$. Moreover, the paper [92] shows that HORS is both faster and yields better security level than BiBa, despite the same parameters.

For the hash function H used in signing, the authors propose SHA-1 and RIPMED-160, wherein 2017, SHA-1 is broken and is not recommended for use. There exist a collision attack on RIPMED, nevertheless it does not apply to RIPMED-160 [78]. Even that, the SHA-1 was broken, the HORS scheme does not need a collision resistance hash function. However, a better security level will be achieved with SHA-2, BLAKE, BLAKE 2, or SHA-3.

The reason for letting H be a cryptographic hash function is that it should provide the complexity to make it infeasible to find messages number $r+1$ such that $H(m_{r+1}) \subseteq H(m_1) \cup ... \cup H(m_r)$, while providing r signatures.

Furthermore, to improve the security significantly, the parameters k and t should be increased. Especially the parameter t if the meaning is to provide more than one signature. Unluckily, it has to be taken into account that it will make the signature scheme slower and the keys will become larger.

Part III

Many Times Signatures

Chapter 6

Stateful Signature Schemes

This chapter gives an overview of hash-based signatures, which are more practical than all signatures schemes described so far. Signature schemes described in this chapter are stateful many-times hash-based signatures schemes, where the signer may sign several or many messages with one key pair without decreasing the security of the signature. However, due to the statefulness, the signer must keep track of the number of signatures already generated.

6.1 Merkle Signature Scheme

In 1979, when public key cryptography along with hash-based one-time signatures was created. Ralph C. Merkle came up with an idea of how to make one-time signature more practical [79]. This survey ended with a signature scheme which is able to create many signatures by using binary trees as the basic structure combined with one-time signatures.

6.1.1 Reducing the Public Key Size

Earlier in this thesis, in Chapter 4.1.5, we described a method to reduce the memory requirements for storing the private key. Unfortunately, the same method can not be applied to public keys, since they cannot be generated on the fly. However, in 1979, Ralph Merkle patented a structure to fix that problem in [79]. The Merkle tree is a binary tree where a node is a hash value of concatenation of its child nodes. The idea was to use One-Time signatures still to sign the message, then hash their public key and let them be leaf nodes of the Merkle tree. By successfully concatenating sibling nodes and then hashing them, one will generate the whole Merkle tree. The top node of the tree called the root node may now be used as the public key for all One-Time signatures used to generate the tree. Thereby, reducing the public key size to a single hash value.

6.1.2 Structure

Merkle tree is a binary tree used to authenticate multiple One-Time public keys. Thus, give the possibility to verify multiple signatures using only one overall public key. The number of messages to sign is decided by the number of leaf nodes, which again is decided by the height of the tree. The binary tree used in hash-based signatures have to be a perfect binary tree. This means that every leaf nodes need to be on the same level and all nodes except leaf nodes, need to have two child nodes. The signer can freely choose the one-time signature he wants to use. Thereafter, the leaf nodes of the Merkle tree consist of the hashed value of the chosen one-time signature's public key. Then, siblings leaf nodes are concatenated and hashed, where the hashed value is the value of the parent node. This procedure propagates and generates the whole tree. At the top of the tree, we have the root node, which authenticates the entire Merkle tree. The root node is placed at level h of the tree, whereas the leaf nodes are at level 0. On each level, the nodes are indexed from 0 starting from the leftmost node.

The authentication path $Auth$ is a list of sibling nodes from each level of the tree. This list contains h nodes from level 0 up to level $h-1$. The list is sent as a part of the signature so that the receiver will be able to reconstruct the root node, which is the public key of the signature scheme. The notation $Auth_i$ indicates the authentication path for the i'th leaf node, counted from left.

6.1.3 Key Generation

To generate a Merkle tree, the signer must provide the key generation algorithm with two parameters h and n. Both parameters need to be positive integers, where the first parameter tells us the height of the tree as well as how many signatures the scheme will be able to create and verify. Given parameter h, the Merkle signature

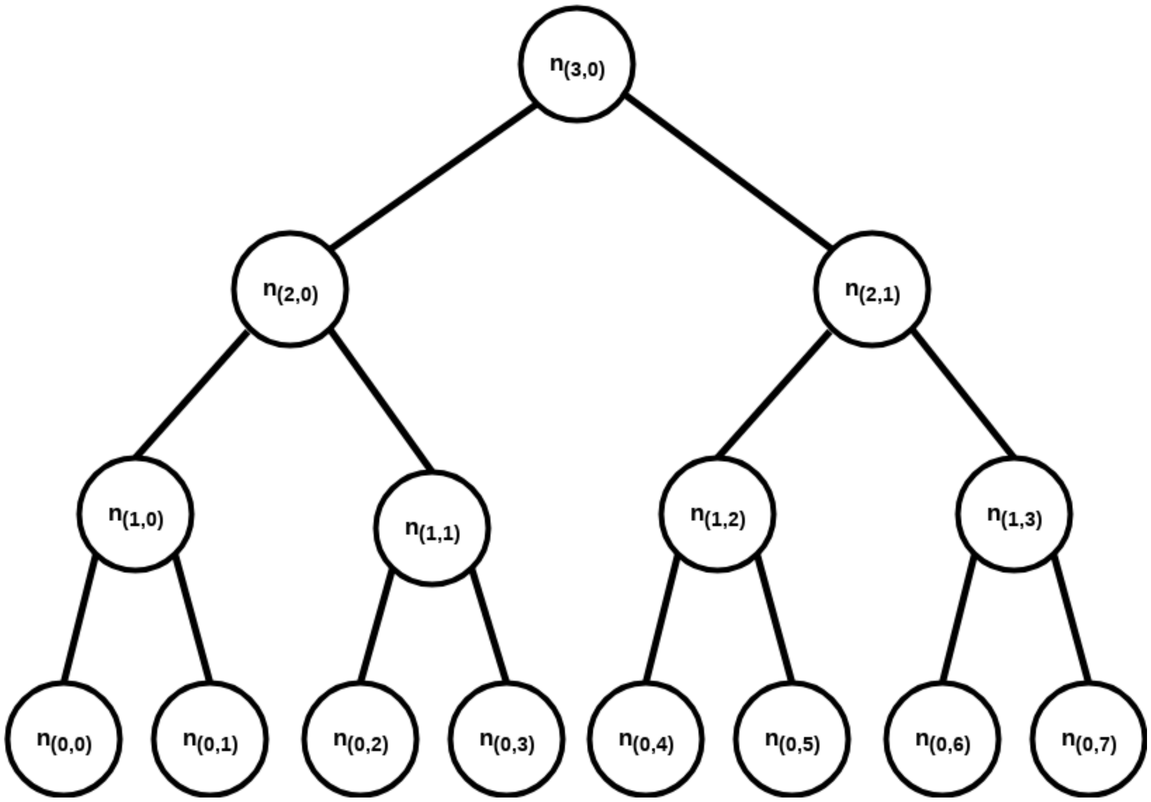


Figure 6.1: Binary tree from Merkle signature scheme with height $h = 3$.

scheme will be able to sign 2^h messages. The second parameter n represents the desired security level in bit.

The private key of Merkle signature scheme, consisting of a series of one-time signatures key pairs. The one-time signature keys in leaf nodes can be freely chosen by the sender, before generating the tree. The sender needs to generate 2^h One-Time key pairs, and store them securely as the private key.

The public key of this signature scheme is the root node of Merkle's tree. The root node consists of a hash value of length n. To be able to compute the public key, the signer needs first to generate the whole tree.

6.1.4 Signing

The signer needs to keep track of how many signatures has been generated already from this scheme. For simplicity, the signatures are generated from the leaf nodes in chronological order. Therefore, the only variable to store is the index variable i, which indicates the index of the leaf node to use to generate the next signature.

To create a signature with Merkle signature scheme, the sender has first to calculate the message digest $d = H(m)$. Secondly, use chosen one-time private key from leaf node at index i to sign the message digest. The index i, the one-time signature σ_{OTS} as well as the one-time public key pk_{OTS} are parts of the signature σ of the Merkle signature scheme. Next, to complete the signature, the authenticate path $Auth_i$ need to be added as the last part in the signature.

The following rule calculates the authentication path:

$$a_j = \begin{cases} n_{(j,\lfloor i/2^j \rfloor - 1)} & \text{if } \lfloor i/2^j \rfloor \ mod\ 2 = 1 \\ n_{(j,\lfloor i/2^j \rfloor + 1)} & \text{otherwise} \end{cases} \tag{6.1}$$

Where, i is the index of chosen leaf node and j goes from 0 to $h-1$. The authentication node from level a_j is the left sibling node whenever the result of $\lfloor i/2^j \rfloor$ is odd, and right sibling node when the result is even.

The complete signature of the Merkle signature scheme consists of following parts:

$$\sigma = (i, \sigma_{OTS}, pk_{OTS}, a_0, ..., a_{h-1}) \tag{6.2}$$

We provide with a concrete example of calculation of the authentication path for better understanding. In this example, we have a Merkle tree of height $h = 3$, which gives us the possibility to generate up to 8 signatures. Then, assume that we use the leaf node on index $i = 5$ so that we need to calculate $Auth_5$. We are using the formula from Equation (6.1) calculate the authentication path. Since the height of the tree is $h = 3$, therefore $Auth_5$ will consist of 3 elements. One authentication node from each level excluding the top level with the root node. Figure 6.1 on Page 88 shows the calculated authentication path on a Merkle tree.

$$a_0 = n_{(0,5-1)} = n_{(0,4)} \mid \left\lfloor 5/2^0 \right\rfloor = 5 \tag{6.3}$$

$$a_1 = n_{(1,2+1)} = n_{(1,3)} \mid \left\lfloor 5/2^1 \right\rfloor = 2 \tag{6.4}$$

$$a_2 = n_{(2,1-1)} = n_{(2,0)} \mid \left\lfloor 5/2^2 \right\rfloor = 1 \tag{6.5}$$

$$Auth_5 = (a_0, a_1, a_2) = n_{(0,4)}, n_{(1,3)}, n_{(2,0)} \tag{6.6}$$

$$\sigma = (5, \sigma_{OTS}, pk_{OTS}, Auth_5) \tag{6.7}$$

6.1.5 Verifying

Given the signature, message and public key of the signature scheme (σ, m, pk), the verification goes as follows. First, verify the message with the one-time signature.

$$vrfy(m, i, \sigma_{OTS}, pk_{OTS}) \stackrel{?}{=} 1 \tag{6.8}$$

Then, only if the one-time signature is accepted, the receiver may proceed to step two. Further, verify the public key by, reconstructing the root node and then compare it with the public key. If the comparison hold, the signature is accepted. Otherwise, the signature is rejected.

To reconstruct the root node (public key), there is a need for sibling nodes, to be able to recompute the parent node. For this purpose, the receiver uses the authentication path from the signature sent by the signer. However, there is a big difference if the node is the left or right child of the parent node. Therefore, the index is a part of the signature to be able to calculate it. The reconstructing process is performed by verification path using nodes from the authentication path. The verification path consists of $h + 1$ nodes, where the first node is set to be the hashed one-time public key.

$$v_0 = H(pk_{OTS}) \tag{6.9}$$

Consequently, the last element of the verification path v_h should be equal to the root node of the Merkle tree. The following formula is used to reconstruct the root node:

$$v_j = \begin{cases} H(a_{j-1}||v_{j-1}) & \text{if } \lfloor i/2^{j-1} \rfloor \ mod \ 2 = 1 \\ H(v_{j-1}||a_{j-1}) & \text{otherwise} \end{cases} \tag{6.10}$$

Thereafter, the last step of the verification process is to compare the last node of the verification path v_h with the actual public key of the Merkle signature scheme.

$$v_h \stackrel{?}{=} pk \tag{6.11}$$

If the last comparison holds, then the signature can be trusted. Otherwise, the signature should be rejected.

For better understanding, we provide with a small example, showing the reconstruction of the root node. For this purpose, we use the authentication path from the previous example as well as the same tree. So that, we are going to reconstruct the

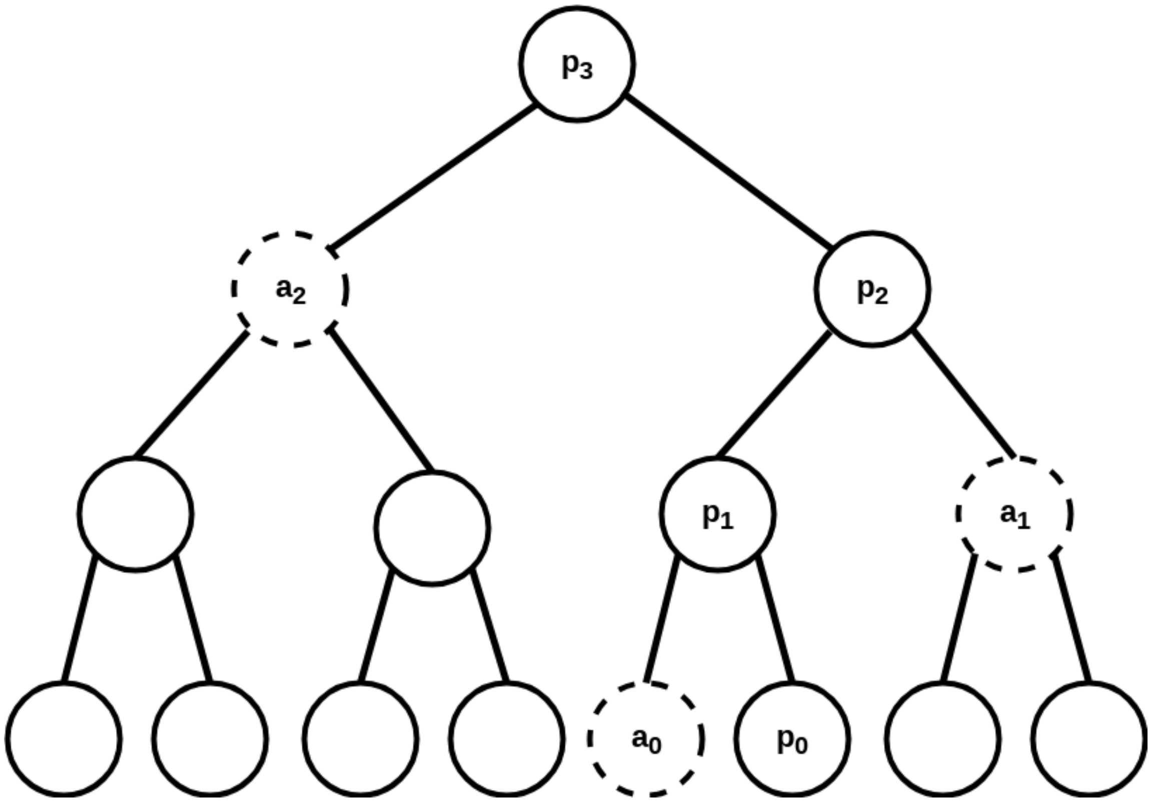


Figure 6.2: Authentication path and verification of signature from index $i = 5$ with tree height $h = 3$.

public key of the Merkle tree with height $h = 3$ and index $i = 5$. Thus, verifying the signature from that index.

$$(5, \sigma_{OTS}, pk_{OTS}, a_0, a_1, a_2) \tag{6.12}$$

Assuming that the one-time signature verification has been accepted, the receiver can now proceed to the verification path. By hashing the one-time public key, the receiver will obtain one of the leaf nodes of the Merkle tree, which is set to be the first verification path node.

$$v_0 = n_{(0,5)} = H(pk_{OTS}) \tag{6.13}$$

Further calculations are performed due to the formula from Equation 6.10.

$$v_1 = H(a_0 \mid\mid v_0) \mid \lfloor 5/2^0 \rfloor = 5 \tag{6.14}$$

$$v_2 = H(v_1 \mid\mid a_1) \mid \left\lfloor 5/2^1 \right\rfloor = 2 \tag{6.15}$$

$$v_3 = H(a_2 \mid\mid v_2) \mid \left\lfloor 5/2^2 \right\rfloor = 1 \tag{6.16}$$

The verification path node v_3 is the last node of verification of Merkle tree with height $h = 3$. Thus, if the authentication path was not altered in transit, then the v_3 node should now be equal to the root node of the Merkle tree.

$$v_3 \overset{?}{=} pk \tag{6.17}$$

To emphasize, only when both the one-time signature and the recomputed root node are verified and holds, the signature can be accepted and trusted.

6.1.6 Security

The security of the Merkle signature scheme is based on the security of hash functions as well as the security of the one-time signature used. Therefore, as long as the underlying one-time signature is secure, the whole scheme depends on the security of hash function used in the scheme.

6.2 Making MSS More Practical

The time required for signature generation rely mostly on time spend on the computation of the authentication path. Therefore, it is crucial to have efficient tree traversal algorithm so that signature schemes based on Merkle tree might compete with the signature time of RSA or ECDSA. Secondly, the number of signatures should be increased for different applications, without consuming too much of memory. Below, a summary of work regarding these aspects is presented.

6.2.1 Traversal algorithm

In 2004, M. Szydlo described a traversal algorithm for Merkle trees making it more practical to use [102]. The method presented has improved the efficiency due to time and space cost, compared to the original paper of Merkle [79].

Particularly, the algorithm produces an authentication path for a binary tree with height h in time $2h$ and space less than $3h$. Where, time is interpreted as hash function evaluations, whereas space is the number of node values stored.

This paper shows an improvement over all previous results, which measured cost by multiplying space and time. Moreover, Szydlo proved that the complexity of the algorithm is optimal.

6.2.2 CMSS

A preliminary version CMSS first appeared in L.C.Cornado Ph.D. thesis [27] under the supervision of J. Buchmann. Then, the paper by Buchmann et al. [19] presents the full version of the algorithm as well as a test result of implementation.

CMSS is an improved version of the Merkle Signature Scheme with traversal improvements from Szydlo [102] and ideas from L.C.Cornado Ph.D. thesis [27].

CMSS is very similar to MSS but uses a slightly different principle. Instead of using one big binary tree, where the root is the public key, and the leaves contain WOTS to sign messages. The CMSS signature schemes use tree chaining, which use many instances of MSS trees. On the higher layers, leaves of the main MSS tree are used to sign roots of lower layers MSS trees with WOTS. Whereas, the leaves of MSS trees on lower layers are used to sign the messages also with WOTS. CMSS uses the same parameters of WOTS on each layer.

Both the main tree and subtrees have the same height h. This change has made it possible to increase the signature capacity from 2^{20} to 2^{40}. CMSS can sign $n = 2^{2h}$ messages for any h. However, when n goes over 2^{40}, the key pair generation time is no longer practical.

Furthermore, CMSS reduces the private key by storing only the seeds for the PRNG, and then the private keys are generated when needed. This makes the key generation dynamic and reduced its time of generation. The private key is updated after every signature generation, to keep the private key small and to make CMSS forward secure.

The public key of CMSS is the root node value of the main MSS tree.

In addition, CMSS uses Szydlo's algorithm from [102] to reduces signature generation time.

6.2.3 GMSS

In 2007, a generation Merkle Signature Scheme was introduced by J. Buchmann et al. in [17]. GMSS is an improved version of CMSS which allows great flexibility. The user may choose the signature capacity and possibility to adjust the signature generation and verification time, as well as the signature size. GMSS is able to provide with cryptographically unlimited (2^{80}) number of signatures while maintaining the efficiency.

GMSS reduces the signature size as well as the signature generation cost. A more efficient generation of trees during the signing process was achieved by storing some seeds as a part of the private key. At the same time, GMSS reduces the signing time by distributing the cost for one signature generation across several previous signatures and the key generation. The authors have also improved the worst case scenario of Szydlo's traversal algorithm from [102] by using a scheduling strategy to pre-compute upcoming signatures.

The structure of the scheme has changed. GMSS use multi-layer of MSS instances where each of them may have different heights. In the leaves, a Witernitz one-time signature [79] is used, which may have a different parameter for memory and time trade-off in each MSS instance.

The paper introduces a parameter set P which the GMSS algorithm is initialized with.

$$P_h = (T, (h_1, ..., h_T), (w_1, ..., w_T)) \tag{6.18}$$

Where, T is the number of layers of MSS instances, h_i is the height of the MSS tree on layer i, w_i is the WOTS parameter for signing the root of the MSS subtree, whereas w_T is used to sign the messages on the lowest layer of MSS tree. All h_i together is the height h of the whole GMSS multi-layer tree. The parameter set P_h allows up to 2^h signatures. Szydlo's algorithm is used to compute the optimal parameter sets. By begin able to adjust the parameter set P for each layer of MSS trees, the signature size in GMSS is reduced by more than 26% compared to its predecessors. CMSS can be seen as a special case of GMSS with parameter set $P_h = (2, (h_1, h_1), (w, w))$.

The GMSS signature consists of several parts. At first, the index of the used leave node. Then, the one-time signature of the message. Furthermore, the one-time signature of all intermediate MSS trees as well as the authentication path for all used trees from the signing leaf node to the root of the MSS tree at the highest layer. The parameter set P has to be shared with the receiver to be able to verify a signature. Beside that, the verification process is similar to the previous Merkle based signature schemes.

6.2.4 Merkle Tree Traversal Revisited

In 2008, J. Buchmann et al. [18] presents a Merkle tree traversal algorithm that improved the traversal algorithm of Szydlo [102] by reducing the signing time even more. Until this time, Szydlo's algorithm was the fastest tree traversal algorithm. However, Szydlo's algorithm had a disadvantage of having a big difference in time between its average and worst case. This makes the computation of the authentication path uncertain with respect to runtime.

The authors in [18] improved both the worst and average case runtime of the algorithm, as well as they, reduced the distance in runtime, of these two cases, closer to each other.

The idea of this algorithm was to distinguish leaves nodes from the inner node and then to try to reduce the number of leave nodes since they are extremely more expensive to calculate. In practice, the average case of the algorithm computes $(h-1)/2$ leaves and $(h-3)/2$ internal nodes. While the worst case computes $h/2$ leaves and $3/2(h-3)+1$ internal nodes.

In MSS trees of height $h = 20$, this algorithm was able to reduce the hash function evaluation by 15% compared to Szydlo's algorithm. When it comes to the memory requirement, the computations need $3.5H - 4$ nodes of memory.

6.2.5 Reducing Security Assumptions in Merkle Trees

In 2008, E.Dahmen et al. [31] published a paper where they relaxed the security requirements of the hash function used in Merkle's authentication tree. They proved that by slightly changing the construction of MSS, they succeeded to get even superior security using second preimage resistant hash functions of equal n-bit output.

The main difference was made in the authentication tree construction. Where the authors apply an XOR bit masks on the child nodes before they get concatenated and hashed to obtain the parent node. Additionally, the leave nodes are not the hashed values of OTS public keys but the OTS public key bit strings themselves. These constructional changes were enough to relax the security requirement from collision resistant to second-preimage resistance hash functions. Consequently, the new proposed scheme is no longer prone to birthday and collisions attack.

Unfortunately, the public key of the new proposed scheme got bigger since it needs additionally to contain all the bit masks and the key for the hash function as well. However, the signature size got significantly shorter. Even though for example in certificates the public key is a part of the signature as well, the size of the signature is still significantly shorter than the original scheme.

6.3 XMSS Family

The first XMSS scheme was introduced in 2011 by Buchmann et al. [16]. During the years several updates with improvements and versions of the scheme for specific application has been proposed. In 2015, Merkle hash-based signature was proposed for standardization by McGrew and Curcio [76]. Following with updates from Hülsing et al. [54] improving the internet draft. The standardization process, as in 2018, is still ongoing and the draft is updated periodically. In June 2018, the 12'th version of XMSS was published as RFC8391 [51].

6.3.1 XMSS

In 2011, Buchman et al. [16] proposed new multiple time signature called eXtendet Merkle Signature Scheme (XMSS). This was the first practical forward-secure signature scheme with minimal security requirements. XMSS is based on the ideas of MSS and GMSS, both described in Section 6.1 and Subsection 6.2.3 respectively. The scheme was created to sign a fixed length message. However, Target Collision Resistant (TRC) can be used in order to sign longer messages. Since it is proven that a TRC function can be created from a one-way function [95], therefore the scheme still holds the minimal security assumptions.

The Authors in [17] have emphasized that it is impractical to make big Merkle trees. Therefore, to hold the efficiency, the authors of XMSS have created the eXtendet Merkle Signature Scheme to have the height of $h = 20$, which can sign up to 2^{20} messages.

The structure of XMSS is similar to MSS-SPR [31] (described above in Subsection 6.2.5), which was the most efficient stateful Merkle based signature scheme at this time. Both schemes utilize bitmask in node creations. However, XMSS differ from previous schemes in leaves creation. The one-time public key elements are stored in another XMSS tree. These trees holding one-time public keys are called L-trees. Since the one-time public key elements might not necessarily be a power of 2. Therefore, when a leave node in an L-tree does not has a right sibling, then the node is lifted to a higher level until it becomes the right siblings of another node. To fit all l one-time public key elements, the L-trees consist of height $\lceil \log l \rceil$. Therefore, in total there is need for $h + \lceil \log l \rceil$ bitmasks to cover all layers.

The one-time signature used in XMSS is W-OTS^{PRF} [20] described in Subsection 4.4.1 on Page 63 in this thesis. W-OTS^{PRF} utilizes only pseudorandom functions, whereas XMSS utilizes a hash function family H with second preimage resistant and a function family F with the pseudorandom property. This makes the scheme hold the minimum security assumption. The function H is used to construct the nodes, whereas the function F is used to make the XMSS a key evolving scheme. Given a seed to the function F it will produce two values. The first value is a seed to generate a one-time key pair, and the second value is the seed for the function F in the next signature generation. Thus XMSS is a forward-secure signature scheme. Moreover, the authors have proven that XMSS is Existentially Unforgeable under adaptive Chosen Message Attack (EU-CMA) in the standard model.

The XMSS signature size is 1/4 of the signature size of MSS-SPR signature size and have the signing and verifying speed competitive to RSA and DSA.

6.3.2 XMSS$^+$

In 2012, XMSS$^+$ was proposed by Hülsing et al. [53] XMSS$^+$ was specially optimized for use on constrained devices like for example smart cards. The XMSS$^+$ uses tree chaining idea from Subsection 6.2.2, the distributed signature generation from Subsection 6.2.3 and it utilize the W-OTS^{PRF} One-Time Signature.

Particularly, XMSS$^+$ uses two layers of XMSS trees. The lower layer of XMSS trees signs the messages with a One-Time signature from the leaves. Whereas the top level XMSS tree signs with One-Time signature the roots of lower layer XMSS trees. The root node of the top XMSS tree is the overall public key for all signatures made with this instance of the scheme. Every instance of XMSS tree uses the same bitmasks. Therefore the public key size is reduced since it contains fewer bitmasks.

The height of XMSS$^+$ remains the same as in original XMSS scheme. However, the

XMSS tree is divided into two layers, where each layer contains XMSS trees of height $h_j = 10$. The overall height of XMSS$^+$ scheme is $(h = h_1 + h_2 = 10 + 10 = 20)$ the sum of one XMSS tree from each layer.

This structural change was crucial for the increase in performance of the new scheme. Since the change made it possible to only generate the top level XMSS tree and the most left XMSS tree on the lower layer. Since when first lower layer XMSS tree runs out of leaves to sign the messages, the second XMSS tree on the lower layer is created and signed by the second leave of the XMSS tree on the higher layer. Causing the reduces in key generation time for 2^h signatures from $O(2^h)$ to $O(\sqrt{2^{h/2}})$. Additionally, this makes it practical to generate keys on small devices like smart cards with was never achieved before when it comes to hash-based signatures. Moreover, the XMSS$^+$ preserves the strong security guarantees of XMSS scheme.

The second improvement made in XMSS$^+$ was to use the unused updated in BDS algorithm to compute the signatures of the next XMSS root on the lower layer. This reduces the worst case signing time for the scheme. Thus making the scheme competitive to RSA and ECDSA.

The XMSS$^+$ is a forward secure signature scheme and EU-CMA secure since it holds the same security assumptions as XMSS.

XMSS and XMSS$^+$ with W-OTS$^+$

In 2013, Hülsing alone published a paper [52] presenting a new One-Time Signature called W-OTS$^+$ (fully described in Subsection 4.4.2 on Page 66). In the paper, the author provided a comparison of both XMSS and XMSS$^+$ schemes using W-OTS and W-OTS$^+$ as underlying One-Time Signatures.

In both W-OTS and W-OTS$^+$ increasing the parameter, w gives shorter signatures. However, in W-OTS the increasing in the parameter w yielded lower security level, making some of the parameters set impractical to use.

Switching the underlying one-time signature in both XMSS and XMSS$^+$ to W-OTS$^+$ remains the same timing and size of the scheme but providing significantly higher security level of the scheme.

To match the same security level in XMSS$^+$ with W-OTS and XMSS$^+$ with W-OTS$^+$. For some parameters, the second yields signatures more than 50% smaller than the previous signature scheme. Considering that signature sizes is the main drawback of hash-based signatures, this is enormous progress.

6.3.3 XMSS^{MT}

In 2013, Hülsing et al. [55] published new version of XMSS signature scheme called XMSS^{MT}. The new scheme applies the tree chaining idea (first mentioned in [19] described in 6.2.2) to XMSS scheme, combined with the improvements of BDS algorithm from XMSS^{+} publication [53].

The main structural change of the scheme is that XMSS^{MT} is a multi layer version of XMSS. In contrast to XMSS which is only one layer with one XMSS tree, and XMSS^{+} which has two layers with XMSS tree instances. This makes it possible to generate virtually unlimited number (2^{80}) of signatures. XMSSMT may contain different height XMSS tree on each layer. Each XMSS tree contains one-time signatures in the leave nodes to sign. The XMSS trees in the lowest layer are used to sign a message, whereas the XMSS tree on higher layers are used to sign the root of lower layer XMSS trees. The XMSS^{MT} scheme utilize W-OTS^{PRF} underlying one-time signature.

The signature of XMSS^{MT} scheme contains the index of the leave node used to sign the message as well as the signature and authentication path for the XMSS tree from all d layers.

$$\Sigma = (i, \sigma_0, \mathrm{Auth}_0, \sigma_1, \mathrm{Auth}_1, ..., \sigma_d, \mathrm{Auth}_d) \tag{6.19}$$

XMSS^{MT} comes with several trade-offs [55].

- The BDS parameter k_i: trade-off between signature time and private key size.
- Winternitz parameter w_i: trade-off between runtimes and signature size.
- Layers d: trade-off between both key generation and signature time and signature size.
- XMSS tree height h_i: influence the security of the scheme as well as the performance.

For these trade-offs the Authors have presented an linear optimization model which gives the best parameters for maximum benefits. This leads faster key and signature generation times compared to previous schemes.

The XMSS^{MT} scheme highly adjustable by the parameters and can meet the requirements of any case for digital signatures, as well as it is forward and EU-CMA secure.

6.3.4 XMSS-T

In 2016, Hülsing et al. [57] introduced new both many-time signature XMSS-T as well as the underlying one-time signature WOST-T. The second was described in Section 4.4.3 on Page 69.

The primary goal of this paper was to mitigate the multi-target attack on hash-based signatures. Since, as mentioned in Subsection 4.4.3 about WOST-T, the multi-target attack downgrades the complexity of breaking a hash function, reducing the security level of all hash-based signature schemes. XMSS-T was created to tighten the security of XMSS based schemes.

The new signature scheme XMSS-T is based on XMSS^{MT} with some constructional changes. Therefore XMSS-T is also a hypertree consisting of d layers, where each layer is of equal height. The Winternitz parameter w in the underlying one-time signature WOTS-T control the space-time trade-off.

The most significant difference in the structure was the introduction of addressing scheme. Such scheme, given a structure and an index of a substructure create a unique address for the substructure within an XMSS-T key pair. The unique addresses use them to generate functions keys and bitmasks. The addressing scheme is light in

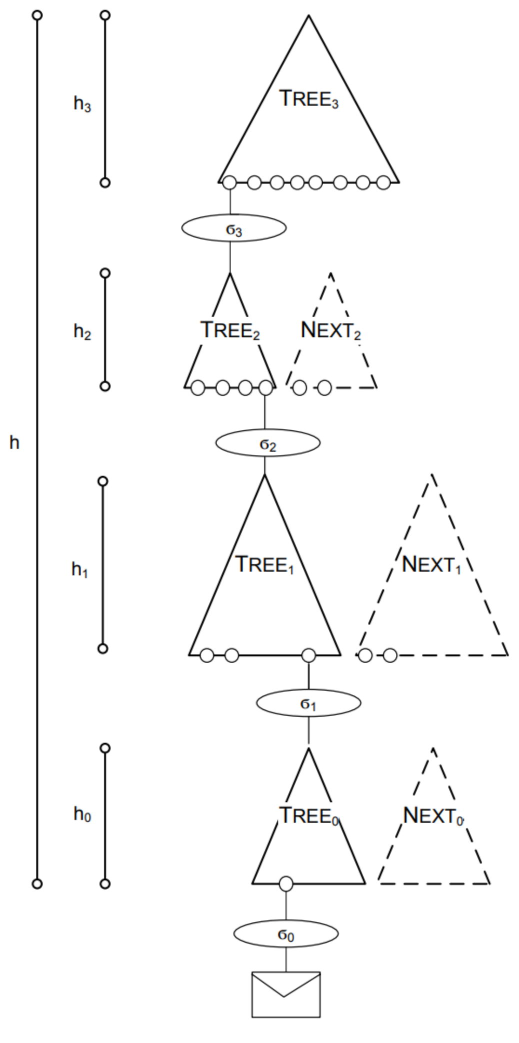


Figure 6.3: Structure of XMSS^{MT} many-time signature scheme [55].

calculations, and it uses only the information that is known when a hash function is called. The addressing scheme is publicly known and can be reused in all XMSS-T key pairs.

Moreover, the Authors changed the node construction in the tree. In XMSS-T the node calculation is done by concatenating the child values, then XOR the result with a bitmask, subsequently, hash the result with a keyed hash function. Both functions keys and bitmasks are different for every hash function call, due to the addressing scheme.

Additionally, the private key consists of two secret values. The first, as before, is a secret seed for OTS key generation. Whereas, the second value is used to generate a randomization element to randomize the message digest of a message while signing. The randomized element is set as part of the signature.

All these together mitigate the multi-target attack on the XMSS-T signature scheme. However, the tightened security comes with a drawback in increased runtimes of the scheme of a factor less than 3.

The scheme has been proven to be EU-CMA in the standard model. Moreover, the XMSS-T scheme replaced the XMSS scheme in the standardization draft and later on was published as RFC8391 [51].

Chapter 7

Stateless Signature Schemes

All digital signature schemes described so far have the drawback of being stateful and therefore force the signer to keep a record of the exact number of previously signed messages. Moreover, any error in this record will result in the insecurity of the whole scheme. Furthermore, the stateful signature schemes can only produce a limited number of signatures. The number of signatures can be increased, even to the point of being effectively unlimited. However, this also increases the signature size. In this chapter, we go one step further and describe the stateless signature schemes.

7.1 SPHINCS

In 2015, Bernstein et al. [9] presented SPHINCS, a post-quantum stateless hash-based digital signature scheme. Setting enormous progress in making hash-based signatures more practical. Unlike all previous hash-based signature schemes described so far, SPHINCS is stateless allowing it to be a drop-in replacement for current signature schemes.

Structurally, the SPHINCS signature scheme is very similar to the XMSS^{MT} stateful signature schemes described in the previous chapter. However, the crucial change, changing the scheme from stateful to stateless, happens in the lowest layer of the structure. The One-Time signature W-OTS$^+$ is used no more at the lowest layer to sign the randomized message digest. Instead, the Authors exchanged it with an improved version of Few-Time signature HORS, namely HORST. The Few-Time signature HORST adds a Merkle tree to the old HORS scheme. Thereby, reducing HORS's public key size from tn-bit to only n-bit in HORST.

Similarly to both XMSS tree and L-tree, HORST uses unique bitmasks on each level of the tree while calculating the nodes. Nevertheless, HORST needs to include the authentication path for every secret key used in the HORST signature in the overall SPHINCS signature. So that the HORST public key can be computed from the HORST signature. These changes give significant improvements in the overall SPHINCS signature size. However, the signature size of SPHINCS is still significantly bigger than the signature size of stateful schemes.

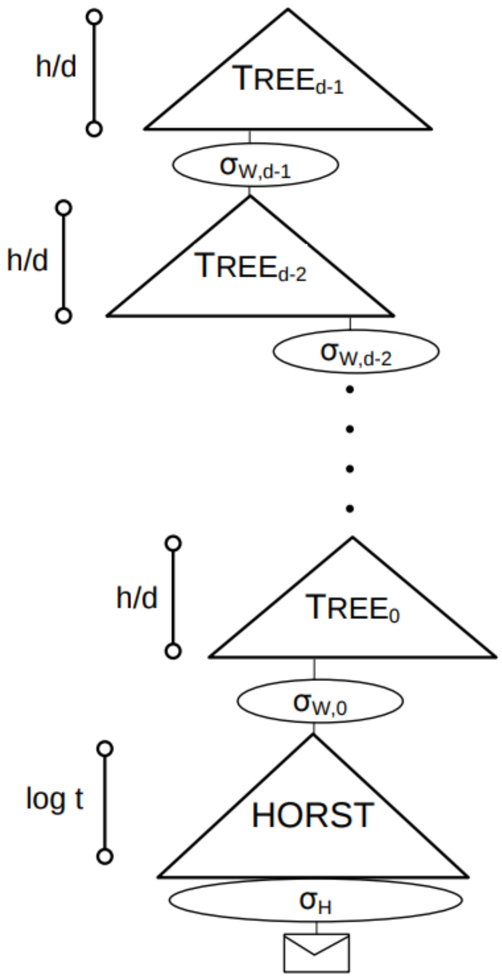


Figure 7.1: Structure of SPHINCS stateless many-time signature scheme [9].

By the fact that SPHINCS is stateless, there are no updates on the private key after key generation phase, and no index stored indicating which leaves was already used. Since HORST is a Few-Time signature, the same key pair can be used to sign several messages. SPHINCS is relying on HORST as the signature scheme to sign the message, having one instance of HORST in each leave node. While signing, the choice of the signing leave node is performed pseudo-randomly. The threshold for using the same HORST to sign a message is r-times. For a secure instance of SPHINCS giving the possibility of securely signing 2^{50}, messages are $r = 8$. Such instance of SPHINCS contains 2^{60} HORST key pairs. SPHINCS base its security on the fact that the probability of using one HORST key pair more than 8 times during 2^{50} signature generations is less than 2^{-128}. Yielding strong long-term security. The Authors have proven that SPHINCS is EU-CMA in the standard model.

In 2016, Hülsing et al. [56] have successfully implemented SPHINCS on a microcontroller. Proving that it was feasible both in term of performance and memory usage. The Authors have as well implemented the stateful XMSS^{MT} signature scheme for comparison, to emphasize the implication of removing the state from a signature scheme. Concluding that the verification if fast in both schemes, however, SPHINCS is 30 times slower on signature generation. Making SPHINCS not the best candidate for an application (on a microprocessor) where the need to generate signatures is frequent.

7.2 SPHINCS$^+$

In 2017, SPHINCS$^+$ was published by Bernstein et al. [8]. SPHINCS$^+$ is an improved version of the original SPHINCS, and in high level view they might look the same. However, the scheme got improved in terms of performance, signature size and security. Moreover, the whole subroutine of few-time signature was changed from using HORST to the new few-time signature FORS.

SPHINCS$^+$ includes the multi-target attack protection presented by Hülsing el al. in [58]. Namely, generation of new function keys and bitmask for every hash function call. To get an abstraction, the Authors introduced the notion of tweakable hash functions. A such function in addition to input value takes a public seed and an address.

The second structural change is the compression of one-time W-OTS$^+$ public keys. Instead of using L-trees as in previous schemes, the one-time public keys are compressed using the tweakable hash function. On input of public seed, the address of one-time signature and the last iterations of W-OTS$^+$ chain, the tweakable hash function creates a short n-bit one-time public key. Consequently, shorter the signature size.

7.2.1 FORS

Forest Of Random Subset (FORS) is an improved version of few-time signature HORST. FORS differ from HORST in a structurally manner. HORST was based on a single tree, whereas FORS consist of k trees of height $\log(t)$. Then to obtain the public key, the tweakable hash function is used on input of a public seed, the address of the structure within SPHINCS key pair and the concatenation of k root nodes of trees. The leaves of the trees contains the hashed value of t random secret key values.

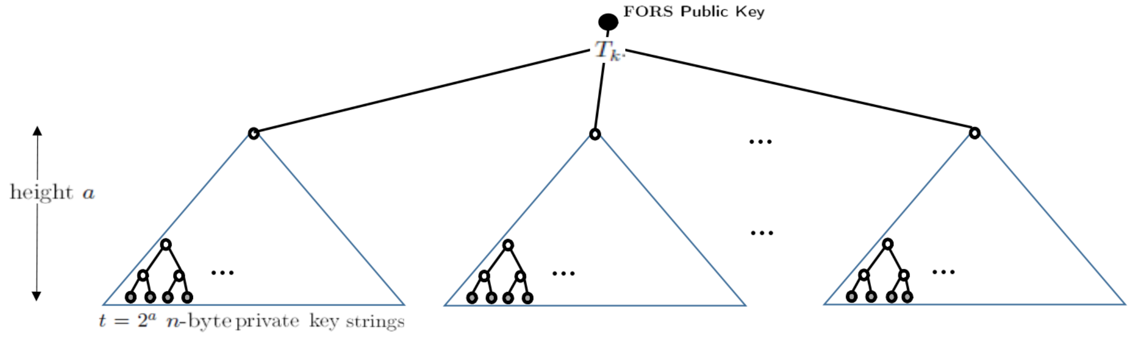


Figure 7.2: Structure of FORS few-time signature scheme [8].

While signing only one internal tree of FORS is used to sign a message. Which one to choose id decided by the pseudorandom value, public seed and the message together. Iterating these values through the tweakable hash function produces the randomized message digest. In addition, the last bits are used as the index indicating which tree within FORS to use. Thus, the index can be omitted from the signature, again reducing its size. The public seed is a part of the public key and the pseudorandom value is included to the signature. Consequently, a message gets linked to a concrete FORS instance and will not work for any other FORS instances. The last provides multi-target attack resilience of the scheme.

This new structure of FORS improved the security of the the scheme, therefore the parameters can be choose differently resulting in shorter and faster signatures.

7.3 Gravity-SPHINCS

Gravity-SPHINCS was published by Aumasson and Endignoux [2] in 2017. This signature scheme is an improved version of the stateless signature scheme SPHINCS. Both SPHINCS$^+$ [8] and Gravity-SPHINCS was published at the same time.

The Authors of Gravity-SPHINCS have implemented several changes to the scheme to shorten the signature size while maintaining the desired security level. Gravity-SPHINCS optimize the original scheme by the following:

- New few-time signature scheme PORS exchanged HORS. A more complex scheme, however more secure.
- Octopus: optimized multi-authentication in Merkle trees, eliminates redundancies from the authentication path.
- The secret key caching: the Authors changed the height of the top level Merkle tree to make it bigger and caching part of it in the private key. This change result in shorter signatures and slightly increased private key.
- Non-masked hashing: the XOR calculation with bitmasks when generating nodes is taken away to simplify the signature scheme. However, then the signature scheme again relies on collision-resistant hash function.
- Batch signing: the possibility of signing several messages at once. Thus, reducing both signature time and size.

The overall signature scheme has the public key consisting of 32-bytes and the private key consisting of 64-bytes.

7.3.1 PORS

The few-time signature scheme got exchanged from HORS to PORS. Gravity-SPHINCS used PRNG to Obtain a Random Subset (PORS) to increase the security level significantly. This is a more complex scheme than HORS. However, the computational overhead is minimal.

PORS uses the message and a salt to generate a random seed for PRNG to generate k distinct values, as well as the index of the Gravity-SPHINCS leaf. Thus, keeping both the verifier and attacker to the specific instance of PORS, eliminating multi-target attacks.

Gravity-SPHINCS used the same optimization of reducing the public key as HORS by utilizing a hyper-tree creating PORST. In additions, PORST uses the octopus algorithm optimization when calculating the authentication path.

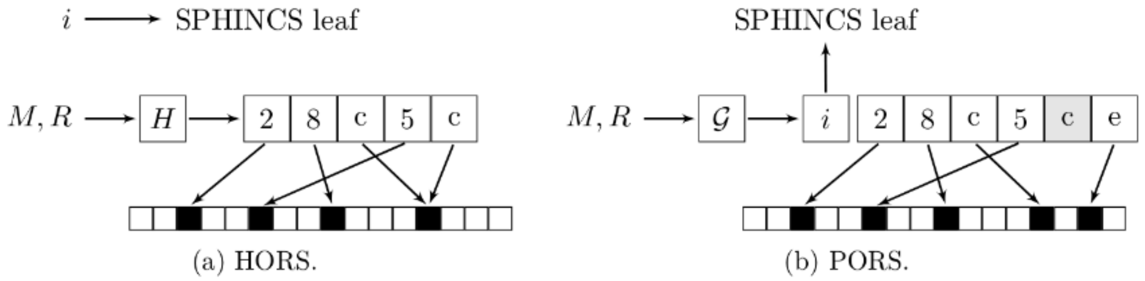


Figure 7.3: Comparison of the structures of HORS and PORS few-time signature scheme [2].

Chapter 8

Analysis and Discussion

8.1 Standardization of hash based signatures

In 2016 NIST called for proposal of algorithm which are believed to be quantum resilient with a deadline one year later in November 2017. In January 2018, NIST have published the result of the first round. There were 82 algorithms proposed, for encryption, key exchange and digital signatures. Where 23 of them was signature schemes and only four of them was hash based signature schemes (but only two of them was published). In April 2018, it has been arranged a workshop where submitter presented their solutions. This will be followed by a three to five year phase of analysis with report about findings. Then, two years later, between 2023 and 2025 a standardization draft for post quantum cryptography will we ready [81]. NIST emphasizes that this is not a competition and that several candidates can be approved for a single purpose/application.

The stateful signature scheme XMSS is also on its standardization way. Current version of this internet draft is updated to an RFC8391 [51].

8.2 Stateful versus Stateless

Both stateful and stateless signature scheme have their advantages and disadvantages.

The main drawback with stateful signature schemes is that they need to keep an internal state which needs to be updated each time a signature is created. Additionally, it requires extra overhead in terms of memory. The next problem with stateful schemes is system restore. It is possible for a stateful scheme to restore itself to an earlier internal state, meaning that a one-time signature from the leave might be used twice, destroying the security of the whole scheme. In other words, signature forgery becomes easier. A similar scenario may happen with backups when key synchronization takes place.

Due to optimization of stateful schemes, it is impractical to generate several signatures in parallel. This is a considerable disadvantage. For example, in a scenario where a stateful scheme is used for server authentication, where the same key might be used for signing of several connections in parallel.

It does not mean that one should avoid using stateful schemes. However, the user needs to be aware of this risk and carefully manage the state to prevent forgery. On the positive site, stateful signature schemes are usually faster with shorter signature sizes. Such an advantage will undoubtedly find its applications to fit.

When it comes to stateless signature schemes none of the aforementioned problems exist. With a stateless scheme one can freely sign messages without compromising in security. In a stateless hash-based scheme, one leaf node can sign several messages. However, this comes with a cost in performance. Stateless schemes are usually slower than stateful schemes and consist of larger signature sizes.

Taking into consideration hash-based schemes, in both stateful and stateless schemes the number of signatures is fixed. However, both the schemes described in this thesis have raised that number to be virtually unlimited.

8.3 Comparison of Hash-based Signatures

Over the last 40 years, we have seen an enormous improvement in hash-based schemes. As the technology evolves and the universal quantum computers become more practical. The development of new structures and hash-based algorithms got significantly more attention. In the call for algorithms by NIST, two stateless hash-based digital signatures were published. In addition, two stateful hash-based signature schemes are on theirs standardization way as well.

The security assumption behind hash-base schemes is well understood even in quantum cases. This gives a little advantage over other proposed schemes which are more complicated. Thus, more challenging to understand and implement.

Hash-based digital signatures have been proven to be practical in different scenarios. Back in 2002, Perring [88] proposed a new protocol for broadcast authentication built on BiBa a hash-based signature scheme. Next, in 2006, Buchmann et al. [19] showed that hash-based signature schemes could be effectively implemented for email signing or X.509 certificates generation with competitive times to RSA and ECDSA. Then in 2007, Buchmann et al. [17] presented an efficient implementation of SSL/TLS using GMSS, begin resilient to DoS attacks and minimize the latency of signature exchange. Moreover, yet another milestone was achieved in 2012 by Hülsing [53]. An efficient implementation of forward secure hash-based signatures on a smart card. Along with on-card key generation.

However, despite the progress of these schemes, the main drawback still appears. Particularly, the trade-off between speed and signature size. When signatures size shrinks, the speed of the signature decreases and vice versa. In stateless signature schemes, the drawback is even more noticeable. However, as long as the application using hash-based signatures can tolerate at least one of the two, then efficient solutions exist.

Table 8.1 on Page 110 presents a comparison of hash-based digital signature schemes with signature schemes used nowadays. The presented schemes are:

- two stateless signature schemes (SPHINCS$^+$-256 and Gravity-SPHINCS),
- one stateful signature scheme (XMSS which also includes a multi-version of itself, the XMSS^{MT}),
- and two currently used standard signature schemes (RSA-3072 and ECDSA(P-256)).

	Public key	Secret key	Signature	Classical security	Quantum security	Signature capacity
SPHINCS$^+$-256	64	128	29 792 / 49 216	256	128	2^{64}
Gravity-SPHINCS	32	64	22 304 / 35 168	128	128	2^{64}
XMSS (XMSS^{MT})	64	132	4 964	256	128	2^{60}
RSA-3072	384	1 728	384	256	0	Unlimited
ECDSA (P-256)	64	96	64	256	0	Unlimited

Table 8.1: Comparison of hash-based signature schemes with current standards. The sizes are provided in bytes.

Instance of SPHINCS$^+$	Key generation	Signature generation	Verification
SHAKE256-256s	1 210 939 356	13 842 403 104	20 889 204
SHAKE256-256f	75 031 996	1 664 510 764	41 469 276
SHA-256-256s	1 095 050 628	12 893 347 756	19 141 296
SHA-256-256f	68 819 608	1 558 148 364	41 469 276

Table 8.2: Instances of SPHINCS$^+$ providing 128-bit quantum security [8]. Provided values are the number of CPU cycles.

Gravity-SPHINCS	Key generation	Signature generation	Verification
small	781 646 000	9 548 000	79 333
fast	24 229 712 000	14 748 000	238 000
NIST-fast	11 789 080 000	16 905 333	337 333

Table 8.3: CPU cycles for Gravity-SPHINCS sub algorithms [2].

XMSS^{MT}	Signature size	Key generation	Signature generation	Verification	Signature capacity
SHA2_20/2_256	4 963	2 476 032	7 227	2 298	2^{20}
SHA2_20/4_256	9 251	154 752	4 170	4 576	2^{20}
SHA2_40/2_256	5 605	$2,535 * 10^6$	12 417	2 318	2^{40}
SHA2_40/4_256	9 893	4 952 064	7 227	4 596	2^{40}
SHA2_40/8_256	18 469	309 504	4 170	9 152	2^{40}
SHA2_60/3_256	8 392	$3,803 * 10^6$	13 417	3 477	2^{60}
SHA2_60/3_256	14 824	7 428 096	7 227	6 894	2^{60}
SHA2_60/12_256	27 688	464 256	4 170	13 728	2^{60}

Table 8.4: Instances of XMSS^{MT} signature scheme. Size in bytes and times in hash function iteration [51].

Part IV

Summary

Chapter 9

Conclusion and Further Work

9.1 Conclusion

The thesis has taken the reader through the basics of modern security concepts as well as emphasized the difference between conventional and quantum computers. Furthermore, explained the threat brought by Shor's and Grover's algorithms when performed on a large quantum computer. Continuing with an exploration of One-Time, Few-Time and Many-Time signature schemes. Wherein key generation, signature generation, signature verification and security aspects of the signature schemes was described. However, we will like to point the reader to the original paper of each described algorithm for more specific mathematically security reduction. Furthermore, we explained the difference between stateful and stateless signature scheme.

Hash-based signatures are highly adjustable to adapt to variously environment and applications. Although future applications will not have only one stand-alone solution for the digital signature. Still, hash-based signatures will be a good fit for many of them. As it is for 2018, there is no significantly difference in efficiency and signature sizes when it comes to SPHINCS family stateless schemes. On the other hand, from stateful schemes the LMS scheme provides with shorter signatures compared to XMSS. It has been proven that it is practical to implement both stateful and stateless signature scheme on constrained devices like a smart card or IoT devices.

9.2 Further Work

Hash-based signatures schemes have gotten a lot of attention in the last years. Especially most recently when they are included in the NIST "call for algorithms". However, there are still topics to be covered in more extend for example:

- Examine the physical aspect of implementation attacks on hash-based signature schemes.
- More performance testing on hash-based signature schemes proposed for standardization.
- Further research for shorter signature sizes and faster both signing and verification process.

Bibliography

[1] François Arnault. Rabin-miller primality test: composite numbers which pass it. *Mathematics of Computation*, 64(209):355–361, 1995.

[2] Jean-Philippe Aumasson and Guillaume Endignoux. Improving stateless hash-based signatures. Technical report, Cryptology ePrint Archive, Report 2017/933, https://eprint. iacr. org/2017/933. 40, 2017.

[3] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. Sha-3 proposal blake. *Submission to NIST*, 2008.

[4] Georg Becker. Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep*, 2008.

[5] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 431–448, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[7] Daniel Bernstein, Erik Dahmen, and Buch. *Introduction to Post-Quantum Cryptography.* Springer-Verlag Berlin Heidelberg, 2010.

[8] Daniel J Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M Lauridsen, et al. Sphincs. 2017.

[9] 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.

[10] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-) security of 64-bit block ciphers: Collision attacks on http over tls and openvpn. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 456–467. ACM, 2016.

[11] Lev S. Bishop, Sergey Bravyi, Andrew Cross, Jay M. Gambetta, and John Smolin. Quantum Volume. Technical report, IBM, March 2017. `https://www.research.ibm.com/ibm-q/resources/quantum-volume.pdf`.

[12] Simon Bone and Matias Castro. A brief history of quantum computing. *Imperial College in London, http://www. doc. ic. ac. uk/˜ nd/surprise_ 97/journal/vol4/spb3*, 1997.

[13] Gilles Brassard, Peter Høyer, and Alain Tapp. *Quantum Cryptanalysis of Hash and Claw-Free Functions*, pages 163–169. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[14] briolidz. Ipsec modes. https://briolidz.files.wordpress.com/2012/01/esp.png?w=625.

[15] William Buchanan and Alan Woodward. Will Quantum Computers be the End of Public Key Encryption? *Journal of Cyber Security Technology*, 1(1):1–22, 2016.

[16] 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.

[17] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle signatures with virtually unlimited signature capacity. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 31–45, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[18] Johannes Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In *International Workshop on Post-Quantum Cryptography*, pages 63–78. Springer, 2008.

[19] Johannes Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. Cmss – an improved merkle signature scheme. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology - INDOCRYPT 2006*, pages 349–363, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[20] Johannes A Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. *Africacrypt*, 11:363–378, 2011.

[21] Ciara Byrne. The Golden Age Of Quantum Computing Is Upon Us (Once We Solve These Tiny Problems). `https://www.fastcompany.com/3045708/big-tiny-problems-for-quantum-computing`, 2015.

[22] Y. Nir P. Eronen T. Kivinen C. Kaufman, P. Hoffman. Internet key exchange protocol version 2 (ikev2). Technical report, The Internet Engineering Task Force (IETF), `10.17487/RFC7296`, October 2014.

[23] Matthew Campagna, Lidong Chen, Özgür Dagdelen, Jintai Ding, Jennifer K. Fernick, Nicolas Gisin, Donald Hayford, Thomas Jennewein, Norbert Lütkenhaus, Michele Mosca, Brian Neill, Mark Pecen, Ray Perlner, Grégoire Ribordy, John M. Schanck, Douglas Stebila, Nino Walenta, William Whyte, and Zhenfei Zhang. Quantum safe cryptography and security. Technical report, ETSI (European Telecommunications Standards Institute), June 2015. An introduction, benefits, enablers and challenges.

[24] Computer Security Resource Center. Post-quantum crypto project, 2016. Online; accessed 28 mars 2017.

[25] Shu-jen Chang, Ray Perlner, William E Burr, Meltem Sönmez Turan, John M Kelsey, Souradyuti Paul, and Lawrence E Bassham. Third-round report of the sha-3 cryptographic hash algorithm competition. *NIST Interagency Report*, 7896, 2012.

[26] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. NIST: Report on Post-Quantum Cryptography. Technical report, NIST, 2016.

[27] Luis Carlos Coronado García. *Provably secure and practical signature schemes.* PhD thesis, Technische Universität, 2006.

[28] S. Farrell S. Boeyen R. Housley W. Polk D. Cooper, S. Santesson. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. Technical report, NIST, Microsoft, Trinity College Dublin, Entrust, Vigil Security, "https://tools.ietf.org/html/rfc5280", May 2008.

[29] M. Schneider J. Turner D. Maughan, M. Schertler. Internet security association and key management protocol (isakmp). Technical report, National Security Agency (NSA), Securify Inc., RABA Technologies Inc., "https://tools.ietf.org/html/rfc2408", November 1998.

[30] D-Wave. Quantum Computing: How D-Wave Systems Work. http://www.dwavesys.com/our-company/meet-d-wave.

[31] Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, pages 109–123, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[32] Quynh Dang. *Recommendation for applications using approved hash algorithms*. US Department of Commerce, National Institute of Standards and Technology, 2008.

[33] Quynh H Dang. Randomized hashing for digital signatures. *Special Publication (NIST SP)-800-106*, 2009.

[34] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976.

[35] Chris Dods, Nigel P Smart, and Martijn Stam. Hash based digital signature schemes. *Lecture notes in computer science*, 3796:96, 2005.

[36] Jack Dongarra. Report on the sunway taihulight system. Technical report, University of Tennessee - Oak Ridge National Laboratory, "http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf", 2016.

[37] Raouf Dridi and Hedayat Alghassi. Prime factorization using quantum annealing and computational algebraic geometry. *Scientific Reports*, 7:43048, 2017.

[38] Thai Duong and Juliano Rizzo. Flickr's api signature forgery vulnerability. *Tech. Rep.*, 2009.

[39] Miloslav Dusek, Norbert Lutkenhaus, and Martin Hendrych. "Chapter 5 - Quantum Cryptography". In E. Wolf, editor, *Progress in Optics*, volume 49 of *Progress in Optics*, pages 381 – 454. Elsevier, 2006.

[40] Morris J Dworkin, Elaine B Barker, James R Nechvatal, James Foti, Lawrence E Bassham, E Roback, and James F Dray Jr. Advanced encryption standard (aes). *Federal Inf. Process. Stds.(NIST FIPS)-197*, 2001.

[41] Nicky Mouha Elaine Barker. Sp 800-67 rev. 2. recommendation for the triple data encryption algorithm (tdea) block cipher. *NIST Special Publication*, 2017.

[42] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[43] PUB FIPS. 46-3: Data encryption standard (des). *National Institute of Standards and Technology*, 25(10):1–22, 1999.

[44] PUB Fips. 186-2. digital signature standard (dss). *National Institute of Standards and Technology (NIST)*, 20:13, 2000.

[45] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[46] Dieter Gollman. *Computer Securuty.* John Wiley & Sons, 2001.

[47] Lov Grover. A Fast Quantum Mechanical Algorithm For Database Search. Technical report, Bell Labs, New Jersey, 1996.

[48] Shai Halevi and Hugo Krawczyk. The rmx transform and digital signatures. In *2nd NIST Hash Workshop*, volume 21, 2006.

[49] Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In *Crypto*, volume 4117, pages 41–59. Springer, 2006.

[50] R Housley. Rfc 2630: Cryptographic message syntax cms, 1999.

[51] A Huelsing, D Butin, S Gazdag, J Rijneveld, and A Mohaisen. Xmss: extended merkle signature scheme. Technical report, 2018.

[52] Andreas Hülsing. W-ots+-shorter signatures for hash-based signature schemes. *Africacrypt*, 7918:173–188, 2013.

[53] Andreas Hülsing, Christoph Busold, and Johannes Buchmann. Forward secure signatures on smart cards. In *International Conference on Selected Areas in Cryptography*, pages 66–80. Springer, 2012.

[54] Andreas Hülsing, Denis Butin, Stefan Gazdag, and Aziz Mohaisen. Xmss: Extended hash-based signatures. In *Crypto Forum Research Group Internet-Draft.(2015). draft-irtf-cfrg-xmss-hash-based-signatures-01*, 2015.

[55] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal parameters for xmss mt. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, pages 194–208, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[56] Andreas Hülsing, Joost Rijneveld, and Peter Schwabe. Armed sphincs. In *Public-Key Cryptography–PKC 2016*, pages 446–470. Springer, 2016.

[57] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. *IACR Cryptology ePrint Archive*, 2015:1256, 2015.

[58] 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.

[59] Andreas Hülsing. Hash-based signatures. Technical report, Post-Quantum Cryptography, 2017.

[60] IANA. Dnssec information. Accessed online; july 2017.

[61] Audun Jøsang. A consistent definition of authorization. In *International Workshop on Security and Trust Management*, pages 134–144. Springer, 2017.

[62] EJ Jung. Stream cipher. `"http://www.cs.usfca.edu/~ejung/courses/686/lectures/03stream.pdf"`, January 2010. Online; accessed 4 june 2016.

[63] S. Kent. Privacy enhancement for internet electronic mail: Part ii: Certificate-based key management. Technical report, BBN, IAB IRTF PSRG, IETF PEM, `"https://tools.ietf.org/html/rfc1422"`, February 1993.

[64] S. Kent. Ip authentication header. Technical report, BBN Technologies, `"https://tools.ietf.org/html/rfc4302"`, December 2005.

[65] S. Kent. Ip encapsulating security payload (esp). Technical report, BBN Technologies, "https://tools.ietf.org/html/rfc4303", December 2005.

[66] Cameron F. Kerry, Acting Secretary, and Charles Romine Director. Fips pub 186-4 federal information processing standards publication digital signature standard (dss), 2013.

[67] Zach Kirsch. *Quantum Computing: The Risk to Existing Encryption Methods.* PhD thesis, Tufts University, Massachusetts, 2015. http://www.cs.tufts.edu/comp/116/archive/fall2015/zkirsch.pdf.

[68] H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In *Network and Distributed System Security*, "10.1109/NDSS.1996.492418", 1996. IBM, IEEE.

[69] Information Technology Laboratory. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, National Institute of Standards and Technology, "http://www.nist.gov/customcf/get_pdf.cfm?pub_id=919061", 2015. Online; accessed 12 june 2016.

[70] Chis Lee. How ibm's new five-qubit universal quantum computer works. Technical report, D-Wave, "https://www.youtube.com/watch?v=UV_RlCAc5Zs", 2016. Online; accessed 4 mai 2016.

[71] Frank T Leighton and Silvio Micali. Large provably fast and secure digital signature schemes based on secure hash functions, July 11 1995. US Patent 5,432,852.

[72] Prof. Leslie. Leslie lamport home page, 2017. Online; accessed 1 mars 2017 http://lamport.azurewebsites.net/pubs/pubs.html.

[73] P. Hesse S. Joseph R. Nicholas M. Cooper, Y. Dzambasow. Internet x.509 public key infrastructure: Certification path building. Technical report, Orion Security Solutions, A&N Associates, Gemini Security Solutions, Van Dyke Technologies, BAE Systems, "https://tools.ietf.org/html/rfc4158", September 2005.

[74] B. Kaliski M. Nystrom. Pkcs #10: Certification request syntax specification version 1.7. Technical report, RSA Security, "https://tools.ietf.org/html/rfc2986", November 2000.

[75] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. The Impact of Quantum Computing on Present Cryptography. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 9(3):405–414, 2018.

[76] David McGrew and Michael Curcio. Hash-based signatures. 2015.

[77] Dr. David A. McGrew, Michael Curcio, and Scott Fluhrer. Hash-Based Signatures. Internet-Draft draft-mcgrew-hash-sigs-11, Internet Engineering Task Force, April 2018. Work in Progress.

[78] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. On the collision resistance of ripemd-160. *Information Security*, pages 101–116, 2006.

[79] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

[80] Ralph Charles Merkle, Ralph Charles, et al. Secrecy, authentication, and public key systems. *none*, 1979.

[81] Dustin Moody. The ship has sailed: The nist post-quantum crypto "competition".

[82] Juha Muhonen and T Dehollain. Storing Quantum Information For 30 Seconds In a Nanoelectronic Device. *Nature Nanotechnology*, 9:986–991, 2014.

[83] National Security Agency. National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information. Technical report, NSA, 2003. `http://csrc.nist.gov/groups/STM/cmvp/documents/CNSS15FS.pdf`.

[84] NIST. Block cipher modes. Technical report, NIST, "`http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html`", 2001. Online; accessed 5 june 2016.

[85] NSA. The case for elliptic curve cryptography, 2009. Online; accessed 28. mars 2017.

[86] H. Orman. The oakley key determination protocol. Technical report, Department of Computer Science University of Arizona, "`https://tools.ietf.org/html/rfc2412`", November 1998.

[87] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[88] Adrian Perrig. The biba one-time signature and broadcast authentication protocol. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 28–37. ACM, 2001.

[89] John Proos and Christof Zalka. Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves. *Quantum Info. Comput.*, 3(4):317–344, 2003.

[90] Lili Qiu, Yin Zhang, Feng Wang, Mi Kyung, and Han Ratul Mahajan. Trusted computer system evaluation criteria. In *National Computer Security Center*. Citeseer, 1985.

[91] M.0. Rabin. Digitalized signatures. Technical report, Hebrew University of Jerusalem, Massachusetts Institute of Technology, `https://smartech.gatech.edu/bitstream/handle/1853/40598/g-36-619_142482.pdf`, December 1978.

[92] Leonid Reyzin and Natan Reyzin. Better than biba: Short one-time signatures with fast signing and verifying. In *Australasian Conference on Information Security and Privacy*, pages 144–153. Springer, 2002.

[93] 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.

[94] Dongyoung Roh, Sangim Jung, and Daesung Kwon. Winternitz signature scheme using nonadjacent forms. *Security and Communication Networks*, 2018, 2018.

[95] 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.

[96] A. Malpani S. Galperin C. Adams R. Ankney S. Santesson, M. Myers. X.509 internet public key infrastructure online certificate status protocol - ocsp. Technical report, 3xA Security, TraceRoute Security, CA Technologies, A9, University of Ottawa, `"https://tools.ietf.org/html/rfc6960"`, June 2013.

[97] Bruce Schneier. NSA Plans for a Post-Quantum World, 2015.

[98] Claude E Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949.

[99] P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 124–134, Washington, DC, USA, 1994. IEEE Computer Society.

[100] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. *IACR Cryptology ePrint Archive*, 2017:190, 2017.

[101] M. StJohns. Automated updates of dns security (dnssec) trust anchors. Technical report, Independent, `"https://tools.ietf.org/html/rfc5011"`, September 2007.

[102] Michael Szydlo. Merkle tree traversal in log space and time. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 541–554, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[103] E. Rescorla T. Dierks Independent. Message flow for a full handshake. Figure 1 in [104], August 2008.

[104] E. Rescorla T. Dierks Independent. The transport layer security (tls) protocol version 1.2. Technical report, RTFM Inc., `"https://tools.ietf.org/html/rfc5246"`, August 2008.

[105] Gene Tsudik. Message authentication with one-way hash functions. In *INFOCOM'92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE*, pages 2055–2059. IEEE, 1992.

[106] Unknown. Aes encryption mode. `"http://i.stack.imgur.com/bXAUL.png"`, 2016. Online; accessed 4 june 2016.

[107] Unknown. Asymmetric cryptography. `"https://i\discretionary{-}{}{}msdn.sec.s-msft.com/dynimg/IC21919.gif"`, 2016. Online; accessed 8 mai 2016.

[108] Unknown. Pki structure. `"http://1.bp.blogspot.com/_ZbCE6_nPONI/S_CUvF8mazI/AAAAAAAAABw/QEOi8YoD4_c/s1600/3.jpg"`, 2018. Online; accessed 10 mars 2018.

[109] Unknown. X.509 certificate. `""`, 2018. Online; accessed 10 mars 2018.

[110] Umesh Vazirani. On The Power of Quantum Computation. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 356(1743):1759–1768, 1998.

[111] C. Vu and M. Fey. IBM Builds Its Most Powerful Universal Quantum Computing Processors. Technical report, IBM, May 2017. `https://www-03.ibm.com/press/us/en/pressrelease/52403.wss`.

[112] M.E. Hellman. W. Diffie. New directions in cryptography. Technical report, IEEE Transactions on Information Theory, `https://www-ee.stanford.edu/~hellman/publications/24.pdf`, November 1976.

[113] T.F. Walter. Is Quantum Computing for Real? An Interview with Catherine McGeoch of D-Wave Systems. *Ubiquity*, 2017:2:1–2:20, July 2017.

[114] www.joshuthomas.com. Symmetric cryptography. `"http://www.joshuthomas.com/wp-content/uploads/2014/07/keymanagement-in-symmetric-systems.jpg"`, 2016. Online; accessed 8 mai 2016.

Appendix A

The Impact of Quantum Computing on Present Cryptography

Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, Audun Jøsang
Department of Informatics, University of Oslo, Norway
Email(s): {vasileim, kamerv, mateusdz, josang}@ifi.uio.no

Abstract—**The aim of this paper is to elucidate the implications of quantum computing in present cryptography and to introduce the reader to basic post-quantum algorithms. In particular the reader can delve into the following subjects: present cryptographic schemes (symmetric and asymmetric), differences between quantum and classical computing, challenges in quantum computing, quantum algorithms (Shor's and Grover's), public key encryption schemes affected, symmetric schemes affected, the impact on hash functions, and post quantum cryptography. Specifically, the section of Post-Quantum Cryptography deals with different quantum key distribution methods and mathematical-based solutions, such as the BB84 protocol, lattice-based cryptography, multivariate-based cryptography, hash-based signatures and code-based cryptography.**

Keywords—quantum computers; post-quantum cryptography; Shor's algorithm; Grover's algorithm; asymmetric cryptography; symmetric cryptography

I. Introduction

There is no doubt that advancements in technology and particularly electronic communications have become one of the main technological pillars of the modern age. The need for confidentiality, integrity, authenticity, and non-repudiation in data transmission and data storage makes the science of cryptography one of the most important disciplines in information technology. Cryptography, etymologically derived from the Greek words hidden and writing, is the process of securing data in transit or stored by third party adversaries. There are two kinds of cryptosystems; symmetric and asymmetric.

Quantum computing theory firstly introduced as a concept in 1982 by Richard Feynman, has been researched extensively and is considered the destructor of the present modern asymmetric cryptography. In addition, it is a fact that symmetric cryptography can also be affected by specific quantum algorithms; however, its security can be increased with the use of larger key spaces. Furthermore, algorithms that can break the present asymmetric cryptoschemes whose security is based on the difficulty of factorizing large prime numbers and the discrete logarithm problem have been introduced. It appears that even elliptic curve cryptography which is considered presently the most secure and efficient scheme is weak against quantum computers. Consequently, a need for cryptographic algorithms robust to quantum computations arose.

The rest of the paper deals initially with the analysis of symmetric cryptography, asymmetric cryptography and hash functions. Specifically, an emphasis is given on algorithms that take advantage of the difficulty to factorize large prime numbers, as well as the discrete logarithm problem. We move on by giving an introduction to quantum mechanics and the challenge of building a true quantum computer. Furthermore, we introduce two important quantum algorithms that can have a huge impact in asymmetric cryptography and less in symmetric, namely Shor's algorithm and Grover's algorithm respectively. Finally, post-quantum cryptography is presented. Particularly, an emphasis is given on the analysis of quantum key distribution and some mathematical based solutions such as lattice-based cryptography, multivariate-based cryptography, hash-based signatures, and code-based cryptography.

II. Present Cryptography

In this chapter we explain briefly the role of symmetric algorithms, asymmetric algorithms and hash functions in modern cryptography. We analyze the difficulty of factorizing large numbers, as well as the discrete logarithm problem which is the basis of strong asymmetric ciphers.

A. Symmetric Cryptography

In symmetric cryptography, the sender and the receiver use the same secret key and the same cryptographic algorithm to encrypt and decrypt data. For example, Alice can encrypt a plaintext message using her shared secret key and Bob can decrypt the message using the same cryptographic algorithm Alice used and the same shared secret key. The key needs to be kept secret, meaning that only Alice and Bob should know it; therefore, an efficient way for exchanging secret keys over public networks is demanded. Asymmetric cryptography was introduced to solve the problem of key distribution in symmetric cryptography. Popular symmetric algorithms include the advanced encryption standard (AES) and the data encryption standard (3DES).

B. Asymmetric Cryptography

Asymmetric cryptography or public key cryptography (PKC) is a form of encryption where the keys come in pairs. Each party should have its own private and public key. For instance, if Bob wants to encrypt a message, Alice would send her public key to Bob and then Bob can encrypt the message with Alice's public key. Next, Bob would transmit the encrypted message to Alice who is able to decrypt the message with her private key. Thus, we encrypt the message with a public key and only the person who owns the private key can decrypt the message.

Asymmetric cryptography additionally is used for digital signatures. For example, Alice can sign a document digitally with her private key and Bob can verify the signature with Alice's known public key. The security of PKC rests on computational problems such as the difficulty of factorizing large prime numbers and the discrete logarithm problem. Such kind of algorithms are called one-way functions because they are easy to compute in one direction but the inversion is difficult [1].

1) Factorization Problem - RSA Cryptosystem: One of the most important public-key schemes is RSA invented by Ronald Rivest, Adi Shamir, and Leonard Adleman in 1977. RSA exploits the difficulty of factorizing bi-prime numbers. According to Paar and Pelzl [2], RSA and in general asymmetric algorithms are not meant to replace symmetric algorithms because they are computationally costly. RSA is mainly used for secure key exchange between end nodes and often used together with symmetric algorithms such as AES, where the symmetric algorithm does the actual data encryption and decryption. Kirsch [3] stated that RSA is theoretically vulnerable if a fast factorizing algorithm is introduced or huge increase in computation power can exist. The latter can be achieved with the use of quantum mechanics on computers, known as quantum-computers.

2) Discrete Logarithm Problem (DLP): Asymmetric cryptographic systems such as Diffie-Hellman (DH) and Elliptic Curve Cryptography (ECC) are based on DLP. The difficulty of breaking these cryptosystems is based on the difficulty in determining the integer r such that $g^r = x \mod p$. The integer r is called the discrete logarithm problem of x to the base g, and we can write it as $r = \log_g x \mod p$. The discrete logarithm problem is a very hard problem to compute if the parameters are large enough.

Diffie-Hellman is an asymmetric cipher that uses the aforementioned property to transmit keys securely over a public network. Recently, keys larger or equal to 2048 bits are recommended for secure key exchange. In addition, another family of public key algorithms known as Elliptic Curve Cryptography is extensively used. ECC provides the same level of security as RSA and DLP systems with shorter key operands which makes it convenient to be used by systems of low computational resources. ECC uses a pair (x, y) that fits into the equation $y^2 = x^3 + ax + b \mod p$ together with an imaginary point Θ (theta) at infinity, where $a, b \in Z_p$ and $4a^3 + 27b^2 \neq 0 \mod p$ [2]. ECC needs a cyclic Group G and the primitive elements we use, or pair elements, to be of order G. ECC is considered the most secure and efficient asymmetric cryptosystem, but this tends to change with the introduction of quantum computers as it is explained in the next sections.

III. Quantum Computing vs Classical Computing

In 1982, Richard Feynman came up with the idea of *quantum computer*, a computer that uses the effects of quantum mechanics to its advantage. Quantum mechanics is related to microscopic physical phenomena and their strange behavior. In a traditional computer the fundamental blocks are called bits and can be observed only in two states; 0 and 1. Quantum computers instead use quantum bits also usually referred as *qubits* [4]. In a sense, qubits are particles that can exist not only in the 0 and 1 state but in both simultaneously, known as superposition. A particle collapses into one of these states when it is inspected. Quantum computers take advantage of this property mentioned to solve complex problems. An operation on a qubit in superposition acts on both values at the same time. Another physical phenomenon used in quantum computing is quantum entanglement. When two qubits are entangled their quantum state can no longer be described independently of each other, but as a single object with four different states. In addition, if one of the two qubits state change the entangled qubit will change too regardless of the distance between them. This leads to true parallel processing power [5]. The combination of the aforementioned phenomena result in exponential increase in the number of values that can be processed in one operation, when the number of entanglement qubits increase. Therefore, a *n-qubit* quantum computer can process 2^n operations in parallel.

Two kinds of quantum computers exists; universal and non-universal. The main difference between the two is that universal quantum computers are developed to perform any given task, whereas non-universal quantum computers are developed for a given purpose (e.g., optimization of machine learning algorithms). Examples are, D-Wave's 2000+ qubits non-universal quantum computer [6] and IBM's 17 qubits universal quantum computer with proper error correction. IBM's quantum computer is currently the state of the art of universal quantum computers [7]. Both D-Wave and IBM have quantum computers accessible online for research purposes. Additionally, in October 2017, Intel in collaboration with QuTech announced their 17-qubits universal quantum computer [7].

Bone and Castro [8] stated that a quantum computer is completely different in design than a classical computer that uses the traditional transistors and diodes. Researchers have experimented with many different designs such as quantum dots which are basically electrons being in a superposition state, and computing liquids. Besides, they remarked that quantum computers can show their superiority over the classical computers only when used with algorithms that exploit the power of quantum parallelism. For example, a quantum computer would not be any faster than a traditional computer in multiplication.

A. Challenges in Quantum Computing

There are many challenges in quantum computing that many researchers are working on.

- Quantum algorithms are mainly probabilistic. This means that in one operation a quantum computer returns many solutions where only one is the correct. This trial and error for measuring and verifying the correct answer weakens the advantage of quantum computing speed [3].
- Qubits are susceptible to errors. They can be affected by heat, noise in the environment, as well as stray electromagnetic couplings. Classical computers are susceptible to bit-flips (a zero can become one and vise versa). Qubits suffer from bit-flips as well as phase errors. Direct inspection for errors should be avoided as it will cause the value to collapse, leaving its superposition state.
- Another challenge is the difficulty of coherence. Qubits can retain their quantum state for a short period

of time. Researchers at the University of New South Wales in Australia have created two different types of qubits (Phosphorous atom and an Artificial atom) and by putting them into a tiny silicon (*silicon 28*) they were able to elliminate the magnetic noise that makes them prone to errors. Additionally, they stated that the Phosphorous atom has 99.99% accuracy which accounts for 1 error every 10,000 quantum operations [9]. Their qubits can remain in superposition for a total of 35 seconds which is considered a world record [10]. Moreover, to achieve long coherence qubits need not only to be isolated from the external world but to be kept in temperatures reaching the absolute zero. However, this isolation makes it difficult to control them without contributing additional noise [3].

IBM in 2017, introduced the definition of *Quantum Volume*. Quantum volume is a metric to measure how powerful a quantum computer is based on how many qubits it has, how good is the error correction on these qubits, and the number of operations that can be done in parallel. Increase in the number of qubit does not improve a quantum computer if the error rate is high. However, improving the error rate would result in a more powerful quantum computer [11].

IV. Cryptosystems Vulnerable to Quantum Algorithms

This section discusses the impact of quantum algorithms on present cryptography and gives an introduction to Shor's algorithm and Grover's algorithm. Note that Shor's algorithm explained in the following subsection makes the algorithms that rely on the difficulty of factorizing or computing discrete logarithms vulnerable.

Cryptography plays an important role in every electronic communication system today. For example the security of emails, passwords, financial transactions, or even electronic voting systems require the same security objectives such as confidentiality and integrity [12]. Cryptography makes sure that only parties that have exchanged keys can read the encrypted message (also called authentic parties). Quantum computers threaten the main goal of every secure and authentic communication because they are able to do computations that classical (conventional) computers cannot. Consequently, quantum computers can break the cryptographic keys quickly by calculating or searching exhaustively all secret keys, allowing an eavesdropper to intercept the communication channel between authentic parties (sender/receiver). This task is considered to be computational infeasible by a conventional computer [13].

According to NIST, quantum computers will bring the end of the current public key encryption schemes [14]. Table I adapted from NIST shows the impact of quantum computing on present cryptographic schemes.

129

A. Shor's Algorithm in Asymmetric Cryptography

In 1994, the mathematician Peter Shor in his paper "Algorithms for Quantum Computation: Discrete Logarithms and Factoring" [15], proved that factorizing large integers would change fundamentally with a quantum computer.

Shor's algorithm can make modern asymmetric cryptography collapse since is it based on large prime integer factorization or the discrete logarithm problem. To understand how Shor's algorithm factorizes large prime numbers we use the following example. We want to find the prime factors of number 15. To do so, we need a 4-qubit register. We can visualize a 4-qubit register as a normal 4-bit register of a traditional computer. Number 15 in binary is 1111, so a 4-qubit register is enough to accommodate (calculate) the prime factorization of this number. According to Bone and Castro [8], a calculation performed on the register can be thought as computations done in parallel for every possible value that the register can take (0-15). This is also the only step needed to be performed on a quantum computer.

The algorithm does the following:

- n = 15, is the number we want to factorize
- x = random number such as $1 < x < n - 1$
- x is raised to the power contained in the register (every possible state) and then divided by n
  The remainder from this operation is stored in a second 4-qubit register. The second register now contains the superposition results. Let's assume that $x = 2$ which is larger than 1 and smaller than 14.
- If we raise x to the powers of the 4-qubit register which is a maximum of 15 and divide by 15, the remainders are shown in Table II.
  What we observe in the results is a repeating sequence of 4 numbers (1,2,4,8). We can confidently say then that $f = 4$ which is the sequence when $x = 2$ and $n = 15$. The value f can be used to calculate a possible factor with the following equation:
  Possible factor: $P = x^{f/2} - 1$

In case we get a result which is not a prime number we repeat the calculation with different f values.

Shor's algorithm can be used additionally for computing discrete logarithm problems. Vazirani [16] explored in detail the methodology of Shor's algorithm and showed that by starting from a random superposition state of two integers, and by performing a series of Fourier transformations, a new superposition can be set-up to give us with high probability two integers that satisfy an equation. By using this equation we can calculate the value r which is the unknown "exponent" in the DLP.

B. Grover's algorithm in Symmetric Cryptography

Lov Grover created an algorithm that uses quantum computers to search unsorted databases [17]. The algorithm can find a specific entry in an unsorted database of N entries in $\sqrt{N}$ searches. In comparison, a conventional computer would need $N/2$ searches to find the same entry. Bone and Castro [8] remarked the impact of a possible application of Grover's algorithm to crack Data Encryption Standard (DES), which relies its security on a 56-bit key. The authors remarked that the algorithm needs only 185 searches to find the key.

Currently, to prevent password cracking we increase the number of key bits (larger key space); as a result, the number of

TABLE I. IMPACT ANALYSIS OF QUANTUM COMPUTING ON ENCRYPTION SCHEMES (ADAPTED FROM [14])

Cryptographic Algorithm	**Type**	**Purpose**	**Impact From Quantum Computer**
AES-256	Symmetric key	Encryption	Secure
SHA-256, SHA-3	–	Hash functions	Secure
RSA	Public key	Signatures, key establishment	No longer secure
ECDSA, ECDH (Elliptic Curve Cryptography)	Public key	Signatures, key exchange	No longer secure
DSA (Finite Field Cryptography)	Public key	Signatures, key exchange	No longer secure

TABLE II. 4-QUBIT REGISTERS WITH REMAINDERS

Register 1:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Register 2:	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8

searches needed to crack a password increases exponentially. Buchmann et al. [18] stated that Grover's algorithm have some applications to symmetric cryptosystems but it is not as fast as Shor's algorithm.

C. Asymmetric Encryption Schemes Affected

All public key algorithms used today are based on two mathematical problems, the aforementioned factorization of large numbers (e.g., RSA) and the calculation of discrete logarithms (e.g., DSA signatures and ElGamal encryption). Both have similar mathematical structure and can be broken with Shor's algorithm rapidly. Recent algorithms based on elliptic curves (such as ECDSA) use a modification of the discrete logarithm problem that makes them equally weak against quantum computers. Kirsch and Chow [3] mentioned that a modified Shor's algorithm can be used to decrypt data encrypted with ECC. In addition, they emphasized that the relatively small key space of ECC compared to RSA makes it easier to be broken by quantum computers. Furthermore, Proos and Zalka [19] explained that 160-bit elliptic curves could be broken by a 1000-qubit quantum computer, while factorizing 1024-bit RSA would require a 2000-qubit quantum computer. The number of qubits needed to break a cryptosystem is relative to the algorithm proposed. In addition, they show in some detail how to use Shor's algorithm to break ECC over GF(p).

On the other hand, Grover's algorithm is a threat only to some symmetric cryptographic schemes. NIST [14] points out that if the key sizes are sufficient, symmetric cryptographic schemes (specifically the Advanced Encryption Standard-AES) are resistant to quantum computers. Another aspect to be taken into consideration is the robustness of algorithms against quantum computing attacks also known as quantum cryptanalysis.

In table III, a comparison of classical and quantum security levels for the most used cryptographic schemes is presented.

D. Symmetric Encryption Schemes Affected

For symmetric cryptography quantum computing is considered a minor threat. The only known threat is Grover's algorithm that offers a square root speed-up over classical brute force algorithms. For example, for a n-bit cipher the quantum computer operates on ($\sqrt{2^n} = 2^{n/2}$). In practice, this means that a symmetric cipher with a key length of 128-bit (e.g., AES-128) would provide a security level of 64-bit. We recall here that security level of 80-bit is considered secure. The Advanced Encryption Standard (AES) is considered to be one of the cryptographic primitives that is resilient in quantum computations, but only when is used with key sizes of 192 or 256 bits. Another indicator of the security of AES in the post-quantum era is that NSA (The National Security Agency) allows AES cipher to secure (protect) classified information for security levels, SECRET and TOP SECRET, but only with key sizes of 192 and 256 bits [20].

TABLE III. COMPARISON OF CLASSICAL AND QUANTUM SECURITY LEVELS FOR THE MOST USED CRYPTOGRAPHIC SCHEMES

Crypto Scheme	**Key Size**	**Effective Key Strength/Security Level (in bits)**	
		Classical Computing	**Quantum Computing**
RSA-1024	1024	80	0
RSA-2048	2048	112	0
ECC-256	256	128	0
ECC-384	384	256	0
AES-128	**128**	**128**	**64**
AES-256	**256**	**256**	**128**

E. Hash Functions

The family of hash functions suffer from a similar problem as symmetric ciphers since their security depends on a fixed output length. Grover's algorithm can be utilized to find a collision in a hash function in square root steps of its original length (it is like searching an unsorted database). In addition, it has been proved that it is possible to combine Grover's algorithm with the birthday paradox. Brassard et al. [21] described a quantum birthday attack. By creating a table of size $\sqrt[3]{N}$ and utilizing Grover's algorithm to find a collision an attack is said to work effectively. This means that to provide a $b - bit$ security level against Grover's quantum algorithm a hash function must provide at least a $3b - bit$ output. As a result, many of the present hash algorithms are disqualified for use in the quantum era. However, both SHA-2 and SHA-3 with longer outputs, remain quantum resistant.

V. POST-QUANTUM CRYPTOGRAPHY

The goal of post-quantum cryptography (also known as quantum-resistant cryptography) is to develop cryptographic systems that are secure against both quantum and conventional computers and can interoperate with existing communication protocols and networks [14]. Many post-quantum public key candidates are actively investigated the last years. In 2016, NIST announced a call for proposals of algorithms that are believed to be quantum resilient with a deadline in November

2017. In January 2018, NIST published the results of the first round. In total 82 algorithms were proposed from which 59 are encryption or key exchange schemes and 23 are signature schemes. After 3 to 5 years of analysis NIST will report the findings and prepare a draft of standards [22]. Furthermore, the National Security Agency (NSA) has already announced plans to migrate their cryptographic standards to post-quantum cryptography [23].

The cryptographic algorithms presented in this section do not rely on the hidden subgroup problem (HSP) such as factorizing integers or computing discrete logarithms, but different complex mathematical problems.

A. Quantum Key Distribution

Quantum Key Distribution (QKD) addresses the challenge of securely exchanging a cryptographic key between two parties over an insecure channel. QKD relies on the fundamental characteristics of quantum mechanics which are invulnerable to increasing computational power, and may be performed by using the quantum properties of light, lasers, fibre-optics as well as free space transmission technology. QKD was first introduced in 1984 when Charles Bennett and Gilles Brassard developed their BB84 protocol [24, 25]. Research has led to the development of many new QKD protocols exploiting mainly two different properties that are described right below.

Prepare-and-measure (P&M) protocols use the Heisenberg Uncertainty principle [26] stating that the measuring act of a quantum state changes that state in some way. This makes it difficult for an attacker to eavesdrop on a communication channel without leaving any trace. In case of eavesdropping the legitimate exchange parties are able to discard the corrupted information as well as to calculate the amount of information that has been intercepted [27]. This property was exploited in BB84.

Entanglement based (EB) protocols use pairs of entangled objects which are shared between two parties. As explained in III, entanglement is a quantum physical phenomenon which links two or more objects together in such a way that afterwards they have to be considered as one object. Additionally, measuring one of the objects would affect the other as well. In practice when an entangled pair of objects is shared between two legitimate exchange parties anyone intercepting either object would alter the overall system. This would reveal the presence of an attacker along with the amount of information that the attacker retrieved. This property was exploited in E91 [28] protocol.

Both of the above-mentioned approaches are additionally divided into three families; discrete variable coding, continuous variable coding and distributed phase reference coding. The main difference between these families is the type of detecting system used. Both discrete variable coding and distributed phase reference coding use photon counting and post-select the events in which a detection has effectively taken place [29]. Continuous variable coding uses homodyne detection [29] which is a comparison of modulation of a single frequency of an oscillating signal with a standard oscillation.

A concise list of QKD protocols for the aforementioned families is presented below.

Discrete variable coding protocols:

- BB84 [24, 25] - the first QKD protocol that uses four non-orthogonal polarized single photon states or low-intensity light pulses. A detailed description of this protocol is given below.
- BBM [30] - is an entanglement based version of BB84.
- E91 [28] - is based on the *gedanken experiment* [31] and the generalized Bell's theorem [32]. In addition, it can be considered an extension of Bennett and Brassard's (authors of BB84) original idea.
- SARG04 [33, 34] - is similar to BB84 but instead of using the state to code the bits, the bases are used. SARG04 is more robust than BB84 against the photon number splitting (PNS) attack.
- Six state protocol [35–37] - is a version of BB84 that uses a six-state polarization scheme on three orthogonal bases.
- Six state version of the SARG04 coding [38].
- Singapore protocol [39] - is a tomographic protocol that is more efficient than the Six state protocol.
- B92 protocol [40] - two non-orthogonal quantum states using low-intensity coherent light pulses.

Continuous variable coding protocols:

- Gaussian protocols
    - Continuous variable version of BB84 [41]
    - Continuous variable using coherent states [42]
    - Coherent state QKD protocol [43] - based on simultaneous quadrature measurements.
    - Coherent state QKD protocol [44] - based on the generation and transmission of random distributions of coherent or squeezed states.
- Discrete-modulation protocols
    - First continuous variable protocol based on coherent states instead of squeezed states [45].

Distributed phase reference coding protocols:

- Differential Phase Shift (DPS) Quantum Key Distribution (QKD) protocol [46, 47] - uses a single photon in superposition state of three basis kets, where the phase difference between two sequential pulses carries bit information.
- Coherent One Way (COW) protocol [48, 49] - the key is obtained by a time-of-arrival measurement on the data line (raw key). Additionally, an interferometer is built on a monitoring line, allowing to monitor the presence of an intruder. A prototype was presented in 2008 [50].

Discrete variable coding protocols are the most widely implemented, whereas the continuous variable and distributed phase reference coding protocols are mainly concerned with overcoming practical limitations of experiments.

1) BB84 protocol: BB84 is the first quantum cryptographic protocol (QKD scheme) which is still in use today. According to Mayers [51] BB84 is *provable secure*, explaining that a secure key sequence can be generated whenever the channel bit error rate is less than about 7% [52]. BB84 exploits the polarization of light for creating random sequence of qubits (key) that are transmitted through a quantum channel.

BB84 uses two different bases, base 1 is polarized 0^o (horizontal) or 90^o (vertical) with 0^o equal to 0 and 90^o equal to 1. Base 2 is polarized 45^o or 135^o with 45^o equal to 1 and 135^o equal to 0. Alice begins by sending a photon in one of the two bases having a value of 0 or 1. Both the base and the value should be chosen randomly. Next, Bob selects the base 1 or 2 and measures a value without knowing which base Alice has used. The key exchange process continues until they have generated enough bits. Furthermore, Bob tells Alice the sequence of the bases he used but not the values he measured and Alice informs Bob whether the chosen bases were right or wrong. If the base is right, Alice and Bob have equal bits, whereas if it is wrong the bits are discarded. In addition, any bits that did not make it to the destination are discarded by Alice. Now Alice can use the key that they just exchanged to encode the message and send it to Bob. BB84 is illustrated visually in Figure 1.

Worthy to mentioning is that this method of communication was broken by Lydersen et al. in 2010 [53]. Their experiment proved that although BB84 is *provable secure* the actual hardware implemented is not. The authors managed to inspect the secret key without the receiver noticing it by blinding the APD-based detector (avalanche photodiode).

Yuan et al. [54] proposed improvements to mitigate blinding attacks, such as monitoring the photocurrent for anomalously high values. Lydersen et al. [55] after taking into consideration the improvements of Yuan et al. [54] succeeded again to reveal the secret key without leaving any traces.

2) Photon Number Splitting Attack: The crucial issue in quantum key distribution is its security. In addition to noise in the quantum channel, the equipment is impractical to produce and detect single photons. Therefore, in practice, laser pulses are used. Producing multiple photons opens up a new attack known as Photon Number Splitting (PNS) attack. In PNS attack, an attacker (Eve) deterministically splits a photon off of the signal and stores it in a quantum memory which does not modify the polarisation of the photons. The remaining photons are allowed to pass and are transmitted to the receiver (Bob). Next, Bob measures the photons and the sender (Alice) has to reveal the encoding bases. Eve will then be able to measure all captured photons on a correct bases. Consequently, Eve will obtain information about the secret key from all signals containing more than one photon without being noticed [57].

Different solutions have been proposed for mitigating PNS attacks. The most promising solution developed by Lo et al. [58] uses decoy states to detect PNS attacks. This is achieved by sending randomly laser pulses with a lower average photon number. Thereafter, Eve cannot distinguish between decoyed signals and non-decoyed signals. This method works for both single and multi-photon pulses [59].

B. Mathematically-based Solutions

There are many alternative mathematical problems to those used in RSA, DH and ECDSA that have already been implemented as public key cryptographic schemes, and for which the Hidden Subgroup Problem (HSP) [60] does not apply; therefore, they appear to be quantum resistant.

The most researched mathematical-based implementations are the following:

- Lattice-based cryptography [61]
- Multivariate-based cryptography [62]
- Hash-based signatures [63]
- Code-based cryptography [64]

The existing alternatives and new schemes emerging from these areas of mathematics do not all necessarily satisfy the characteristics of an ideal scheme. In the following subsections we are going to give an overview of these cryptographic schemes.

1) Lattice-based Cryptography: This is a form of public-key cryptography that avoids the weaknesses of RSA. Rather than multiplying primes, lattice-based encryption schemes involve multiplying matrices. Furthermore, lattice-based cryptographic constructions are based on the presumed hardness of lattice problems, the most basic of which is the shortest vector problem (SVP) [61]. Here, we are given as input a lattice represented by an arbitrary basis and our goal is to output the shortest non-zero vector in it.

The Ajtai-Dwork (AD) [65], Goldreich-Goldwasser-Halevi (GGH) [66] and NTRU [67] encryption schemes that are explained below are lattice-based cryptosystems.

In 1997, Ajtai and Dwork[65] found the first connection between the worst and the average case complexity of the Shortest Vector Problem (SVP). They claimed that their cryptosystem is *provably secure*, but in 1998, Nguyen and Ster [68] refuted it. Furthermore, the AD public key is big and it causes message expansion making it an unrealistic public key candidate in post-quantum era.

The Goldreich-Goldwasser-Halevi (GGH) was published in 1997. GGH makes use of the Closest Vector Problem (CVP) which is known to be NP-hard. Despite the fact that GGH is more efficient than Ajtai-Dwork (AD), in 1999, Nguyen[69] proved that GGH has a major flaw; partial information on plaintexts can be recovered by solving CVP instances.

NTRU was published in 1996 by Hoffstein et al. [67]. It is used for both encryption (*NTRUEncrypt*) and digital signature (*NTRUSign*) schemes. NTRU relies on the difficulty of factorizing certain polynomials making it resistant against Shor's algorithm. To provide 128-bit post-quantum security level NTRU demands 12881-bit keys [70]. As of today there is not any known attack for NTRU.

In 2013, Damien Stehle and Ron Steinfeld developed a *provably secure* version of NTRU (SS-NTRU) [71].

In May 2016, Bernstein et al. [72] released a new version of NTRU called "NTRU Prime". NTRU Prime countermeasures

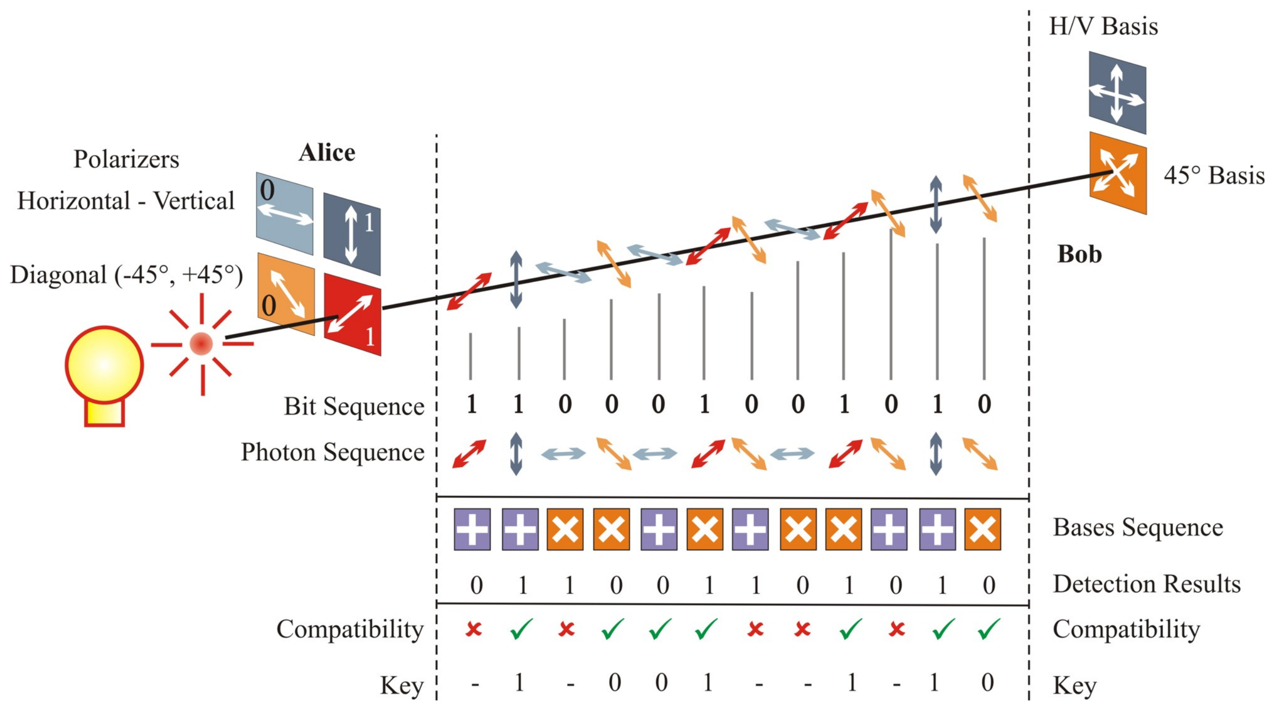


Fig. 1. Key exchange in the BB84 protocol implemented with polarization of photons (adapted from [56]).

the weaknesses of several lattice based cryptosystems, including NTRU, by using different more secure ring structures.

In conclusion, among all the lattice-based candidates mentioned above NTRU is the most efficient and secure algorithm making it a promising candidate for the post-quantum era.

2) Multivariate-based Cryptography: The security of this public key scheme relies on the difficulty of solving systems of multivariate polynomials over finite fields. Research has shown that development of an encryption algorithm based on multivariate equations is difficult [13]. Multivariate cryptosystems can be used both for encryption and digital signatures. Tao et al. [73] explained that there have been several attempts to build asymmetric pubic key encryption schemes based on multivariate polynomials; however, most of them are insecure because of the fact that certain quadratic forms associated with their central maps have low rank. The authors [73] proposed a new efficient multivariate scheme, namely Simple Matrix (ABC), based on matrix multiplication that overcomes the aforementioned weakness. In addition, multivariate cryptosystems can be used for digitals signatures. The most promising signature schemes include Unbalanced Oil and Vinegar (multivariate quadratic equations), and Rainbow. UOV has a large ratio between the number of variables and equations (3:1) making the signatures three times longer than the hash values. In addition, the public key sizes are large. On the other hand, Rainbow is more efficient by using smaller ratios which result in smaller digital signatures and key sizes [12]. 133

3) Hash-based Signatures: In this subsection, we introduce the Lamport signature scheme invented in 1979 by Leslie Lamport. Buchmann et al. [18] introduced concisely the scheme. A parameter b defines the desired security level of our system. For 128-bit b security level we need a secure hash function that takes arbitrary length input and produces 256-bit length output; thus, SHA-256 is considered an optimal solution that can be fitted with our message m.

Private key: A random number generator is used to produce 256 pairs of random numbers. Each number is 256 bits. In total our generated numbers are $2 \times 256 \times 256 = 16$ KB. Therefore, we can precisely say that the private key consists of $8b^2$ bits.

Public key: All generated numbers (private key) are hashed independently creating 512 different hashes (256 pairs) of 256-bit length each. Therefore, we can precisely say that the public key consists of $8b^2$ bits.

The next step is to sign the message. We have a hashed message *m* and then for each bit (depending on its value 0 or 1) of the message digest we choose one number from each pair that comprise the private key. As a result, we have a sequence of 256 numbers (relative to the bit sequence of the hashed message *m*). The sequence of numbers is the digital signature published along with the plaintext message. It is worth noting that the private key should never be used again and the remaining 256 numbers from the pairs should be destroyed *(Lamport one-time signature)*.

The verification process is straightforward. The recipient calculates the hash of the message and then, for each bit of the hashed message we choose the corresponding hash from the public key (512 in number). In addition, the recipient hashes each number of the sender's private key which should correspond to the same sequence of hashed values with the recipients correctly chosen public key values. The security of this system derives by the decision of using the private key only once. Consequently, an adversary can only retrieve 50 percent of the private key which makes it impossible to forge a new valid signature.

Buchmann et al. [18] explained that in case we want to sign more than one messages, chaining can be introduced. The signer includes in the signed message a newly generated public key that is used to verify the next message received.

Witernitz described a one time signature (WOTS) which is more efficient than Lamport's. Specifically, the signature size and the keys are smaller [74]. However, OTSs are not suitable for large-scale use because they can be used only once.

Merkle introduced a new approach that combines Witernitz's OTS with binary trees (Merkle Signature Scheme). A binary tree is made of nodes. In our case each node represents the hash value of the concatenation of the child nodes. Each of the leaf nodes (lowest nodes in the tree hierarchy) contains a Witernitz's OTS which is used for signing. The first node in the hierarchy of the tree known as root node is the actual public key that can verify the OTSs contained in the leaf nodes [74].

In 2013, A. Hulsing improved the WOTS algorithm by making it more efficient without affecting its security level even when hash functions without collision resistance are used [75].

Currently two hash-based signature schemes are under evaluation for standardization. Specifically, the eXtended Merkle Signature Scheme (XMSS) [76] which is a stateful signature scheme, and Stateless Practical Hash-based Incredibly Nice Collision-resilient Signatures (SPHINCS) [77] which is as the name indicates a stateless signature scheme.

4) Code-based Cryptography: Code-based cryptography refers to cryptosystems that make use of error correcting codes. The algorithms are based on the difficulty of decoding linear codes and are considered robust to quantum attacks when the key sizes are increased by the factor of 4. Furthermore, Buchmann et al. [18] state that the best way to solve the decoding problem is to transform it to a Low-Weight-Code-World Problem (LWCWP) but solving a LWCWP in large dimensions is considered infeasible. It would be easier to comprehend the process of this scheme by using Buchmann's [18] concise explanation of McEliece's original code-based public-key encryption system. We define b as the security of our system and it is a power of 2. $n = 4b \lg b, \quad d = \lg n, \quad \text{and} \quad t = 0.5n/d$.

For example, if $b = 128$ then $n = 512 \log_2(128)$ which is equal to 3584. $d = 12$ and $t = 149$. The receiver's public key in this system is dtn matrix K with coefficients F_2. Messages to be encrypted should have exactly t bits set to 1 and for the encryption the message m is multiplied by K. The receiver generates a public key with a hidden Goppa code structure (error-correction code) that allows to decode the message with Patterson's algorithm, or even by faster algorithms. The code's generator matrix K is perturbated by two invertible matrices which are used to decrypt the ciphertext to obtain the message m.

As for any other class of cryptosystems, the practice of code-based cryptography is a trade-off between efficiency and security. McEliece's cryptosystem encryption and decryption process are fast with very low complexity, but it makes use of large public keys (100 kilobytes to several megabytes).

VI. Conclusion

In today's world, where information play a particularly important role, the transmission and the storage of data must be maximally secure. Quantum computers pose a significant risk to both conventional public key algorithms (such as RSA, ElGamal, ECC and DSA) and symmetric key algorithms (3DES, AES). Year by year it seems that we are getting closer to create a fully operational universal quantum computer that can utilize strong quantum algorithms such as Shor's algorithm and Grover's algorithm. The consequence of this technological advancement is the absolute collapse of the present public key algorithms that are considered secure, such as RSA and Elliptic Curve Cryptosystems. The answer on that threat is the introduction of cryptographic schemes resistant to quantum computing, such as quantum key distribution methods like the BB84 protocol, and mathematical-based solutions like lattice-based cryptography, hash-based signatures, and code-based cryptography.

Acknowledgment

This research is supported by the Research Council of Norway under the Grant No.: IKTPLUSS 247648 and 248030/O70 for Oslo Analytics and SWAN projects, respectively. This research is also part of the SecurityLab of the University of Oslo.

References

[1] M. Dušek, N. Lütkenhaus, and M. Hendrych, "Quantum cryptography," *Progress in Optics*, vol. 49, pp. 381–454, 2006.

[2] C. Paar and J. Pelzl, "Introduction to Public-Key Cryptography," in *Understanding Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 149–171.

[3] Z. Kirsch, "Quantum Computing: The Risk to Existing Encryption Methods," Ph.D. dissertation, Tufts University, Massachusetts, 2015, http://www.cs.tufts.edu/comp/116/archive/fall2015/zkirsch.pdf.

[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th ed. New York, NY, USA: Cambridge University Press, 2011.

[5] R. Jozsa, "Entanglement and Quantum Computation," in *Geometric Issues in the Foundations of Science*, S. Huggett, L. Mason, K. Tod, S. Tsou, and N. Woodhouse, Eds. Oxford University Press, July 1997.

[6] W. Tichy, "Is quantum computing for real?: An interview with catherine mcgeoch of d-wave systems," *Ubiquity*, vol. 2017, no. July, pp. 2:1–2:20, Jul. 2017. [Online]. Available: http://doi.acm.org/10.1145/3084688

[7] M. Soeken, T. Häner, and M. Roetteler, "Programming quantum computers using design automation," *arXiv preprint arXiv:1803.01022*, 2018.

[8] S. Bone and M. Castro, "A Brief History of Quantum Computing," *Surveys and Presentations in Information Systems Engineering (SURPRISE)*, vol. 4, no. 3, pp. 20–45, 1997, http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol4/spb3/.

[9] J. Muhonen and T. Dehollain, "Storing Quantum Information For 30 Seconds In a Nanoelectronic Device," *Nature Nanotechnology*, vol. 9, pp. 986–991, 2014.

[10] D-Wave, "Quantum Computing: How D-Wave Systems Work," http://www.dwavesys.com/our-company/meet-d-wave.

[11] L. S. Bishop, S. Bravyi, A. Cross, J. M. Gambetta, and J. Smolin, "Quantum volume," Technical report, 2017., Tech. Rep., 2017.

[12] M. Campagna and C. Xing, "Quantum Safe Cryptography and Security: An Introduction, Benefits, Enablers and Challenges," ETSI, Tech. Rep. 8, 2015.

[13] W. Buchanan and A. Woodward, "Will Quantum Computers be the End of Public Key Encryption?" *Journal of Cyber Security Technology*, vol. 1, no. 1, pp. 1–22, 2016.

[14] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, "NIST: Report on Post-Quantum Cryptography," NIST, Tech. Rep., 2016.

[15] P. W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, ser. SFCS '94. Washington, DC, USA: IEEE Computer Society, 1994, pp. 124–134.

[16] U. Vazirani, "On The Power of Quantum Computation," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 356, no. 1743, pp. 1759–1768, 1998.

[17] L. Grover, "A Fast Quantum Mechanical Algorithm For Database Search," Bell Labs, New Jersey, Tech. Rep., 1996.

[18] D. Bernstein, E. Dahmen, and Buch, *Introduction to Post-Quantum Cryptography*. Springer-Verlag Berlin Heidelberg, 2010.

[19] J. Proos and C. Zalka, "Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves," *Quantum Info. Comput.*, vol. 3, no. 4, pp. 317–344, 2003.

[20] National Security Agency, "National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information," NSA, Tech. Rep., 2003.

[21] G. Brassard, P. Høyer, and A. Tapp, *Quantum Cryptanalysis of Hash and Claw-Free Functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 163–169.

[22] D. Moody, "The ship has sailed: The nist post-quantum crypto competition." [Online]. Available: https://csrc.nist.gov/CSRC/media//Projects/Post-Quantum-Cryptography/documents/asiacrypt-2017-moody-pqc.pdf

[23] N. Koblitz and A. Menezes, "A riddle wrapped in an enigma," *IEEE Security Privacy*, vol. 14, no. 6, pp. 34–42, Nov 2016.

[24] C. H. Bennett and G. Brassard, "Quantum Cryptography: Public Key Distribution, and Coin-Tossing," in *Proc. 1984 IEEE International Conference on Computers, Systems, and Signal Processing*, no. 560, 1984, pp. 175–179.

[25] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental Quantum Cryptography," *Journal of Cryptology*, vol. 5, no. 1, pp. 3–28, 1992.

[26] E. Panarella, "Heisenberg uncertainty principle," in *Annales de la Fondation Louis de Broglie*, vol. 12, no. 2, 1987, pp. 165–193.

[27] H. Singh, D. Gupta, and A. Singh, "Quantum key distribution protocols: A review," *Journal of Computational Information Systems*, vol. 8, pp. 2839–2849, 2012.

[28] A. K. Ekert, "Quantum cryptography based on bell's theorem," *Physical review letters*, vol. 67, no. 6, p. 661, 1991.

[29] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf, M. Dušek, N. Lütkenhaus, and M. Peev, "The security of practical quantum key distribution," *Reviews of modern physics*, vol. 81, no. 3, p. 1301, 2009.

[30] C. H. Bennett, G. Brassard, and N. D. Mermin, "Quantum cryptography without bell's theorem," *Physical Review Letters*, vol. 68, no. 5, p. 557, 1992.

[31] D. Bohm, *Quantum theory*. Courier Corporation, 1951.

[32] J. F. Clauser, M. A. Horne, A. Shimony, and R. A. Holt, "Proposed experiment to test local hidden-variable theories," *Physical review letters*, vol. 23, no. 15, p. 880, 1969.

[33] V. Scarani, A. Acin, G. Ribordy, and N. Gisin, "Quantum cryptography protocols robust against photon number splitting attacks for weak laser pulse implementations," *Physical review letters*, vol. 92, no. 5, p. 057901, 2004.

[34] A. Acin, N. Gisin, and V. Scarani, "Coherent-pulse implementations of quantum cryptography protocols resistant to photon-number-splitting attacks," *Physical Review A*, vol. 69, no. 1, p. 012309, 2004.

[35] C. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, pp. 175–179, 1984.

[36] D. Bruß, "Optimal eavesdropping in quantum cryptography with six states," *Physical Review Letters*, vol. 81, no. 14, p. 3018, 1998.

[37] H. Bechmann-Pasquinucci and N. Gisin, "Incoherent and coherent eavesdropping in the six-state protocol of quantum cryptography," *Physical Review A*, vol. 59, no. 6, p. 4238, 1999.

[38] K. Tamaki and H.-K. Lo, "Unconditionally secure key distillation from multiphotons," *Physical Review A*, vol. 73, no. 1, p. 010302, 2006.

[39] B.-G. Englert, D. Kaszlikowski, H. K. Ng, W. K. Chua, J. Řeháček, and J. Anders, "Efficient and robust quantum key distribution with minimal state tomography," *arXiv preprint quant-ph/0412075*, 2004.

[40] C. H. Bennett, "Quantum cryptography using any two nonorthogonal states," *Physical review letters*, vol. 68, no. 21, p. 3121, 1992.

[41] N. J. Cerf, M. Levy, and G. Van Assche, "Quantum distribution of gaussian keys using squeezed states," *Physical Review A*, vol. 63, no. 5, p. 052311, 2001.

[42] F. Grosshans and P. Grangier, "Continuous variable quantum cryptography using coherent states," *Physical review letters*, vol. 88, no. 5, p. 057902, 2002.

[43] C. Weedbrook, A. M. Lance, W. P. Bowen, T. Symul, T. C. Ralph, and P. K. Lam, "Quantum cryptography without switching," *Physical review letters*, vol. 93, no. 17, p. 170504, 2004.

[44] J. Lodewyck, M. Bloch, R. García-Patrón, S. Fossier, E. Karpov, E. Diamanti, T. Debuisschert, N. J. Cerf, R. Tualle-Brouri, S. W. McLaughlin *et al.*, "Quantum key distribution over 25 km with an all-fiber continuous-variable system," *Physical Review A*, vol. 76, no. 4, p. 042305, 2007.

[45] C. Silberhorn, T. C. Ralph, N. Lütkenhaus, and G. Leuchs, "Continuous variable quantum cryptography: Beating the 3 db loss limit," *Physical review letters*, vol. 89, no. 16, p. 167901, 2002.

[46] K. Inoue, E. Waks, and Y. Yamamoto, "Differential phase shift quantum key distribution," *Physical Review Letters*, vol. 89, no. 3, p. 037902, 2002.

[47] ——, "Differential-phase-shift quantum key distribution using coherent light," *Physical Review A*, vol. 68, no. 2, p. 022317, 2003.

[48] N. Gisin, G. Ribordy, H. Zbinden, D. Stucki, N. Brunner, and V. Scarani, "Towards practical and fast quantum cryptography," *arXiv preprint quant-ph/0411022*, 2004.

[49] D. Stucki, N. Brunner, N. Gisin, V. Scarani, and H. Zbinden, "Fast and simple one-way quantum key distribution," *Applied Physics Letters*, vol. 87, no. 19, p. 194108, 2005.

[50] D. Stucki, C. Barreiro, S. Fasel, J.-D. Gautier, O. Gay, N. Gisin, R. Thew, Y. Thoma, P. Trinkler, F. Vannel *et al.*, "Continuous high speed coherent one-way quantum key distribution," *Optics express*, vol. 17, no. 16, pp. 13 326–13 334, 2009.

[51] D. Mayers, "Unconditional Security in Quantum Cryptography," *Journal of the ACM*, vol. 48, no. 3, pp. 351–406, 2001.

[52] C. Branciard, N. Gisin, B. Kraus, and V. Scarani, "Security of Two Quantum Cryptography Protocols Using The Same Four Qubit States," *Physical Review A*, vol. 72, no. 3, p. 032301, sep 2005.

[53] L. Lydersen, C. Wiechers, D. E. C. Wittmann, J. Skaar, and V. Makarov, "Hacking Commercial Quantum Cryptography Systems by Tailored Bright Illumination," *Nature Photonics*, pp. 686–689., October 2010.

[54] Z. Yuan, J. Dynes, and A. Shields, "Avoiding the Blinding Attack in QKD," *Nature Photonics*, vol. 4, pp. 800–801, December 2010.

[55] L. Lydersen, C. Wiechers, C. Wittmann, D. Elser, J. Skaar, and V. Makarov, "Avoiding the Blinding Attack in QKD," *Nature Photonics*, vol. 4, pp. 801–801, December 2010.

[56] V. Makarov, "Quantum Cryptography and Quantum Cryptanalysis," Ph.D. dissertation, Norwegian University of Science and Technology Faculty of Information Technology, NTNU, 2007, http://www.vad1.com/publications/phd-thesis-makarov-200703.pdf.

[57] G. Brassard, N. Lütkenhaus, T. Mor, and B. C. Sanders, "Security aspects of practical quantum cryptography," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2000, pp. 289–299.

[58] H.-K. Lo, X. Ma, and K. Chen, "Decoy state quantum key distribution," *Physical review letters*, vol. 94, no. 23, p. 230504, 2005.

[59] M. Haitjema, "A survey of the prominent quantum key distribution protocols," 2007.

[60] S. J. Lomonaco, J. Kauffman, and L. H, "Quantum Hidden Subgroup Problems: A Mathematical Perspective," *Quantum*, pp. 1–63., 2002.

[61] D. Micciancio, “Lattice-Based Cryptography,” in *Post-Quantum Cryptography*, 2009, no. 015848, pp. 147–192.

[62] J. Ding and B.-Y. Yang, “Multivariate Public Key Cryptography,” *Post-Quantum Cryptography*, pp. 193–241, 2009.

[63] C. Dods, N. P. Smart, and M. Stam, “Hash Based Digital Signature Schemes,” *Cryptography and Coding*, vol. 3796, pp. 96–115, 2005.

[64] R. Overbeck and N. Sendrier, “Code-based Cryptography,” in *Post-Quantum Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 95–145.

[65] M. Ajtai and C. Dwork, “A Public-Key Cryptosystem With Worst-Case/Average-Case Equivalence,” *Proceedings of The 29th Annual ACM Symposium on Theory of Computing - STOC ’97*, pp. 284–293., 1997.

[66] O. Goldreich, S. Goldwasser, and S. Halevi, “Public-Key Cryptosystems from Lattice Reduction Problems,” *Advances in Cryptology - {CRYPTO} ’97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, vol. 1294, pp. 112–131, 1997.

[67] J. Hoffstein, J. Pipher, and J. H. Silverman, “NTRU: A Ring-Based Public Key Cryptosystem,” *Algorithmic number theory*, pp. 267–288, 1998.

[68] P. Nguyen and J. Stern, *Cryptanalysis of the Ajtai-Dwork Cryptosystem*. Springer Berlin Heidelberg, 1998, pp. 223–242.

[69] P. Nguyen, “Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem,” *Advances in Cryptology - CRYPTO*, vol. 1666, pp. 288–304, 1999.

[70] P. S. Hirschhorn, J. Hoffstein, N. Howgrave-Graham, and W. Whyte, *Choosing NTRUEncrypt Parameters in Light of Combined Lattice Reduction and MITM Approaches*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 437–455.

[71] D. Stehle and R. Steinfeld, “Making NTRUEncrypt and NTRUSign as Secure as Standard Worst-Case Problems over Ideal Lattices,” Cryptology ePrint Archive, Report 2013/004, 2013.

[72] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, “NTRU Prime,” *IACR Cryptology ePrint Archive*, vol. 2016, p. 461, 2016.

[73] C. Tao, A. Diene, S. Tang, and J. Ding, “Simple Matrix Scheme for Encryption,” in *International Workshop on Post-Quantum Cryptography*. Springer, 2013, pp. 231–242.

[74] R. C. Merkle, *A Certified Digital Signature*. New York, NY: Springer New York, 1990, pp. 218–238.

[75] H. Andreas, *W-OTS+ –Shorter Signatures for Hash-Based Signature Schemes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 173–188.

[76] J. Buchmann, E. Dahmen, and A. Hülsing, “XMSS-a Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions,” *Post-Quantum Cryptography*, pp. 117–129, 2011.

[77] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O’Hearn, *SPHINCS: Practical Stateless Hash-Based Signatures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 368–397.