Radboud University Nijmegen

Faculty of Science

Post-quantum Hash-based Signatures for Multi-chain Blockchain Technologies

Master's Thesis Computing Science

Author:
Breus Blaauwendraad

Supervisors:
Dr. Zekeriya Erkin
Dr. Peter Schwabe

External Supervisors:
Oğuzhan Ersoy, MSc
Bart de Jong, MSc

December 2019

Acknowledgements

I would like to thank my supervisors Dr. Zekeriya Erkin and Oğuzhan Ersoy from Delft University of Technology, and Dr. Peter Schwabe from the Radboud University Nijmegen, for their advise and guidance during my research. I also am grateful to the people from the Accenture Security team in the Netherlands for allowing me to work with them, and in particular Bart de Jong for his supervision. Furthermore, I gratefully acknowledge the people with whom I had fruitful discussions, and those who tirelessly answered all the questions I have sent to them by mail: Dr. Andreas Hülsing, Dr. Bas Westerbaan, Tommy Koens, Wouter van der Linde, and Scott Fluhrer. In addition, I want to extend my sincere thanks to my friends Bart Veldhuizen and Emiel Bos for their practical suggestions on my writing. Finally, I want to thank my girlfriend for helping me during stressful moments and cheering me up when I felt down. Thank you all!

Abstract

The use of blockchain technology is becoming increasingly prevalent in the field of information technology. To mitigate the privacy and scalability concerns of the original blockchain design, blockchain systems consisting of multiple (block)chains emerged. The security of these systems relies to a large extent on cryptographic digital signatures for the authentication of transactions and blocks. However, the digital signatures typically used in blockchain are prone to the rising threat of quantum computers, as they rely on hardness assumptions which no longer hold in the presence of a sufficiently strong quantum computer.

To mitigate this threat, research has been conducted on post-quantum cryptography, aimed at being secure against quantum computers. One of the most promising solutions for post-quantum digital signatures are hash-based signatures, as they rely on the well-understood security properties of hash functions, and not on any hardness assumptions. A major drawback of hash-based signatures are the large signature sizes, in particular for stateless variants. Nonetheless, earlier work has shown that stateful hash-based signatures can be used effectively in combination with a blockchain.

In this work, research is conducted on stateful hash-based signatures for multi-chain blockchain technologies, and a digital signature scheme specifically designed for these systems is proposed. In addition, a standalone implementation of the scheme is built, and analyses on both the theoretical and practical performance of the scheme are conducted.

When using this scheme in combination with a blockchain, it results in smaller signatures and favorable performance features compared to state-of-the-art hash-based signature schemes with similar security properties, at the cost of two additional requirements for implementation: Signatures belonging to the same public key must be verified in a chronological order, and no preceding signature can be missing to verify a later signature. Alternatively, for each missing signature, the corresponding compressed one-time verification key must be added to a later signature.

These requirements however, correspond to the verification of blocks in a blockchain, which must be verified in a chronological order, and no intermediate blocks can be missing. Transactions in a blockchain are also validated in chronological order, but when transactions do not end up in the blockchain for any reason, their corresponding (missing) compressed verification keys must be added to a subsequent transaction.

Contents

List of Figures

List of Tables

List of Abbreviations

BPQS	Blockchained Post-quantum Signatures
BLT	Buldas, Laanoja, and Truu (cryptosystem)
CMA	Chosen Message Attack
CMSS	Chained Merkle Signature Scheme
CR	Collision Resistance
ECDSA	Elliptic Curve Digital Signature Algorithm
EdDSA	Edwards-curve Digital Signature Algorithm
eTCR	Extended Target Collision Resistance
ETCR	Enhanced Target Collision Resistance
EU-CMA	Existential Unforgeable under Chosen Message Attack
GMSS	Generalized Merkle Signature Scheme
HBS	Hash-Based Signature
HSS	Hierarchical Signature Scheme
IETF	Internet Engineering Task Force
IRTF	Internet Research Task Force
KMA	Known Message Attack
LM-OTS	Leighton-Micali One-Time Signature
LMS	Leighton-Micali Signatures
MBPQS	Multi-blockchain Post-quantum Signatures
MM-OW	Multi-function Multi-target One-Wayness
MM-SPR	Multi-function Multi-target Second-Preimage Resistant

MSS	Merkle's Signature Scheme
NIST	National Institute of Standards and Technology
OTS	One-Time Signature
OW	One-Wayness
PPT	Probabilistic Polynomial Time
PQ	Post-Quantum
PRF	Pseudo-Random Function
PRNG	Pseudo-Random Number Generator
QRL	Quantum Resistant Ledger
QROM	Quantum Random Oracle Model
RFC	Request For Comments
RMA	Random Message Attack
RO(M)	Random Oracle (Model)
RSA	Rivest-Shamir-Adleman (cryptosystem)
SHA	Secure Hash Algorithms
SM	Standard Model
SM-OW	Single-function Multi-target One-Wayness
SM-SPR	Single-function Multi-target Second-Preimage Resistant
SPHINCS	Stateless Practical Hash-based Incredibly Nice Signatures
SPR	Second-Preimage Resistant
SPR-MSS	Second-Preimage Resistant Merkle Signature Scheme
SU(F)-CMA	Strongly Unforgeable under Chosen Message Attack
TCR	Target Collision Resistance
WOTS(-T)	Winternitz One-Time Signature (Tightened security)
XMSS(-T)$^{(MT)}$	eXtended Merkle Signature Scheme (Tightened security) (Multi-Tree)
XNYSS	eXtended Naor-Yung Signature Scheme

Chapter 1

Introduction

In this chapter, the topics and motivations for this thesis are introduced. First, the context of the thesis topic is outlined, showing the relevancy of this work, by introducing the concept of blockchain and considering a possible future threat related to it. Subsequently, existing work aimed at mitigating the aforementioned threat is examined, as well as how this thesis contributes to further solving this problem. Lastly, the structure of the thesis is discussed.

1.1 Blockchain

Blockchain (a distributed ledger technology) is increasingly gaining traction in a variety of sectors such as online banking, supply chain management, identity and access management, and smart energy grids [61]. Experts predict that by 2027 more than 10% of the global gross domestic product will be stored on blockchains [37].

In this thesis, our definition of blockchain is a collectively maintained, cryptographically secured, append-only ledger consisting of the transactional history of participants in a peer-to-peer network, where a transaction is a report of proceedings or a smart contract. A smart contract is an automated, self-enforcing contract executing pre-defined code on the blockchain.

One or more transactions are combined into a set; called a block. An append-only chain of blocks is created using a linked list of blocks, where each block contains the hash digest of its preceding block as reference. The append-only property of the chain is regulated by the second-preimage resistance of the hash function, which means that it is infeasible to find a different valid block resulting in the same hash digest. Therefore, any change in the content of a block invalidates the succeeding blocks. In other words, whilst blocks can be appended to the ledger, once a successor block is added, they cannot be altered.

An instance of the blockchain is held by each peer in the network and an agreement on the current state of this shared ledger is reached via a consensus algorithm. Essentially, this algorithm defines the set of rules for the decision-making process in the network. Most importantly, it is used to determine the next block to be appended to the chain.

The transactions in the blockchain are authenticated by the involved peers with a (cryptographic) digital signature. This digital signature provides a way to ensure that transactions are approved by the rightful peers, making it one of the key components of the blockchain technology.

1.2 Post-quantum Digital Signatures

Digital signature schemes typically used in blockchain technology are RSA [30], ECDSA [54], and EdDSA [8, 93, 92, 59]. The security of these schemes relies on the hardness assumptions of the RSA-problem and the discrete logarithm problem, respectively [30, 54, 8]. For conventional (i.e. non-quantum) computers, no known algorithm exists to solve these problems in polynomial-time, that is, in an efficient way. However, these hardness assumptions might become obsolete in the future.

In 1994, Peter Shor invented a quantum algorithm which, in combination with a sufficiently powerful universal quantum computer, can be used to effectively solve the aforementioned hard problems [78, 66, 27]. As far as publicly known, such a machine does not exist yet, but major effort is being put into its development, and considerable progress has been made in recent years. In 2017, IBM announced a quantum computer that handles 50 quantum bits, or *qubits*, enabling computations which are extremely difficult to simulate with conventional computers [60]. At the start of 2018, Intel's 49-qubit quantum computer Tangle Lake was unveiled and on March 5 of that year, Google presented its 72 qubit quantum computer

Bristlecone [46, 58]. On June 1 2019, a joint project between Microsoft and Qutech, founded by TNO and TU Delft, was launched with the "aim to build topological qubits into a working quantum computer" [74]. Such qubits are more stable, mitigating one of the major issues of qubit scaling [91]. Hence, this effort might accelerate progress towards a quantum computer which is able to break current public key cryptography. Recently, in October 2019, Google claimed to have reached quantum supremacy, solving 10,000 years of classical computation in merely 200 seconds [3]. Even though the claim is not undisputed, the project is considered an excellent demonstration of the progress in this field [75].

According to Federov et al., the majority of scientists believe that a universal quantum computer is required to break current public-key standards, although some believe that specialized quantum (annealing) computers can be used as well [35]. The latter would further exacerbate the quantum-threat for classical public-key cryptography. This claim is supported by research where simulations of such a machine are used to solve small instances of the integer factorization problem, which is closely related to the RSA-problem [76]. Moreover, it has been shown that the 2000-qubit quantum annealing computer of D-wave can be used for efficient integer factorization [53].

In an attempt to tackle the aforementioned quantum-threats, research is being conducted on post-quantum (PQ) cryptography [7, 25]. This form of cryptography, based on classical computational mechanisms, is aimed to be secure in the presence of strong quantum computers. Current PQ digital signatures use constructions based on lattices, supersingular isogenies, codes, multivariate equations, or hash functions [7, 10, 18, 90].

1.3 Hash-based Signatures

In this thesis, research is conducted on the applicability of (post-quantum) hash-based signatures (HBSs) for blockchain. An HBS scheme combines a hash-based one-time signature (OTS) scheme, of which the keys can be used only once to guarantee its security, with an authentication structure to overcome this limitation [7]. The authentication structure is used to associate multiple OTS keys with one public key, transforming the OTS scheme in a many-time signature scheme.

One of the main advantages of HBS schemes is that their security properties are well-understood, and do not rely on number-theoretic or structured hardness assumptions [16]. Furthermore, software and hardware dedicated to hashing is available on a large scale, providing an infrastructure for fast implementation. However, HBS schemes come with relatively large signature sizes and long key generation times compared to conventional digital signatures [47].

There exist two types of HBS schemes: stateful and stateless. In the stateful variants, the signature sizes are smaller, but the signer needs to keep track of the state which specifies the (un)used OTS keys. In stateless HBS schemes, the set of usable OTS keys is sufficiently large such that the probability of picking the same key more than once is provably negligible, eliminating the need to keep track of the key state. However, this approach results in signature sizes 15-20 times larger compared to stateful variants for the same security levels [9].

Typically, many signatures are stored on a blockchain, as each transaction is signed by

one or multiple peers [92]. Therefore, larger signature sizes result in increased storage and network throughput requirements for the blockchain. Considering the already large signature sizes of HBSs in general, this research focuses on stateful HBS schemes.

The practical usability of stateful HBS schemes for blockchain technology has already been demonstrated in the Quantum Resistant Ledger (QRL) cryptocurrency [88]. Furthermore, two stateful HBS schemes specifically designed for blockchain have been proposed [86, 23]. These two schemes demonstrate that the blockchain structure can be leveraged to reduce the signature sizes compared to universally applicable HBS schemes. However, the aforementioned schemes are focused on single-chain blockchains, while several blockchain frameworks use a multi-chain design to increase privacy and scalability of the system [44, 1, 22]. Using these schemes in a multi-chain blockchain would result in multiple public keys for each user, or increased signature sizes. In the first proposal, Blockchained Post-Quantum Signatures (BPQS), the idea of combining multiple HBS schemes to sign messages in multiple blockchains is briefly mentioned, but not further addressed [23].

1.4 Contribution and Scope

In this thesis, research is conducted on the usability of HBSs to secure multi-chain blockchain technology against the quantum-threat. Therefore, an HBS scheme design, tailored for multi-chain blockchain technology, is proposed. This scheme, based on BPQS, shares the strong security properties of XMSS-T. Furthermore, the scheme can be used to sign a virtually unlimited number of signatures, and provides smaller signatures and better performance than XMSS-T^{MT} when used in combination with a blockchain. To analyze the practical performance of the scheme, a proof-of-concept implementation of the proposed scheme is presented. Since the scheme is designed for a specific type of blockchain technology, and not aimed at a framework-specific blockchain technology, the proof-of-concept is a standalone implementation of the signature scheme. This can be used as a reference for anyone implementing the scheme in a blockchain framework.

1.5 Thesis Outline

In Chapter 2, background information is given on primitives used in this thesis, such as cryptographic security notions, hash functions, digital signatures, and blockchain technology. In Chapter 3, literature related to this thesis is discussed, presenting an overview of the existing work. Chapter 4 presents a detailed description of the proposed digital signature scheme, and subsequently, its security is considered. In Chapter 5, the performance of the scheme is analyzed through both a complexity analysis, and performance measurements against several benchmarks. In Chapter 6, a discussion on the research is presented, including future work for our design, and finally, a conclusion is drawn.

Chapter 2

Background Information

The following chapter covers the background information which is considered to be known to the reader throughout the rest of the thesis. First, the cryptographic primitives used in the construction of the digital signature scheme proposed in Chapter 4 are explained. Afterwards, the concept of blockchain technology, and its different variants are discussed to form a precise definition of the systems targeted by the proposed scheme. Lastly, various building blocks for the design are presented. This chapter, or parts of it, may be skipped if the reader is familiar with the discussed concepts.

2.1 Cryptographic Primitives

2.1.1 Hash Functions

A hash function is a deterministic one-way function mapping an arbitrary-length input to a fixed-length output, called a *hash digest*, *hash value*, or *hash*. Hash functions serve as building blocks for cryptographic structures and authenticated data structures. Whenever hash functions are mentioned in this thesis, generally a keyed hash function is implied. A keyed hash function is a member of a collection of related hash functions, called a hash function family. The key defines a particular hash function instance from the family.

The security of hash functions is divided in three general properties: *preimage resistance* – also called *one-wayness (OW)* –, *second-preimage resistance (SPR)*, and *collision resistance (CR)* [80]. Informally, preimage resistance means that given a hash digest, it is infeasible to find an input resulting in that particular hash digest. Second-preimage resistance implies that given an input, it is infeasible to find another input with the same hash digest. Collision resistance means that it is infeasible to find any two different inputs resulting in equal hash digests; a collision. Here, infeasible means that there exists no publicly known algorithm that can be used to efficiently compute the value(s) required to falsify the security properties.

In [6], Bellare and Rogaway introduced the notion of *target collision resistance (TCR)*. Instead of finding any two messages resulting in a collision, the adversary must find a collision for a fixed target message which he can choose upfront. After choosing the target message, the hash function key is provided to the adversary, defining a specific hash function instance from the hash function family. The adversary wins if he can present a message, different from the target message, resulting in a hash collision with the target message for this hash function instance.

Later, the concept of *enhanced target collision resistance (ETCR)* was introduced in [42], followed by the slightly changed notion of *extended target collision resistance (eTCR)* in [50]. While the setups of ETCR and eTCR are identical to TCR, additionally, the adversary may present a different hash key for which he found a collision with the target message. However, the hash function key used with the target message may not be changed. ETCR differs from eTCR in that collisions may be with the same message as the target, as long as the hash key is different. In eTCR, the found message must be different from the target message. For the rest of this thesis, only eTCR is relevant.

In what follows, formal definitions of a hash function family, and of the security properties of hash functions, are given.

Definition 2.1.1 *Hash function family. If $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a hash function family where $\mathcal{K}$ is the finite set of m-bit keys, $\mathcal{X}$ is the set of all possible bit-strings, and Y is the finite set of possible n-bit outputs. Then, $\forall h_k \in H : \{0,1\}^* \to \{0,1\}^n$, the following security properties are defined:*

Preimage resistance a.k.a. One-wayness (OW): *Given a hash digest y, it is infeasible to find any x s.t. $h_k(x) = y$.*

Second-preimage resistance (SPR): *Given x, it is infeasible to find any $x' \neq x$ s.t. $h_k(x) = h_k(x')$.*

Collision resistance (CR): *It is infeasible to find any pair* $(x, x') \wedge x \neq x'$ *s.t.* $h_k(x) = h_k(x')$.

Target Collision resistance (TCR): *For* x *chosen, and* k *given after choosing, it is infeasible to find any* $x' \neq x$ *s.t.* $h_k(x) = h_k(x')$.

Extended Target Collision resistance (eTCR): *For* x *chosen, and* k *given after choosing, it is infeasible to find any* $x' \neq x$ *and (possibly new)* k' *s.t.* $h_k(x) = h_{k'}(x')$

In these definitions, infeasible implies that for every probabilistic polynomial time (PPT) algorithm the probability of finding the sought-after value is $\leq \epsilon$*, where* ϵ *is negligible.*

2.1.2 Multi-target Security for Hash Functions

The aforementioned security properties bear the implicit assumption that the hash function is used only once. However, for the construction of hash-based signature (HBS) schemes, many hash function invocations are required. Whenever the same hash function is used multiple times, an attacker can target multiple instances at once. For example, the complexity of inverting a hash function with output length n for generic attacks is $\mathcal{O}(2^n)$, as every input has probability $\frac{1}{2^n}$ to result in the targeted output, assuming the hash function behaves like a random function. However, when an attacker has t possible targets, the complexity becomes $\mathcal{O}(\frac{2^n}{t})$, as every input now has a $\frac{t}{2^n}$ probability to result in the targeted output. Consequently, Hülsing, Rijneveld, and Song formalized the multi-target security properties for hash functions in [50]. These security properties are considered for both a single hash function instance, as well as multiple hash function instances. Since extended target collision resistance is inherently multi-function, no single-function definition exists. In the definitions, the attacker is assumed to have t possible targets where $1 \leq i \leq t$. Subsequently, the multi-target security properties of hash functions are defined as:

Definition 2.1.2 *Multi-target security properties of hash functions [50].*

Single-function, multi-target OW (SM-OW): *Given* $k \xleftarrow{\$} \mathcal{K}$ *and* $\{y_1, ..., y_t\}$*, where* $y_i = h_k(x_i)$*, it is infeasible to find any* x' *s.t.* $\exists y_i = h_k(x')$

Multi-function, multi-target OW (MM-OW): *Given* $\{k_1, ..., k_t\} \xleftarrow{\$} \mathcal{K}$ *and* $\{y_1, ..., y_t\}$*, where* $y_i = h_{k_i}(x_i)$*, it is infeasible to find any* (k, x')
s.t. $\exists y_i = h_{k=k_i}(x')$.

Single-function, multi-target SPR (SM-SPR): *Given* $k \xleftarrow{\$} \mathcal{K}$ *and* $\{x_1, ..., x_t\}$*, it is infeasible to find any* x' *s.t.* $\exists x_i \neq x' \wedge h_k(x_i) = h_k(x')$.

Multi-function, multi-target SPR (MM-SPR): *Given* $\{k_1, ..., k_t\} \xleftarrow{\$} \mathcal{K}$ *and* $\{x_1, ..., x_t\}$*, it is infeasible to find any* (k, x') *s.t.* $\exists x_i \neq x' \wedge h_{k_i}(x_i) = h_{k=k_i}(x')$.

Multi-target eTCR (MM-eTCR): *Given* $\{(k_1, x_1), ..., (k_t, x_t)\}$ *where* (k_i, x_i) *is an eTCR challenge, it is infeasible to find any* (x', ik') *s.t.* $\exists (x_i, k_i) \wedge x_i \neq x' \wedge h_{k_i}(x_i) = h_{k'}(x')$.

Again, infeasible means that for every probabilistic polynomial time (PPT) algorithm the probability of finding the sought-after value $\leq \epsilon$.

Hülsing et al. show that the security levels against multi-function multi-target attacks for classical and quantum computers are equal to the security levels for the corresponding single-target property [50]. In other words, the security level of OW is equal to MM-OW, and that of SPR is equal to MM-SPR. The security against single-function multi-target attacks decreases linearly in the number of targets (t). Since eTCR is by definition multi-function, its security decreases linearly for every additional eTCR challenge presented simultaneously to the adversary.

2.1.3 Pseudo-random Functions

Besides hash functions, this thesis uses pseudo-random functions (PRFs). Given a secret input seed, a PRF returns a pseudorandomly generated sequence of values from its output space. A PRF is deterministic; the same input to the PRF always results in the same output. Because computers are deterministic, creating a PRF with seemingly random behavior is non-trivial. Nonetheless, PRFs are vital for the construction of secure cryptographic primitives. For example, a secret (e.g. key) must be unpredictable (i.e. random) to guarantee its secrecy.

Intuitively, a PRF must be indistinguishable from a function chosen randomly from all possible functions with the same domain and value range. That is, there exists no efficient algorithm that can effectively determine whether a PRF or any other function with the same domain and range was used, based on an input and corresponding output. A PRF family is a collection of PRFs with similar properties. A key defines the particular instance from the PRF family. More formally, a PRF family is defined as:

Definition 2.1.3 *Pseudo-random function (PRF) family. A function family F is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{K}$ is the set of keys, $\mathcal{X}$ is the set of inputs, and $\mathcal{Y}$ is the set of possible outputs. Then, $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is called a PRF family if it possesses the following properties:*

1. *$\forall k \in \mathcal{K} \wedge x \in \mathcal{X}$, there exists an efficient algorithm to compute $F(k, x) = f_k(x)$*
2. *$\forall f_k \in F$ is indistinguishable from a function of the set $\mathcal{S}$ of all existing functions with the same output space. Hence, for any probabilistic polynomial time adversary $\mathcal{A}$:*

$$|\Pr(f \xleftarrow{\$} \mathcal{S} : \mathcal{A}^{f \in \mathcal{S}}) - \Pr(f \xleftarrow{\$} F : \mathcal{A}^{f \in F})| < \epsilon$$

2.1.4 Digital Signatures

In the real world, signatures are used to prove one's identity, and one's agreement to signed documents. The security of such a signature relies on the fact that it is hard to falsify (copy) someone's unique signature. In the digital world, however, it is trivial to copy any data and therefore, mathematical structures are employed to construct (cryptographic) digital signatures [56]. Digital signatures provide strong computational security guarantees and have as an additional benefit that the signatures enforces data integrity as well. In other

words, any change to the signed message renders the signature invalid which can be easily checked by a verifier.

A digital signature scheme consists of three algorithms: $KeyGen$, S, and V, for key generation, signing, and verification, respectively. $KeyGen$ generates a key pair consisting of a private key for signing and a public key for verification. Accordingly, these keys are often referred to as 'signing key' and 'verification key'. A signature for a message is generated using the signing algorithm S with the message and private key as arguments. Subsequently, the verifier uses the algorithm V with the verification key, message, and signature as arguments to verify the validity of the signature over the message. The formal definition is as follows:

Definition 2.1.4 *Digital signature scheme [39]. A digital signature scheme is a triplet of PPT algorithms* $(KeyGen, S, V)$ *and a message (input) space* $\mathcal{M}$*, where*

- $KeyGen(1^n) = (pk, sk)$*, takes a security parameter* n *as argument and generates a verification/signing key pair* (pk, sk)*.*
- $S(m, sk) = \sigma$*, takes a message* $m \in \mathcal{M}$ *and private key* sk *and outputs a corresponding signature* σ*.*
- $V(\sigma, m, pk) = \{accept, deny\}$*, takes a signature* σ*, message* $m \in \mathcal{M}$*, and verification key* pk *and outputs whether the signature is accepted or denied for the given message and key.*

Additionally, a digital signature has the following ***correctness*** *property:*

$$Pr[(sk, pk) \leftarrow KeyGen(1^n) : V(S(m, sk), m, pk) = 1] = 1.$$

2.1.5 Security Definitions of Digital Signatures

To prevent ambiguity when discussing the security of digital signatures, exact definitions are adopted. The first security property of digital signatures relevant for this thesis is *forward security*. This means that the unforgeability of historical signatures is still preserved when an attacker finds the signing key [52]. For example, if the signing key changes over time, it will not be valid to forge historic signatures. Furthermore, the security of a digital signature is defined as a combination of an *attack model* and an *attack goal*. Consequently, a digital signature can be defined as being secure against a certain attack goal in a particular attack model. These definitions are taken from Goldwasser et al. [39] which were later expanded and formalized by Katz in [56]. The attack goals are what an attacker can achieve, which are defined, according to [56, 39], as follows:

Total Break: The attacker can compute the signing key and forge any signature on any message.

Universal Forgery: The attacker can create valid signatures for any message with a self-constructed efficient signing algorithm.

Selective Forgery: The attacker can forge a signature for a particular message chosen by the attacker before the attack.

Existential Forgery: The attacker can create at least one valid pair, consisting of a message and corresponding signature, for a message not earlier signed by a legitimate signer.

Strong Existential Forgery: The attacker can create at least one valid pair, consisting of a message and corresponding signature, for a message which may have been signed by a legitimate signer before. In that case, the signature must be different than the legitimate signature.

These attack goals are ordered from strongest (hardest) to weakest (easiest). The total break is the strongest goal possible, and a strong existential forgery the weakest. The stronger the digital signature scheme, the weaker the attack goals it can resist. Intuitively, if even relatively weak (easy to execute) forgeries are impossible, the digital signature is clearly very strong.

Subsequently, an attack model defines the powers the attacker has before its attempt to break the digital signature. These are divided in two main categories:

Key-only Attack: In this type of attack, the attacker only has access to the verification key.

Message Attack: In this type of attack, the attacker has access to both the verification key and a set of messages with their corresponding signatures. Depending on the type of message attack, these messages can possibly be chosen by the attacker.

The type of message attack models are specified as follows:

Known Message Attack (KMA): The attacker can observe the signatures of a set of non-chosen messages.

Random Message Attack (RMA): The attacker can query message/signature pairs that are random to the attacker.

Generic Chosen Message Attack (CMA): The attacker can observe the signatures of a fixed set of chosen messages.

Directed CMA: This type of attack is similar to the previous one, except that the verification key is known to the attacker before creating the set of chosen messages.

Adaptive CMA: The attacker may, besides everything in the directed chosen message attack, also query the signatures of chosen messages depending on previously received signatures of chosen messages.

These models are ordered from weaker to stronger in terms of security: The more options the attacker has, the stronger a digital signature needs to be to resist an attack. Furthermore, it can be specified how many messages and corresponding signatures an attacker may observe. For example, in 'One-time CMA', the attacker can request a signature on a single message before attempting to forge a signature.

Both the models and goals are often abbreviated when the security of a digital signature is described in the literature. The goals are abbreviated as the first character of the goal, followed by UF or U for *unforgeable*. Furthermore, adaptive and existential are sometimes implied. For example, Strongly (Existential) Unforgeable under (Adaptive) Chosen Message Attack might be abbreviated as SU-CMA or SUF-CMA.

2.1.6 One-time Signatures

A one-time signature (OTS) scheme is a digital signature scheme in which the signing key can be used only once to guarantee its security. More specifically, for every time an OTS signing key is reused, the security level of the corresponding signatures decreases. More generally, in an x-time signature scheme, the signing key can be used a maximum of x times to generate a signature before the security of the scheme is reduced.

The OTS scheme adopted in the digital signature scheme proposed in Chapter 4 is the Winternitz One Time Signatures with Tightened security (WOTS-T) scheme [51]. Advantages of a WOTS scheme are the small verification key size and a flexible trade-off between signature time generation and signature size. Furthermore, a WOTS-T verification key can be computed from its corresponding signature. Therefore, the signature does not need to contain the key, reducing the signature size in the Merkle signature scheme. This makes a WOTS scheme the most suitable OTS scheme to combine with a Merkle tree authentication structure [13]. Since WOTS-T is the most recent WOTS scheme with increased security and reduced signature sizes compared to older versions for the same security levels, this exact variant is used in our design.

The construction of this WOTS-T was an iterative process covering multiple schemes, where each OTS scheme improved on its predecessor. In this section, the OTS schemes leading up to the WOTS-T scheme are briefly discussed. Other OTS schemes, such as Bleichenbacher-Maurer OTS [11], the BiBa OTS [77], HORS [79], and LM-OTS [64] are not be covered.

Lamport OTS

The earliest described OTS is a digital signature based on one-way functions, invented by Lamport [63, 31]. The only prerequisite to construct such an OTS is the existence of a secure one-way (hash) function. To give the reader an intuitive understanding of OTSs based on hash functions, a brief description of the scheme is provided. For this, the hash-function $h : \{0,1\}^* \rightarrow \{0,1\}^n$ serves as the one-way function.

Key generation. To generate a key to sign an n-bit message, a random pair of values $(x_{i,0}, x_{i,1})$ is chosen for each $0 < i < n$. The corresponding verification key is computed by separately hashing both of the values of the signing key pairs, i.e. $(y_{i,0}, y_{i,1}) = (h(x_{i,0}), h(x_{i,1}))$ for each $0 < i < n$. Thus, the resulting key pair consists of two $2n$-length sets: one for the signing key and one for the verification key.

Signing. The signature is constructed by revealing the preimages of the verification key corresponding to that message bit value. For example, if the $i-$th bit of the message is a 1, $\sigma_i = x_{i,1}$ is published as part of the signature. This is done for every bit in the message, resulting in $\sigma = \{\sigma_0, ..., \sigma_{n-1}\}$, where $\sigma_i = x_{i,m_i}$.

Verification. To verify the signature, one checks whether the hash digest of the signature value equals to the corresponding value of the verification key. In other words, the verifier checks if $h(\sigma_i) = y_{i,m_i}$ holds for each value in the signature $0 < i < n$. If it does, the signature is correct.

Minor improvements. Several (minor) improvements for Lamport's OTS were published since its invention. Merkle describes a method in which only the 1-bits of a message and a checksum of the amount of 0-bits are signed. This results in a signature size reduction up to almost a half, based on the number of 0-bits in the original message [69]. In [34], an online/offline variant of one-time signatures is proposed.

WOTS

In 1979, Merkle proposed the Winternitz One-time Signature (WOTS) scheme, named after Robert Winternitz, who gave him the idea. The scheme can be used to simultaneously sign multiple bits in a hash-based OTS, opposed to the bit-for-bit signing in Lamport's OTS. The scheme was first described in [69], and the first detailed description of the scheme can be found in [32]. WOTS provides a time/space trade-off for the signature size and computational effort of the algorithms. Therefore, the signature sizes in Lamport's OTS can be reduced in exchange for an increase in computational effort, mitigating the main drawback of Lamport's OTS.

WOTS+

WOTS+ is a WOTS variant with lowered security requirements for the underlying hash function, proposed by Hülsing in 2013 [47]. Instead of a collision resistant hash function, WOTS+ provably requires a second-preimage resistant one to be secure. This is achieved by XORing the input to each hash function iteration with an element generated using a PRF, called the bitmask. The seed for the PRF is generated during key generation and is added to the verification key. Applying these changes to WOTS, the resulting scheme is provably unaffected by the birthday paradox opposed to the original version. Ergo, using a hash function with half the output length compared to the original scheme results in the same security level. Furthermore, Hülsing presents a proof that the scheme is SU-CMA if the hash function is second-preimage resistant, one-way, and undetectable.

WOTS-T

In 2016, Hülsing et al. presented WOTS-T [51], a multi-target resistant variant of WOTS+. To prevent multi-target attacks, the bitmasks and keys in WOTS+ change in every hash

function invocation. This changes the multi-target security level from SM-OW to MM-OW, which corresponds to the OW security estimate in the original WOTS+ paper. The bitmasks and keys are deterministically generated using a seeded PRF to avoid larger verification keys. The adapted version, WOTS-T, was standardized, replacing WOTS+ in RFC 8391 in 2018 [48]. This OTS is used in the digital signature scheme proposed in Chapter 4. Therefore, a detailed description of it is given in Section 2.3.2.

2.1.7 Merkle Trees

To associate multiple OTS verification keys with a single public key, Merkle trees can be used [7]. A Merkle tree is a perfect binary hash tree of height H, used to verify the membership of data elements to a set.

To clarify how a Merkle tree can be used to transform an OTS scheme in a many-time signature scheme, consider the illustration of a Merkle tree in Figure 2.1. To associate multiple OTS verification keys with a single public key, first, 2^H OTS keys pairs are created, consisting of a signing key X_i, and verification key Y_i. Subsequently, these verification keys are hashed, denoted as h_i, and these hash digests become the leaf nodes of the Merkle tree. Then, the concatenation of two adjacent nodes is hashed and becomes their parent node. This is done recursively until there is only one node left, called the root node. This node embeds all OTS keys and becomes the public key. In each signature, a set of nodes is added which is used to authenticate that the verification key belongs to this public key. This set of nodes is called the authentication path. To illustrate this process, consider again Figure 2.1, where the signer used the signing key X_1, corresponding to Y_1, to sign a message. The signature includes the OTS signature, the corresponding OTS verification key, and the authentication path, which is (h_2, h_{10}, h_{14}) for Y_1. Now, the verifier can compute h_1 from the verification key Y_1 by hashing it. Subsequently, h_9 can be computed by hashing the concatenation of h_1 and h_2, where h_2 is taken from the authentication path. Accordingly, h_{13} can be computed by hashing the concatenation of h_9 and h_{10}. Finally, the Merkle root hash, h_{15}, can be computed, and the verifier checks whether it equals to the public key of the signer. If it does, the verifier knows that the signature belongs to this public key. Accordingly, a many-time signature scheme can be created from a hash-based OTS scheme. The resulting signature scheme is called a 'hash-based signature (HBS) scheme'.

In the use case for a Merkle tree described above, the data elements $(Y_1, ..., Y_8)$ represent OTS verification keys. However, these elements can be any type of data. For example, Merkle trees are also used to compute a Merkle root hash from a transactions list in blockchain technologies. In that case, the data elements are the hash digests of transactions. In general, a Merkle trees provide a fairly cheap method to verify the membership of data elements to a set. Authenticating whether a data elements belongs to the tree requires computing a number of hashes logarithmic to the number of leaf nodes of the tree. Furthermore, only a number of data elements logarithmic to the number of leaf nodes needs to be provided to the verifier as authentication path.

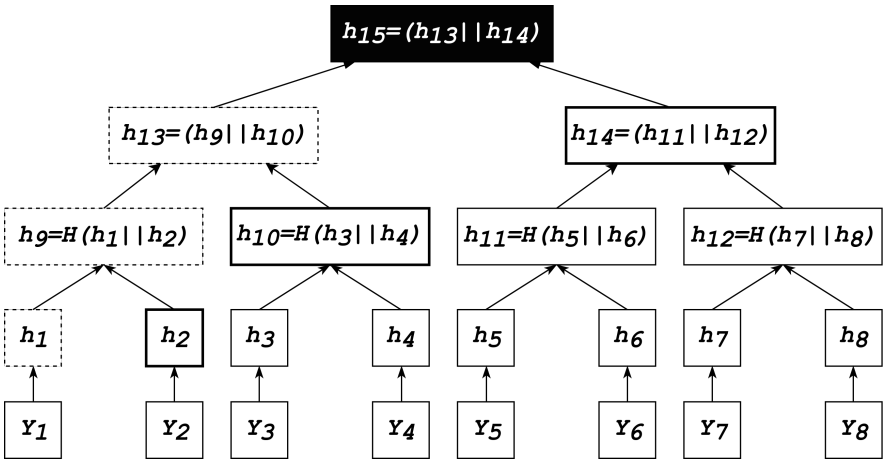

Figure 2.1: A Merkle tree with 2^3 OTS keys including the authentication path (h_2, h_{10}, h_{14}) for the verification key Y_1. The nodes with dashed borders are computed by the verifier.

2.2 Blockchain

In the introduction, an abridged description of blockchain technology is given. Because the workings of blockchain are regularly mentioned in the rest of this thesis, a more extensive description is provided in this section, in particular on the concepts most relevant here, being the data structure and the way cryptography is used in blockchain. Furthermore, Section 2.2.2 describes the different types of blockchain.

2.2.1 Blockchain Technology

Definition 2.2.1 *Blockchain. A blockchain is a collectively maintained, cryptographically secured, append-only ledger consisting of the transactional history of peers in a peer-to-peer network, where a transaction is a report of proceedings or a smart contract.*

First of all, a blockchain is a virtual ledger maintained by a network of peers. Each peer holds an instance of the blockchain locally, and communicates via interoperable client software to other peers in the peer-to-peer network to update this ledger in a synchronized way. Using this client, peers create transactions, which they digitally sign before broadcasting them to the network. This broadcasting can be done by either sending them to every peer in the network, or using a gossip protocol to mitigate network burden. Using a gossip protocol, the transaction is sent to a number of peers in the network, which further distribute the transaction throughout the network in the same way.

Using a consensus protocol, a leader is determined who bears the responsibility of ordering the transactions and adding them in blocks. Optionally, the leader must also either validate the transactions and add only the valid transactions, or add every transaction to a block and mark its validity accordingly. A block in a basic blockchain is depicted in Figure 2.2. The basic block fields are a hash of the current block fields, the hash of the previous block,

a timestamp, a list of transactions, and the Merkle root hash of these transactions. Actual implementations can have additional block fields to extend the functionality of the system. Once the block is created, the block writer optionally signs it, and broadcasts it to the network. Each peer holds a copy of the blockchain to which this block is added after its signature, or proof-of-work, is verified by the peer.

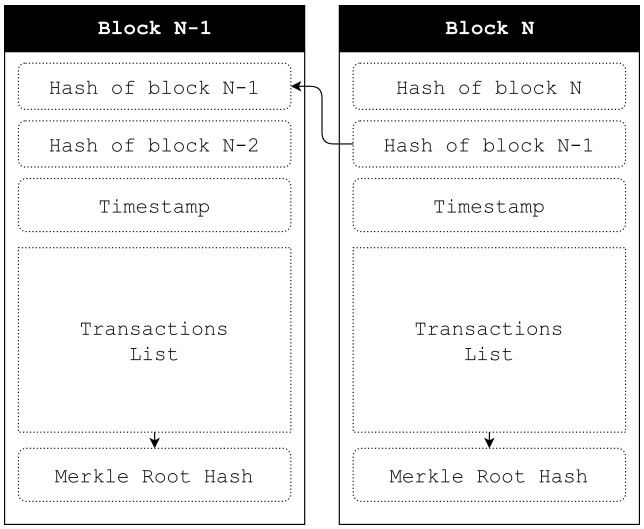

Figure 2.2: Part of a basic blockchain structure consisting of two blocks. The Merkle root hash is calculated from the transactions list.

2.2.2 Blockchain Classification

Blockchain is a particular type of Distributed Ledger Technology (DLT) and likewise, different types of blockchain technology exist. Nowadays, the original concept as proposed in the Bitcoin paper [71] represents merely one category of blockchain. Therefore, the working definition of blockchain (Definition. 2.2.1) is a rather broad one. Nonetheless, the digital signature scheme proposed in this thesis targets a specific type of blockchain mainly used in enterprise applications [92]. These blockchains have a multi-chain structure and accordingly, our proposed digital signature scheme in Chapter 4 is called Multi-Blockchain Post-Quantum Signatures (MBPQS). To form a more precise definition of the targeted blockchains, the classification of blockchain technology according to [92] is presented. The different consensus algorithms are not considered because in theory the applicability of MBPQS is independent from the consensus algorithm.

Classification

Blockchain technology can be categorized according to the centralization of the decision-making process in the network, and whether permission is required to join the network. In this regard, three main categories exist [92]:

Public blockchain: Accessible for anyone (permissionless), consensus is reached by all the miners (which anyone can be) and hence, the decision-making is decentralized. Read permission of the blockchain is public.

Consortium blockchain: Permission is required to join (permissioned), consensus is reached by a selected group of peers and hence, the decision-making is partially decentralized. Read permission of the blockchain can be public or restricted.

Private blockchain: Permission is required to join (permissioned), consensus is reached by a single organization and hence, the decision-making is centralized. Read permission of the blockchain can be public or restricted.

Multi-chain Blockchain

In the original blockchain design [71], all members of the network read from, and write to, the same shared ledger. Attempting to mitigate the privacy and scalablity concerns [45] this entails, *multi-chain blockchain* technology emerged [1, 22, 41]. Intuitively, a multi-chain blockchain consists of multiple blockchain ledgers, which are accessible by a select group of members of the network. In this way, only selective members can read certain transactions, and blocks can be created concurrently in the multiple ledgers. A more precise definition of multi-chain blockchain is given in Definition 2.2.2. It must be noted that this definition only specifies the parts of a multi-chain blockchain relevant for this thesis, as the signature scheme proposed in Chapter 4 is targeted at all blockchains following this design principle. Hence, the implementation details of a multi-chain blockchain, such as the access control mechanism, are open for interpretation. However, the design of the signature scheme assumes blockchains with absolute finality [24], also called forward security [29]. This means that once a transaction or block is included in the ledger, it will not be revoked in the future. This typically applies for private and consortium blockchain, in which the consensus algorithm is used to reach fault-tolerance, and the authenticity of blocks is guaranteed by a digital signature of the block writer [92]. Hence, signing of blocks is one of the main use cases of MBPQS, as discussed in Chapter 6.

Definition 2.2.2 *Multi-chain Blockchain. A multi-chain blockchain consists of a set of members $\mathcal{M}$ and a set of ledgers $\mathcal{L}$ with $|\mathcal{L}| \geq 2$ s.t. each $\ell \in \mathcal{L}$ is associated with one set of members $\mathcal{R} \subseteq \mathcal{M}$ who have read-access to ℓ, and one set of members $\mathcal{A} \subseteq \mathcal{M}$ who have write-access (append) to ℓ with $|\mathcal{R}| \geq 2$, and $|\mathcal{A}| \geq 1$.*

2.3 Building Blocks

The scheme proposed in Chapter 4 uses WOTS-T as OTS scheme, and the authentication structure construction techniques from XMSS-T. Therefore, in this section, these building blocks are described in detail. Here, only the technical specification of XMSS-T according to RFC 8391 is discussed [48]. An overview of HBS schemes leading up to XMSS-T and alternative HBS schemes are covered in Chapter 3. As a note to the reader, RFC 8391 states

that it specifies XMSS, but does in fact describe the multi-target resistant variant XMSS-T as stated in Chapter 6 of the RFC.

In both WOTS-T and the XMSS-T authentication structure, so-called hash function addresses are used protect the schemes against multi-target attacks as described in Section 2.1.2 and later in Section 3.1.7. Because hash addresses are used both in WOTS-T and in the XMSS-T authentication structure, they are discussed before describing the schemes.

2.3.1 Hash Function Addresses

A hash (function) address is a unique identifier used to randomize the hash function instances in XMSS according to the RFC [48]. The hash address is generated in a deterministic way, such that each address is used in exactly one hash function call. The hash address differentiates the seed to the PRF used to generate the bitmasks and hash function keys, guaranteeing uniqueness of hash function calls for a key pair. Subsequently, the public key is also included in the seed to guarantee uniqueness among different key pairs.

Table 2.1 shows the three types of hash addresses used in WOTS-T and XMSS-T. Each hash address type has a total length of 32 bytes. From left to right, the first type, the OTS hash address, is used in the WOTS-T scheme. The second type, the L-tree address, is used in the XMSS-T authentication structure to compress WOTS-T verification keys in tree leaves. This is done using so-called L-trees, which are explained later. The third type, the hash tree address, is used to compute the internal nodes in the authentication structure.

The address fields are made to fit in 32-bit words, except the tree address, which requires two words. All of the address types start with a layer address followed by a tree address which are used in the multi-tree variant of XMSS-T, called XMSS-T^{MT}. These address words describe the position of an XMSS tree in a multi-tree structure. The layer address encodes the layer level of a tree in a multi-tree, starting from level 0 at the bottom layer. The tree address encodes the index of the tree in the layer, starting at 0 from left to right. In the single-tree variant, the layer address and tree address are set to zero. Subsequently, all address types have a 32-bit number denoting the address type. The value of this words is 0 for an OTS address, 1 for an L-tree address, and 2 for a hash tree address. The rest of the hash address words are discussed separately for each address type because their functionality differs for each type.

OTS Hash Address

The next word in the OTS hash address is the OTS address which encodes the index of the leaf in the tree corresponding to this OTS. The chain address and hash address words are used to differentiate between hash functions in the WOTS-T chaining function discussed later. Finally, the $keyAndMask$ word is used to differentiate the hash addresses between key and bitmask generation. For key generation, this word is set to 0, and to create the bitmask, this word is set to 1.

Table 2.1: Hash function addresses used in XMSS-T to randomize hash function invocations in a deterministic way.

OTS Hash Address		**L-tree Address**		**Hash Tree Address**	
layer address	(32-bit)	layer address	(32-bit)	layer address	(32-bit)
tree address	(64-bit)	tree address	(64-bit)	tree address	(64-bit)
type = 0	(32-bit)	type = 1	(32-bit)	type = 2	(32-bit)
OTS address	(32-bit)	L-tree address	(32-bit)	padding = 0	(32-bit)
chain address	(32-bit)	tree height	(32-bit)	tree height	(32-bit)
hash address	(32-bit)	tree index	(32-bit)	tree index	(32-bit)
keyAndMask	(32-bit)	keyAndMask	(32-bit)	keyAndMask	(32-bit)

L-tree Address

As mentioned before, L-trees are used to compress WOTS-T verification keys into leaves of the authentication structure. In the L-tree address, the word after the type word, called L-tree address, describes the index of the leaf in the authentication tree that is computed. The next two words describe the position for each L-tree node used as input to compute the next node in the L-tree. The tree height encodes the node height, and the index describes the node index from left to right. Again, the last word of the hash address is used to differentiate between hash addresses used to generate keys and ones that are used to generate the bitmasks. Because the input of this hash function consists of two n-byte nodes, a $2n$-byte bitmask is required to mask the total input. Therefore, in an L-tree address, there are three possible values for the last word. To generate the key, the word is set to 0. To generate the most significant n bytes of the bitmask, the word is set to 1, and for the least significant n bytes, it is set to 2.

Hash Tree Address

The type word in the hash tree address is followed by a 32-bit zero padding because these bits are not required for anything else in a hash tree address. The last tree words in the hash tree address work exactly similar to those in the L-tree address. Only the tree height and tree index denote the position of the computed node in the Merkle tree instead of the L-tree.

2.3.2 Winternitz One-time Signatures

As mentioned before, WOTS-T according to RFC 8391 is the OTS used in the HBS scheme proposed in this thesis. This section presents a detailed description of the scheme.

Parameters

WOTS-T has two parameters, the Winternitz parameter $w > 1 \in \mathbb{N}$, and the security parameter $n \in \mathbb{N}$. The Winternitz parameter is a space-time trade-off defining the amount of bits which are signed simultaneously ($\log_2(w)$). A higher value of w results in smaller signatures, but increased computation times. The security parameter defines the length of the message, OTS key elements, and signature elements in bytes. In practice, n is determined by the hash function used, and corresponds to its output length.

From these parameters, ℓ is computed, which defines the number of n-byte string elements in the signing key, verification key, and signature. ℓ is computed as $\ell = \ell_1 + \ell_2$, where

$$\ell_1 = \left\lceil \frac{8n}{\log_2(w)} \right\rceil, \quad \ell_2 = \left\lfloor \frac{\log_2(l_1(w-1))}{\log_2(w)} \right\rfloor + 1.$$

Chaining Function

WOTS-T uses a chaining function to compute the signatures and verification keys. In the chaining function, $\mathcal{C}^{i,j}(x, a, \text{SEED})$, $i \in \mathbb{N}$ is the iteration counter, $j \in \mathbb{N}$ the start index, $x \xleftarrow{\$} \{0,1\}^{8n}$ the input value, a an OTS address, and SEED an n-byte public seed. The chaining function is defined as

$$\mathcal{C}^{i,j}(x, a, \text{SEED}) = \begin{cases} x & \text{if } i = 0, \\ F(k_{i,j}, \mathcal{C}^{i-1,j}(x, a_h, \text{SEED}) \oplus r_{i,j}) & \text{if } i > 0, \end{cases}$$

where a_h denotes an OTS address with $Hash\,address = i + j - 1$, $k_{i,j}$ a pseudo-randomly generated key, and $r_{i,j}$ a pseudo-randomly generated bitmask. Their respective values are computed as $k_{i,j} = PRF(\text{SEED}, a_h)$ with $KeyAndMask = 0$, and $r_{i,k} = PRF(\text{SEED}, a_h)$ with $KeyAndMask = 1$ in a_h

Key Generation

The algorithm $KeyGen(s, a, \text{SEED})$ takes an n-byte secret seed s, an OTS address a, and an n-byte public seed SEED, and generates a key pair (sk, pk). The signing key $sk = (sk_1, ..., sk_\ell)$ is a pseudorandomly generated ℓ-length array of n-byte strings from s. In this way, instead of storing sk, only the seed s needs to be stored to be able to deterministically compute sk. The verification key $pk = (pk_1, ..., pk_\ell)$ is also an ℓ-length array of n-byte strings, where $pk_i = c^{w-1,0}(sk_i, a_{c_i}, \text{SEED})$ for $1 \leq i \leq \ell$ and where a_{c_i} is an OTS address with $Chain\,address = i$. Hence, the chain address differentiates the hash address for each element in the key.

Signing

The algorithm $S(m, sk, a, \text{SEED})$ takes an n-byte message m, a signing key sk, an OTS address a, and a public seed SEED and outputs a signature σ on m. To generate this signature, first, m is mapped to an ℓ_1-length array $(m_1, ..., m_{\ell_1})$ of $\frac{|m|}{\ell_1}$-bit elements. Second, a

checksum c is computed as $c = \sum_{i=1}^{\ell_1}(w-1-m_i)$, which is mapped to an ℓ_2-length array $(c_1, ..., c_{\ell_2})$. Consecutively, the arrays m and c are concatenated, resulting in an ℓ-length array $(b_1, ..., b_\ell) = m||c$. Then, each bit-string $b_i, 1 \leq i \leq \ell$, is interpreted as a natural number with $0 \leq b_i \leq w-1$. Now, the signature $\sigma = (\sigma_1, ..., \sigma_\ell)$ is computed as $\sigma_i = \mathcal{C}^{b_i,0}(sk_i, a_{c_i}, \text{SEED})$ for each $1 \leq i \leq \ell$, where a_{c_i} is again an OTS hash address with $Chain\,address = i$.

Verification

As shown, the signature generation is almost identical to the generation of the verification key. Instead of $w-1$ recursive calls of the chaining function on the signing key element, b_i hash chains are invoked. This property is utilized by the verification algorithm $V(\sigma, m, a, \text{SEED}) = pk'$ which takes a signature σ, message m, OTS address a, and public seed SEED as arguments and outputs a verification key pk'. It does so by 'finishing' the hash chain from the signature step to the verification step for each element in the signature. Therefore, first the message and its checksum are mapped to an ℓ-length array $(b_1, ..., b_\ell)$ in an identical way as during the signing process. Subsequently, the pk' is computed from σ as $pk_i = \mathcal{C}^{w-1-b_i,b_i}(\sigma_i, a_{c_i}, \text{SEED})$ where, again, a is an OTS address with $Chain\,address = i$. If the signature is valid, pk' equals the verification key pk. Since the pk is verified against the authentication structure instead, this last step is omitted here. For a standalone version, the verification algorithm V would have a verification key pk as extra argument to validate that pk' equals pk.

2.3.3 Authentication Structure

The authentication structure of the digital signature scheme proposed in Chapter 4 is based on XMSS-T [50]. The XMSS-T authentication structure is a perfect binary tree, exactly like the Merkle tree in Figure 2.1. However, there are two major differences in its construction, namely the construction of the leaf nodes, and the generation of internal nodes. The reason for the changes in the construction of the XMSS-T tree, compared to Merkle's tree construction, is to achieve stronger security properties. More specifically, weaker security properties are required from the used hash function in order to guarantee the security of the scheme. Instead of a collision resistant hash function, a second-preimage resistant hash function is sufficient to guarantee the security of the XMSS-T authentication structure. Since second-preimage resistance is unaffected by the birthday paradox as opposed to collision resistance, this results in smaller signatures for the same security levels for XMSS-T, compared to MSS. In this section, the construction of the XMSS-T authentication structure is discussed, which we call the XMSS-T tree from now on.

Parameters

Since WOTS-T is a building block of XMSS-T, the two share the Winternitz parameter w and security parameter n. Furthermore, XMSS-T has a parameter H, which defines the tree height of the authentication structure described in the next section. H defines the amount of WOTS-T keys associated with the XMSS-T public key, being 2^H.

L-trees

In Merkle's Signature Scheme (MSS), discussed in Section 2.1.7, the security of the scheme relies on the fact that it is hard to find a hash collision for one of the verification keys. Namely, if a collision is found for any of the tree leaves, a forgery key is found. Therefore, MSS relies on the collision resistance of the hash function used. In XMSS-T, the verification key is not compressed into a leaf by directly hashing it directly, but by using an L-tree. In this way, bitmasks can easily be appended to each hash function call to help prove the scheme relies on second-preimage resistance. An L-tree is an unbalanced binary tree of height $H_\ell = \lceil \log_2(\ell) \rceil$, where ℓ is the number of elements in a WOTS-T verification key Y_i. These ℓ n-byte elements form the leaves of the L-tree. The n-byte internal nodes and the root node of the L-tree are denoted as $N_{i,j}$, where $1 \leq j \leq h_\ell$ represents the vertical position of the node, and $0 \leq i < 2^{h_\ell - j}$ represents the horizontal position of a node on the j-th layer. The leaf nodes are defined as $N_{i,0} = L_i$. The construction of internal nodes and the root node in an XMSS-T L-tree and Merkle tree is shown in Figure 2.3. More formally, an XMSS-T tree node $N_{i,j}$ is computed as:

$$N_{i,j} = H_{k_{i,j}}((N_{2i,j-1} || N_{2i+1,j-1}) \oplus (r_{i,j})),$$

where $k_{i,j}$ is a pseudo-randomly generated key for each hash invocation, and $r_{i,j}$ a pseudo-randomly generated bitmask. These values are computed using a PRF, $\mathcal{F}_n : \{0,1\}^{2(8n)} \rightarrow \{0,1\}^{8n}$ as follows. $k_{i,j} = \mathcal{F}_n(\text{SEED}, a_{L_0})$, where a_{L_0} is a L-tree address with $Tree\,height = j$, $Tree\,index = i$, and $KeyAndMask = 0$. $r_{i,j} = (\mathcal{F}_n(\text{SEED}, a_{L_1}) || \mathcal{F}_n(\text{SEED}, a_{L_2}))$, where a_{L_1} and a_{L_2} are L-tree addresses with the same field values as a_{L_0}, except with $KeyAndMask = 1$, and $KeyAndMask = 2$, respectively. Since ℓ is not necessarily a power of 2, the resulting tree might be unbalanced. When a single non-paired node is left on a tree level, this node is elevated to a higher level until it becomes the right sibling of another node.

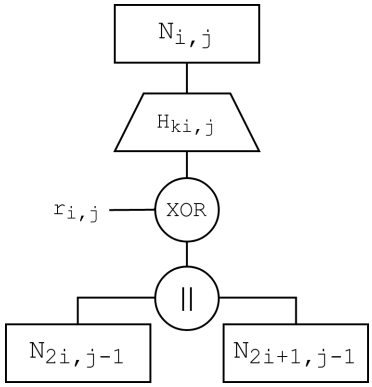

Figure 2.3: Construction of an XMSS-T tree node $N_{i,j}$ from its two children nodes $N_{2i,j-1}$, and $N_{2i+1,j-1}$.

Merkle Tree

As mentioned, the 2^H compressed verification keys form the leaves of the XMSS-T authentication structure. Subsequently, the authentication structure is created as a perfect binary hash tree of height H. The internal nodes and root node of this hash tree are computed in an almost identical manner as the internal nodes of the L-tree. Instead of L-tree addresses, hash tree addresses are used in the construction of the internal nodes, and the root node. The root node of the resulting hash tree $N_{0,H}$ together with the public seed SEED form the XMSS-T public key. The public seed is added to the public key such that a verifier can compute the keys and bitmasks used in the WOTS-T chain function, and to authenticate the verification key.

Chapter 3

Related Work

In this chapter, existing work related to this thesis is presented to provide the reader with both a historical and contextual overview of the topic. To this end, first, an overview of the work leading to the state-of-the-art hash-based signature (HBS) schemes is given and explained. Subsequently, HBS schemes specifically designed for blockchain technology are discussed. Throughout this chapter, the primitives covered in the background chapter are assumed to be known to the reader.

3.1 Universal HBS Schemes

This section describes HBS schemes that are universally applicable, i.e. not specifically designed for blockchain. Although they are not specifically designed for it, they can in fact be used in blockchain to reach post-quantum security, as shown in [88], for example.

Before explaining each HBS scheme and its improvements compared to earlier versions, consider Figure 3.1 where the relationships between the described schemes is illustrated. The meaning of the abbreviations in the diagram can be found in the list of abbreviations. The white nodes represent HBS schemes that lead to the schemes in the black nodes which denote the current (proposed) standards. A solid arrow denotes that a scheme is inspired by, or derived from, another scheme. A dashed arrow means that the scheme pointed towards is the multi-tree variant of the originating scheme. The vertical position of the nodes in the diagram corresponds to the relative chronology of the scheme proposals. HSS is positioned at the same level as LMS because the patent of LMS [64] describes a hierarchical use of LMS, which is considered as the first proposal of HSS. The first full specification of HSS was in [67].

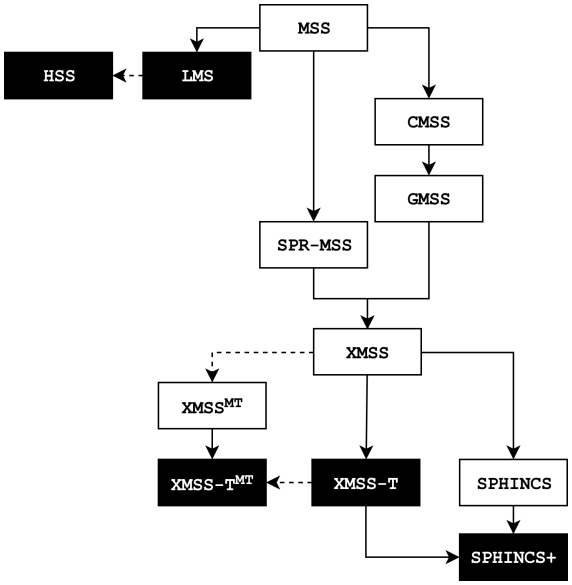

Figure 3.1: Relationship between universal HBS schemes.

3.1.1 MSS

Already in 1979, Ralph Merkle proposed the first many-time HBS scheme called Merkle's Signature Scheme (MSS) [69]. Unlike the earlier one-time signature (OTS) scheme by Lamport [63], MSS can be used to create multiple signatures with the same public key. To this end, multiple OTS keys are associated with one public key using a binary hash tree, also known as a Merkle tree as described in Section 2.1.7.

MSS results in signing keys no larger than those in the used OTS, and using a pseudo-random key generator, they can be reduced to n bits, where n is the security level of the

OTS. The length of the MSS public key is equal to the output length of the hash function used to construct the Merkle tree. The size of a MSS signature equals the OTS size, plus the size of the authentication path – which is $h \times n$ bits, where h is the height of the tree, plus the size of the OTS verification key. In case a Winternitz OTS (WOTS) is used, the verification key can be computed from the OTS signature, and needs not to be included in the signature.

The key generation time scales exponentially in h because the entire tree has to be built at once to generate the keys. Furthermore, it has been shown that the security of the scheme relies on the strong collision resistance property of the underlying hash function [38]. This is a drawback, as this security property is hard to achieve for hash functions, and has already been broken for several historical standardized hash functions, such as MD5 and SHA1 [83, 89, 87].

3.1.2 CMSS

In 2006, Buchman, García, Dahmen, Döring, and Klintsevich proposed Chained Merkle Signature Scheme (CMSS), an improved version of MSS with faster signature and key generation algorithms, a reduced signing key size, but a larger signature size [17]. In CMSS, the key authentication structure consists of a main tree and multiple subtrees instead of a single tree. Both the main tree and subtrees are Merkle trees of height h. The root node of the main tree represents the CMSS public key, and the leaf nodes of the main tree are used to sign root nodes of subtrees. The leaf nodes of a subtree are used to sign messages, resulting in 2^{2h} available signing keys instead of 2^h. The subtrees are dynamically generated when new signing keys are required, reducing the initial key generation time. Moreover, in CMSS, the OTS signing keys are generated using a seeded deterministic pseudo-random number generator (PRNG). Instead of storing the entire signing key, only the seed input for the PRNG is stored, reducing the size of the signing keys. To reduce the signature generation time, an efficient tree traversal algorithm is used to compute the authentication path as described in [84]. Finally, it must be noted that the signature size in CMSS is larger compared to MSS because, besides the signature and authentication path for the signed message, also the signature and corresponding authentication path on the subtree root node are included in the message signature.

3.1.3 GMSS

In 2007, Buchmann, Dahmen, Klitsevich, Okeya, and Vuillaume presented the Generalized Merkle Signature Scheme (GMSS) [15], an improved version of CMSS. Compared to CMSS, it is more flexible and adaptable to the requirements of the users. Furthermore, it greatly reduces the signing time by distributing the computations for one signature over several previous signatures and the key generation, i.e. part of a signature is precomputed. This makes parameter sets with higher computational efforts, resulting in smaller signatures, feasible. Furthermore, GMSS increases the maximum number of signatures in CMSS (2^{40}) to a virtually unlimited number (2^{80}) [15].

The GMSS authentication structure consists of a tree with a number of layers which in turn are Merkle trees. The trees on different layers may have different heights. The afore-

mentioned increased flexibility of the scheme stems from the added parameters defining these heights. Besides the authentication structure and the scheduling strategy for precomputation in the signature generation, the internal logic of GMSS is similar to CMSS.

3.1.4 SPR-MSS

In 2008, Dahmen, Okeya, Takagi, and Vuillaume proposed the Second-Preimage Resistant Merkle Signature Scheme (SPR-MSS), an HBS scheme with, compared to the earlier described schemes, a lower security requirement for the hash function used to construct the authentication structure. Instead of a collision-resistant hash function, a second-preimage resistant one suffices for SPR-MSS [26].

To this end, a key is added to the hash function, and the tree nodes of the authentication structure are XORed with randomly chosen bitmasks before their parent node is computed. These bitmasks and this key for the hash function are generated during key generation and appended to the signing keys. Consequently, SPR-MSS has larger signing keys, but a higher security level compared to earlier schemes, as it is not prone to the birthday paradox. This in turn results in shorter signature sizes compared to earlier HBS schemes, as shorter hash outputs are required in the authentication structure to obtain the same security levels.

3.1.5 XMSS

In 2011, Buchmann, Dahmen and Hülsing presented the eXtended Merkle Signature Scheme (XMSS), which they call "a practical forward secure signature scheme based on minimal security assumptions" [14]. It was the first stateful HBS scheme described in a RFC by the Internet Research Task Force (IRTF) [48]. The version of XMSS described in the RFC uses the key generation variant from [14] which does not result in forward-security. Furthermore, its successor, XMSS-T [50], which is discussed later in this chapter, is one of the two HBS schemes considered for standardization by the National Institute for Standards and Technology (NIST) [85].

XMSS uses an authentication tree construction technique similar to SPR-MSS in order to achieve the lowered second-preimage resistant security requirement for the hash function. For the OTS, the earliest described versions of XMSS use a WOTS variant with a pseudorandom key generator similar to WOTS^{PRF} [13]. However, the security proof for WOTS^{PRF} turned out to be flawed, and the practical security level lower than initially estimated [62]. Therefore, in later descriptions of XMSS, such as (drafts of) RFC 8931, WOTS+ [47] is used instead [48].

The authors of XMSS prove that the only security requirements for the scheme are a second-preimage resistant keyed hash function and a PRF. This implies that the scheme is based on minimal security requirements, as the existence of a secure signature scheme implies the existence of a second-preimage resistant hash function and a PRF family [81, 80, 43, 40]. Furthermore, the authors prove that XMSS is forward secure.

Finally, the paper states that XMSS signatures are four times smaller compared to those of SPR-MSS at a slightly higher security level. However, this advantage is partly due to the

chosen OTS, which is a variant of Lamport's OTS (LD-OTS) for SPR-MSS and a WOTS variant for XMSS.

3.1.6 XMSS^{MT}

In 2013, Hülsing, Rausch, and Buchmann proposed XMSS^{MT}, a multi-tree variant of XMSS that "can be used to sign a virtually unlimited number of messages" [49]. In XMSS^{MT}, multiple XMSS trees are stacked on top of each other in the same way as in CMSS [17]. The OTS key leaves of the XMSS trees at the bottom layer are used to sign messages, and the leaves of higher layered XMSS trees are used to sign the root nodes of underlying XMSS trees. The public key contains the root node of the tree on the highest level. In the signatures, all the authentication paths and signatures from the used OTS key up to the root are included. Consequently, the signature sizes and verification times in XMSS^{MT} are 2 to 10 times longer than in XMSS, depending on the number of layered trees. However, the key generation times for similar numbers of keys, are significantly lower compared to XMSS, because in XMSS^{MT}, not the entire structure has to be computed at once, but trees can be gradually added and computed [48].

3.1.7 XMSS-T

In 2015, Hülsing, Rijneveld, and Song discovered that the bit-security of earlier HBSs reduces linearly to 2^h, where h is the total tree height parameter, due to multi-target attacks on hash functions, as explained in Section 2.1.2 [50].

Usually, multi-targets attacks have an insignificant impact on the theoretical security of the hash function, as the amount of targets d is polynomial in the output length n of the hash function. However, when choosing the tree height for XMSS, it can have serious consequences. Especially in the multi-tree variant XMSS^{MT}, where the tree height can be chosen up to 60, resulting in $d = 2^{66}$ possible targets for an attacker to attack at once. Instead of 256 bits of classical security using a hash function with 256 bit output length, the complexity of an attack is reduced to $\mathcal{O}(\frac{2^{256}}{2^{66}})$. Therefore, to achieve a classical security level of 256 bits, instead of a hash function with 256 bit output length, a hash function with 322 bit output length would be required, resulting in 25% larger signatures [50].

To prevent multi-target attacks, the authors propose changes to WOTS+ and the XMSS tree generation, resulting in WOTS-T and XMSS-T respectively. For every hash function call in WOTS+ and XMSS, different keys and bitmasks are used. This makes every hash function call unique, such that an attacker has to choose a particular hash function to attack, corresponding to the assumptions made in the original security level assessments of WOTS+, and XMSS in [14]. To avoid larger signing keys for the scheme, the bitmasks and keys are deterministically computed using a PRF on a seed value.

3.1.8 LMS

In 1993, Leighton and Micali filed a patent on Leighton and Micali Signatures (LMS), an improved MSS scheme [64]. The patent claims both LMS and its corresponding OTS scheme

called LM-OTS. LMS improves on MSS by adding distinct security strings in each hash invocation. In this way, multi-target attacks, as described in Section 2.1.2, are mitigated. Furthermore, the LMS patent describes a multi-layered hierarchical implementation of the LMS scheme, discussed in the next section. The recent expiration date[1] of the patent, and the emerging quantum-threat for conventional public-key cryptography, has revitalized both the public and academic interest in the scheme. Accordingly, LMS is considered for standardization by the NIST [85], and recently, its specification is published in an RFC [68].

With LMS and XMSS-T being the only two HBS schemes considered for standardization, it seems logical to compare the two. As OTS scheme, LMS uses LM-OTS [64], which is identical to WOTS [69], except for the addition of the aforementioned security strings. XMSS-T uses WOTS-T and an internal tree construction in which every hash input is XORed with pseudorandomly generated bitmasks to enable a security proof in the standard model (SM) [51]. LMS has two slightly different security proofs in the Random Oracle Model (ROM). First by modeling the hash function as a random oracle (RO) [57], and a second prove where the Merkle-Damgård construction, used in the first two Secure Hash Algorithms (SHA) families, is modeled as RO [36]. Furthermore, both LMS and XMSS-T have security proofs in the Quantum ROM (QROM) [33, 50, 14]. The current XMSS-T standard, as found in the RFC, has been proven forward secure and existentially unforgeable under adaptive chosen message attacks (EU-CMA) in the SM, post-quantum EU-CMA in the QROM, and forward secure [50, 48]. However, when the keys and bitmasks in the hash functions are pseudorandomly generated using a PRF, a RO is introduced for the PRF. Conclusively, compared to XMSS, LMS has 3 to 5 times faster algorithms, but relies on stronger security assumptions resulting in weaker security proofs [55, 28]. Intuitively, for each WOTS-T chain in XMSS-T, multiple operations are required to generate randomization elements. However, in LMS, no randomized bitmasks are used, and the hash function is assumed to behave randomly. More specifically, the Merkle-Damgård construction is assumed to behave like a random oracle in the second security proof of LMS. Therefore, for each LM-OTS chain, only a single hash operation is required, instead of the multiple operations in WOTS-T as used by XMSS-T.

3.1.9 HSS

Hierarchical Signature Scheme (HSS) [67] is the multi-level variant of LMS, similar to what XMSS^{MT} is for XMSS. It consists of layered LMS trees, and works similarly to CMSS or XMSS^{MT}, except that it uses the LM-OTS scheme. Another subtle difference is that in HSS, the LMS trees on different layers can have different heights, which is not allowed in XMSS^{MT}. Similarly to the comparison of LMS versus XMSS, HSS is about 3 to 5 times faster than XMSS^{MT} for similar signature sizes, but the latter relies on weaker security assumptions and results in approximately 5% smaller signatures [55].

3.1.10 SPHINCS(+)

In 2015, Bernstein et al. published SPHINCS, the first practical stateless HBS scheme [9]. While SPHINCS is not directly related to our work, it is mentioned to provide the reader

[1] The LMS patent expired on 26 July 2019.

with a contextual overview of the current state-of-the-art HBS schemes. In contrast to XMSS, SPHINCS can be used as a drop-in replacement for digital signature schemes currently used in blockchains, as it is stateless. However, the signature size of SPHINCS-256 (∼41KB) is 5 to 20 times larger than XMSS at approximately the same security levels, depending on the number of keys in XMSS. Since XMSS signatures are already relatively large compared to current standards in blockchain, SPHINCS might not be ready for use in blockchains where one or multiple signatures are stored on the ledger for each transaction [92]. Improvements to SPHINCS, proposed in SPHINCS+ [4], reduce the signature sizes from the earlier stated 41KB to approximately 30KB for similar parameters. Furthermore, SPHINCS+ implements the multi-target mitigation technique, proposed in XMSS-T [50].

Besides the large signatures, another drawback of SPHINCS(+) is its signing speed. Therefore, the authors state in the latest version of SPHINCS+: "most resource-constrained devices can deal with a state and XMSS is far better suited for these devices". Considering that blockchains can deal with state as well, the signature scheme proposed in Chapter 4 is stateful.

3.2 HBS Schemes for Blockchain

In this section, HBS schemes specifically designed for blockchain are discussed. The first two sections describe practical HBS schemes, and the last section describes a higher-level protocol.

3.2.1 XNYSS

Extended Naor-Yung Signature Scheme (XNYSS) is an HBS scheme proposed in 2018 by van der Linde [86]. The scheme is designed for blockchain applications, such as Bitcoin, where addresses (i.e. public keys) are recommended to be used only once to enhance the privacy of the user [2]. The work in [86] claims that the use of an OTS scheme for Bitcoin is not plausible, as transactions are not guaranteed to be adopted in the blockchain due to network failures, transaction rejections, or forks of the blockchain. Hence, in XNYSS, multiple OTS key pairs are associated with one Bitcoin address to have back-up keys in case previous key(s) end up missing. Namely, these keys cannot be reused, as it concerns OTS keys. Thus, even though XNYSS is a few-time signature scheme, it is recommended to use back-up keys only in cases of emergency, to avoid losing one's funds when no back-up keys are left to sign transactions.

Besides the few-time signatures described above, XNYSS also supports many-time signatures using Naor-Yung signature chains [72]. These many-time signatures, called long-term signatures, are constructed as follows: Whenever a peer signs a transaction, it also signs the hash of the verification key corresponding to the next signature, creating a chain of related keys. The key at the root of this chain is the long-term public key. To verify a signature, the verifier computes the verification key from the signature, and checks whether the key belongs to the long-term key chain. Because Bitcoin miners can decide whether or not to process a transaction based on the fee, it can take some time for a transaction to be processed.

Using a single Naor-Yung chain, the signer has to wait until the transaction is accepted in the blockchain before it can sign a new transaction, heavily limiting the rate in which peers can propose transactions. Furthermore, when a transaction does not end up in the blockchain, the signer is unable to sign transactions ever again, as there are no signing keys left associated with its long-term public key. To solve these issues, multiple signing keys are signed in every transaction as back-up keys. Naturally, this works only if transaction failure is limited; if all the back-up keys are used, the peer can no longer move its funds. Nonetheless, after several adopted transactions, this risk diminishes, as the signer has a sufficient number of associated keys to sign new transactions. To decrease this number of required adopted transactions, one could sign more keys with every transaction. However, this results in a significant number of unused keys, and large signing times, as more keys need to be generated during signing. Furthermore, when a transaction is not adopted in the blockchain, all its associated verification keys cannot be reused, as the key chain has a missing link. Inserting such a "missing link" in later signatures is not implemented, as this is considered too expensive, majorly increasing the signature size [86]. To prevent key re-usage in XNYSS, each client needs to maintain a database of unused public keys which where announced to the network in previous transactions. XNYSS has a standalone implementation[2], and an implementation in Bitcoin.[3]

3.2.2 BPQS

In 2018, a group of researchers led by Chalkias from the company R3 published a paper on Blockchained Post-Quantum Signatures (BPQS) [23]. BPQS is an (extensible) post-quantum signature scheme for blockchain and distributed ledger technologies (DLT), inspired by the structure of blockchains, and based on XMSS-T. The design of BPQS is focused on fast first-time signatures as the signature size grows linearly for each consecutive signature. However, the signature size can be reduced by storing the authentication path on the blockchain.

BPQS has two building blocks: a few-time (BPQS-FEW) and an extensible many-time scheme (BPQS-EXT), both illustrated in Figure 3.2. BPQS-FEW is a Merkle tree with only two nodes on every layer. The right leaf is a compressed OTS key, and the left leaf is the root of the underlying two nodes. Root nodes are constructed using the internal node construction technique from XMSS-T. The authentication structure of BPQS-EXT is a tree of layered 2-leaf Merkle trees. The left leaf on each layer is used as fall-back key to sign the root of the tree on the next layer, and the right leaf is used as OTS key.

In the full BPQS scheme, BPQS-FEW can be combined with BPQS-EXT, and optionally any other HBS scheme. Hence, the authentication structure of the digital signature scheme proposed in the next chapter can be considered an extended variation of the full BPQS scheme. For BPQS, a standalone implementation exists.[4]

[2] https://github.com/lentus/xnyss
[3] https://github.com/lentus/wotscoin
[4] https://github.com/corda/bpqs

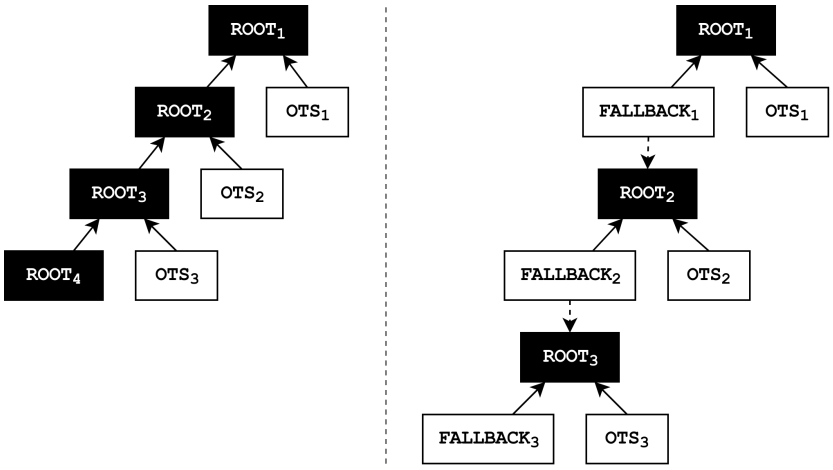

Figure 3.2: Left: BPQS-FEW, the fixed-length few-time signature BPQS variant. Right: BPQS-EXT, the extensible many-time signature variant.

3.2.3 BLT

Server-assisted

In 2017, Buldas, Laanoja, and Truu (BLT) published a paper describing an HBS scheme built from a hash-then-publish digital time-stamping scheme [20]. Instead of using one-time keys, the keys are time-bound, thus making key management unnecessary. The general idea of their scheme is to have signing keys that are valid in a certain time slot, and after this time slot expires, the signing key will become a verification key. Since the signing key to be used is determined by the signing time, the risk of accidental key re-usage is mitigated.

As each signature in the scheme is time-stamped, the authors argue that the scheme comes with "free" cryptographic time-stamping. This could be beneficial when utilizing the scheme in a blockchain as the transactions in a blockchain are typically time-stamped. Due to the time-stamping feature, the scheme must be server-assisted. Besides this, the scheme requires clock synchronization and network issues could result in several problems. For example, when the current time does not match the time-stamp due to latency or clock drift, the signature cannot be composed.

Blockchain-assisted

In 2018, the same authors published a paper describing a blockchain-assisted HBS scheme [21]. In fact, the scheme requires an authenticated data structure to check the server; blockchain is just one possible tool. The blockchain-assisted HBS scheme shares logic with the aforementioned server-assisted scheme. However, while the previous version of the BLT scheme uses time-bound keys, the blockchain-assisted BLT scheme uses OTS sequentially to ensure keys are not reused. This technique mitigates the large overhead of unnecessary key generations for time periods when no signatures are generated.

To eliminate the need of a trusted server, a well-defined append-only repository R is required. The server publishes summaries of the key-usage activities in the network to this repository in fixed-length rounds. The summaries must include a proof of the server's correct operation, validated by the repository before being accepted. Now, the only requirement for the scheme to be secure is that R operates correctly. The requirements for R are that it is an append-only repository, able to publicly store proofs, and verifiable by the peers. Furthermore, according to the paper, the repository can be implemented as a byzantine fault tolerant distributed state machine. These requirements can be met by a blockchain, such that it can be used to replace the server from the previous scheme.

In the paper, the authors state that "the model of server-supported signing is a higher-level protocol and is not directly comparable to traditional signature algorithms" [21]. Nonetheless, in 2019, an extended paper was published comparing BLT and the state-of-the-art HBS schemes XMSS and SPHINCS regarding performance [19]. However, this comparison is limited to 3650 signatures, and because the BLT scheme requires a time-stamping service, the comparison does not give a clear representation of the practical performance of BLT compared to XMSS or SPHINCS.

Chapter 4

Multi-Blockchain Post-Quantum Signatures

In this chapter, the digital signature scheme 'Multi-Blockchain Post-Quantum Signatures' (MBPQS) is proposed. The scheme is a practical BPQS variant specifically designed for blockchain systems with multiple, segregated blockchains, hence its name. The proposed scheme is constructed using the primitives and building blocks described in Chapter 2, and techniques from existing schemes mentioned in Chapter 3. In this chapter, the design, algorithms, and security of MBPQS are discussed. Occasionally, the reader is referred to the background chapter to avoid redundant information.

4.1 Design Choices

For the scheme proposed in this chapter, the approach of BPQS is adopted instead of those in XYNSS or BLT. In what follows, the reasoning behind this decision is provided.

In both XYNSS and BPQS, one-time verification keys are linked to associate them with one public key, as explained in their respective sections in the previous chapter. In XYNSS, peers create and broadcast backup keys to the network, as "inserting a missing link" in the Naor-Yung chain is considered too expensive [86], as mentioned in Section 3.2.1. Indeed, to insert a missing link in the long-term key chain, the missing signature needs to be added. Ergo, considering $w = 16$ and SHA2-256 ($n = 32, \ell = 67$), inserting a missing link costs at least $w \times \ell = 2144$ bytes in XYNSS.

In BPQS-FEW, keys are linked to one another using an unbalanced hash tree, consisting of two nodes per layer. These two nodes can be retrieved from a signature. The right-side node is the compressed verification key, and the left-side node must be added to the signature for verification. Inserting a missing link in such a tree implies adding the missing right-side node to the subsequent signature. Namely, the left-side node can be computed from the nodes on the next layer (which are included in the next signature). Therefore, adding a missing link in BPQS-FEW costs only $n = 32$ bytes.

An other advantage of BPQS over XYNSS is its reduced state maintenance. To prevent key re-usage of the broadcast public keys in XYNSS, each peer maintains a database specifying the unused public keys. To prevent key re-usage in BPQS, each peer only needs to keep track of a single integer specifying the last layer of the tree used for signing [23].

Furthermore, BPQS is chosen over BLT, mainly because the design of BLT diverges significantly from that of HBSs which are demonstrated to be usable in blockchain [21, 88, 86]. Furthermore, as of yet, there is no practical demonstration of BLT in a blockchain, and significant changes are required in existing blockchain frameworks to support the server-assisted signing features. The scheme also relies on a reliable time-stamping service, adding an extra requirement for its integration [20]. Another disadvantage is the limited number of supported signatures per key pair. Like non-hierarchical HBS schemes, the maximum number of signing keys per key pair is practically limited to 2^{20} by the key generation time. Finally, the efficiency of the scheme has been shown only for a limited number of signatures per key pair [19].

The final reason to follow the BPQS approach is because BPQS can be considered an XMSS-T variant with a chopped tree. Therefore, the security properties of XMSS-T are easily transferable to anything based on BPQS. The security properties of XMSS-T are both well-understood, proven for multiple variants, and based on minimal security assumptions [50, 65, 48].

In addition, in the construction of MBPQS, the internal structures from XMSS-T are used instead of those in LMS. The main reason for this are the weaker security assumptions on which XMSS-T relies, compared to LMS [50, 55]. An extensive comparison of the two schemes is given in Section 3.1.8.

4.2 Authentication Structure

MBPQS is a stateful HBS scheme associating multiple WOTS-T keys with a single public key using a key authentication structure, called the *MBPQS tree* hereafter. This tree is designed such that WOTS-T signatures on messages in segregated (blockchain) channels can be authenticated to a single public key. Additionally, the design enables the distribution of (parts of the) authentication paths of signatures over the blockchain, such that it can be omitted from the signature. The missing parts can be retrieved from earlier verified messages, reducing the size of the signatures. To achieve this unique set of properties, MBPQS combines the authentication structure designs of XMSS-T and BPQS.

In Figure 4.1, an MBPQS tree is depicted. The perfect binary tree at the top of the structure is an XMSS-T tree, called the *root tree* henceforth. The leaves of this tree are compressed WOTS-T verification keys. Their corresponding signing keys are used to compute so-called *root signatures*, which sign the root node of key channels. A key channel is a sequence of one or more chained BPQS-FEW trees, which are called *chain trees* in the MBPQS structure. The root node of the first-level chain tree forms the root of the key channel, which is signed with a root signature. The right-side leaves of the chain trees in a key channel are used top-down to create *message signatures* (i.e., signatures on transactions or blocks) in a blockchain. Once the last right-side leaf is used, the left-side leaf at the bottom layer of the chain tree is used to sign the root node of a newly generated chain tree. This signature is called a *growth signature* and can be appended to the preceding or following message signature, depending on the implementation.

Initially, the authentication structure consists only of the root tree, and key channels are only generated when they are required to avoid unnecessary computational effort during the initial key generation. Likewise, a key channel initially consists of a single chain tree, and consecutive chain trees are generated and appended when all message signing keys are used in the corresponding channel.

To reduce the number of generated keys that will never be used to sign messages, while at the same time mitigating the overhead of growth signatures, the height of chain trees can be determined dynamically for each level. The initial chain tree in each key channel has a fixed height corresponding to the expected number of keys that will be used in every blockchain channel. The height of each subsequent chain tree can be determined according to a predefined growth function. In the example in Figure 4.1, the height of the chain tree is defined as $h + (l - 1)g$, where h is the height of the initial chain tree, l is the chain tree level, and g is a chosen growth factor.

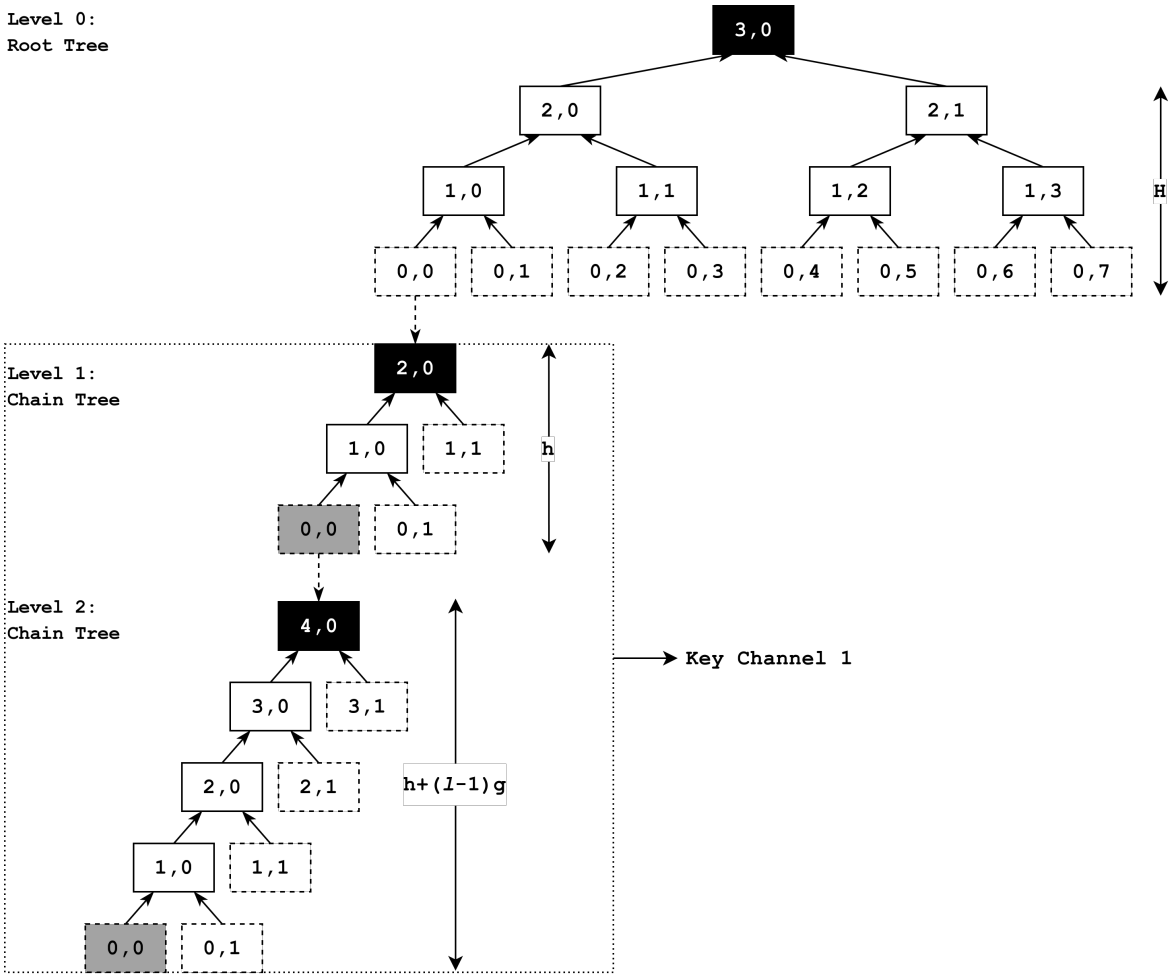

Figure 4.1: MBPQS authentication structure. The black node at the top becomes part of the MBPQS public key. Nodes with a dashed border are compressed WOTS-T verification keys. Dashed arrow lines indicate that the corresponding signing key is used to sign the node the arrow points to. The black nodes with index $(2, 0)$ and $(4, 0)$ are the root nodes of the chain trees in key channel 1. The grey nodes with dashed lines are used to sign additional chain trees.

4.3 Parameters

Below, the parameters of MBPQS are listed with a description of their influence on the scheme.

$w \in \mathbb{N}, w > 1$: The Winternitz parameter used in WOTS-T. The value of w is a space-time trade-off: a higher value of w results in smaller signature sizes but also in slower signing, verification, and key generation; it has little effect on security [48, 14].

$n \in \mathbb{N}$: The security parameter in bytes. It defines the length of the message digests, as well as each node in the tree. Additionally, it defines the security parameter in WOTS-T, determining the length of the one-time signing keys, verification keys, and signatures. Since the output length of the used hash function equals n, either a hash function defines n, or n defines which hash function is used. A higher value of n results in more security in bits, but also in larger signature and key sizes, and in increased computational effort for the signature algorithms.

$H \in \mathbb{N}$: The tree height of the root tree excluding the root node. It determines the number (2^H) of available WOTS-T keys used to add key channels. A higher value of H results in a longer initial key generation time, a larger MBPQS private key, but also in more keys used to add key channels.

$c \in \{0, 1\}$: The key channel caching option, which is a time-space trade-off. If $c = 0$, caching is disabled and during signing, the chain tree must be recomputed up to the left-side node at the height of the leaf currently used. If $c = 1$, caching is enabled and the left-side nodes in the highest-level chain tree of key channels are stored, reducing signing times at the cost of storage space.

$h \in \mathbb{N}$: The tree height of the initial (first-level) chain trees plus one, which equals the number of associated WOTS-T keys in each chain tree. A higher value of h results in longer key generation times for all chain trees, but also in more keys to sign messages before a new chain tree needs to be appended, resulting in fewer growth signatures. If key channel caching is disabled, a higher value of h also results in longer signing times.

$g \in \mathbb{N}$: The chain tree growth factor. As mentioned, the height of a chain tree is defined as: $h + (l - 1)g$ where g specifies by how many signatures each consecutive chain tree grows compared to its predecessor. If $g = 0$, each chain tree in a key channel has the same height. A higher value of g results in faster chain tree growth, and therefore in fewer growth signatures, but also in longer key generation times for consecutive chain trees, and increased signing times if key channel caching is disabled.

4.4 Key State Management

Since MBPQS combines multiple OTS keys, it is vital to manage the state of which OTS keys are used, and which are available. For this reason, the keys in the structure are being used in a predefined order. The keys corresponding to the leaf nodes of the root tree are used from left to right (i.e., from index $(0, 0)$ to index $(0, 2^H - 1)$)), and the keys in the chain trees from top to bottom (i.e. from height $h + (l - 1)g - 2$ to height 0). Despite the blockchain being used to store previous signatures, the current key state cannot be derived from the

blockchain. For example, when a signed message is broadcast, but does not end up in the blockchain for some reason, a peer using the blockchain for its key management could re-use an OTS key, undermining the security of the scheme. Furthermore, when a peer wants to sign multiple transactions rapidly after each other, it would need to wait for the blockchain to update, to receive the current key state.

As a consequence, each peer must hold an up-to-date key state stored in the MBPQS private key (SK). For the root tree, a single value Idx is stored, denoting the index of the first unused leaf node. For each key channel, three values, $SeqNo$, $ChainSeqNo$, and $Layers$, are stored. The value $SeqNo$, is an unique sequence number for each leaf in a key channel, starting at 0 for the node with height $h + (l-1)g - 2$ in the first-level chain tree. $ChainSeqNo$ is the sequence number denoting the first unused leaf in the highest-level chain tree in a key channel starting at 0 for the node at height $h + (l-1)g - 2$. This value is only unique within a chain tree and resets to 0 when a new chain tree is added. Finally, $Layers$ denotes the level of the current chain tree used for signing, corresponding to the chain tree levels in Figure 4.1.

4.5 Algorithms

The algorithms presented in this section form the MBPQS scheme. The algorithms deviate from the algorithms of a digital signature scheme according to Definition 2.1.4 for several reasons. First of all, MBPQS is a so-called *key evolving signature scheme* [5], in which the private key evolves over time, while the public key remains the same. Therefore, multiple algorithms in MBPQS also return, besides a signature, an updated state of the private key. Secondly, because users can sign in multiple segregated key channels, the signing algorithm takes a parameter specifying the key channel to sign in. This is also the case for the algorithm used to add a new chain tree to the specified key channel. Thirdly, because signatures are authenticated to a previous verified node, the algorithms used to verify message and growth signatures have an anchor parameter which denotes the last verified node in a key channel. Finally, because key channels and chain trees are generated and appended on the fly, additional algorithms are specified besides the three mentioned in the definition of a digital signature scheme in Definition 2.1.4. However, it should be noted that these extra algorithms could be combined in the signing and verification algorithms, resulting in the ordinary algorithm triplet.

The algorithms in MBPQS share logic with those in XMSS-T and its corresponding OTS scheme, WOTS-T. Therefore, to avoid redundancy in the pseudocode below, routines with identical functionality to XMSS-T or WOTS-T are denoted as $xmsst.func(..)$, and $wotst.func(..)$, respectively. Furthermore, the hash function addressing schemes from XMSS-T, as discussed in Section 2.3.1, are adopted in MBPQS. However, the addressing schemes are adapted to fit their purpose for MBPQS, which is discussed in Section 4.1. Therefore, in the pseudocode of the algorithms discussed hereafter, the hash address operations are stated explicitly when they differ from XMSS-T, or when their omission could cause ambiguity. The construction of a hash address always starts with the all-zero address, denoted as a_0, which is a generic hash address with all bits zeroed. Address fields which

need to be zero (e.g. the Address Type for OTSs) are not explicitly set to zero to avoid redundant pseudocode. Furthermore, for readability, OTS addresses are denoted as a_O, L-tree addresses as a_L, and hash tree addresses as a_T, with an optional index (i) to differentiate between multiple instances. The usage of hash addresses within *xmsst* and *wotst* functions is unchanged.

4.5.1 Key Generation

The algorithm $KeyGen(P) \rightarrow (SK, PK)$, specified in pseudocode in Procedure 1, is used to generate a key pair. It takes the parameters described in Section 4.3 as input and outputs a private key (SK) and corresponding public key (PK). To this end, first, three n-byte random values SK_1, SK_2, and PS, are sampled. The value SK_1 is used as seed to pseudorandomly generate WOTS-T keys. SK_2 is used as seed to generate pseudorandom values to randomize the message hash function in the signing algorithm. PS is the public seed, used in the WOTS-T chaining function and internal tree node computations, as explained in sections 2.3.2 and 2.3.3, where it is called $SEED$. Subsequently, 2^H WOTS-T seed values are pseudorandomly generated. For each of these seeds, a WOTS-T verification key (pk_i) is computed and compressed in a leaf (L_i). An XMSS-T tree is constructed and its root node is added to the public key. The root node is also included in the private key, because it is used in the initial message compression, as explained in Section 4.6. In this way, the private key on its own suffices to sign messages. In both the private key and public key, a parameter set (P) is included such that the signer and verifier use the same MBPQS instance. Furthermore, the private key includes the key state management values discussed in Section 4.4.

Procedure 1: KeyGen: Initial key generation.

input : P `// P = (w||n||H||h||g|c)`
output: (SK, PK)

1 $SK_1, SK_2, PS \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n$
2 **for** $i \in \{0, ..., 2^H - 1\}$ **do**
3 $\quad a_{O_i} \leftarrow a_0.SetOTSAddr(i)$
4 $\quad s_i \leftarrow PRF(SK_1, a_{O_i})$
5 $\quad \_, pk_i \leftarrow wotst.KeyGen(s_i, a_{O_i}, PS)$
6 $\quad a_{L_i} \leftarrow a_0.SetAddrType(1)$
7 $\quad a_{L_i} \leftarrow a_{L_i}.SetLtreeAddr(i)$
8 $\quad L_i \leftarrow xmsst.GenLeaf(pk_i, a_{L_i}, PS)$
9 $\quad C[i] \leftarrow \{ChainSeqNo : 0, SeqNo : 0, Layers : 0\}$
10 **end**
11 $rt \leftarrow xmsst.GenHashTree(l_0..l_h)$
12 $Root \leftarrow rt.node(H, 0)$
13 $Idx \leftarrow 0$
14 $SK \leftarrow (P||Idx||SK_1||SK_2||PS||Root||C)$
15 $PK \leftarrow (P||PS||Root)$
16 **return** (SK, PK)

4.5.2 Signing

Key Channel Addition

Signing and verification of message signatures is always done in a key channel. Therefore, a channel needs to be added using the algorithm $AddChannel() \rightarrow (RootSig, SK)$ before messages can be signed. The pseudocode for this algorithm is presented in Procedure 2. The algorithm takes the private key, and returns both a root signature on a newly created key channel, and an updated version of the private key.

First, a chain tree of height h is generated according to the pseudocode shown in Procedure 3. If key channel caching is enabled, the internal nodes of the chain tree are cached in the private key. Subsequently, the root node of the chain tree is signed with the first unused WOTS-T key pair in the root tree (with index Idx), and the corresponding authentication path for the used leaf is computed. Next, the layer value in the state of the added key channel is increased by one, representing the first chain tree level. The root signature is constructed by concatenating the index of the key channel, together with the signature on its root node, and the corresponding authentication path. Finally, the value in the state of the private key denoting the index of the first unused leaf in the root tree is increased by one, and the root signature and updated private key are returned.

Procedure 2: AddChannel: Initializes and signs a key channel.

input : SK
output: (RootSig, SK)
1 $ct \leftarrow$ `GenChainTree(`$SK.P.h$, SK`)`
2 **if** $SK.c = 1$ **then**
3 $\quad C = C||ct.Cache$ // $C = SK.C[SK.Idx]$
4 **end**
5 $rootNode \leftarrow ct.node(h-1, 0)$
6 $a_O \leftarrow a_0.SetOTSAddr(SK.Idx)$
7 $s \leftarrow PRF(SK.SK_1, a_O)$
8 $sk, \_ \leftarrow wotst.KeyGen(s, a_O, SK.PS)$
9 $\sigma \leftarrow wotst.Sign(rootNode, sk, a_O, SK.PS)$
10 $AuthPath \leftarrow xmsst.AuthPath(sk)$
11 $C.Layers \leftarrow C.Layers + 1$
12 $RootSig \leftarrow (SK.Idx||\sigma||AuthPath)$
13 $SK.Idx \leftarrow SK.Idx + 1$
14 **return** $(RootSig, SK)$

Procedure 3: GenChainTree: Generate a chain tree of a given height.

input : $height$, SK
output: $ct(chainTree)$

$ct.height = height$
for $i \in \{0..(height-1)\}$ **do**
  $a_{O_i} \leftarrow a_0.SetTreeAddr(ChIdx)$
  $a_{O_i} \leftarrow a_{O_i}.SetLayerAddr(C.Layers)$
  $a_{O_i} \leftarrow a_{O_i}.SetOTSAddr(i)$
  $s_i \leftarrow PRF(SK_1, a_{O_i})$
  $\_, pk_i \leftarrow wotst.KeyGen(s_i, a_{O_i}, SK.PS)$
  $a_{L_i} \leftarrow a_0.SetAddrType(1)$
  $a_{L_i} \leftarrow a_{L_i}.SetLayerAddr(C.Layers)$
  $a_{L_i} \leftarrow a_{L_i}.SetTreeAddr(ChIdx)$
  $a_{L_i} \leftarrow a_{L_i}.SetLtreeAddr(i)$
  if $i = 0$ **then**
    $ct.node(0,0) \leftarrow xmsst.GenLeaf(pk_i, a_{L_i}, SK.PS)$
  end
  $ct.node(i-1,1) \leftarrow xmsst.GenLeaf(pk_i, a_{L_i}, SK.PS)$
  $a_{T_i} \leftarrow a_0.SetAddrType(2)$
  $a_{T_i} \leftarrow a_{T_i}.SetLayerAddr(C.Layers)$
  $a_{T_i} \leftarrow a_{T_i}.SetTreeAddr(ChIdx)$
  $a_{T_i} \leftarrow a_{T_i}.SetTreeHeight(i-1)$
  $ct.node(i,0) \leftarrow xmsst.H_k(ct.node(i-1,0), ct.node(i-1,1), a_{T_i})$
end
return ct

Message Signing

The algorithm $SignMsg(SK, ChIdx, Msg) \rightarrow (Sig, SK)$ generates a signature on a message in the key channel with index $ChIdx$ using the private key. Besides the signature, the algorithm also returns an updated state of the private key. The pseudocode of the algorithm is set out in Procedure 4.

First, the message is hashed with a randomized, keyed hash function as discussed in Section 4.6. Subsequently, the WOTS-T signing key corresponding to the first available leaf in the current chain tree is computed and used to sign the message digest. If key channel caching is disabled, the sibling node of the used leaf is computed by generating the chain tree according to Procedure 3. Otherwise, the sibling node is retrieved from the key channel cache. Finally, the returned signature comprises five values to determine the randomization elements and hash addresses, the WOTS-T signature on the message digest, and the authentication node used to validate the signature.

Procedure 4: SignMsg: Signs a message in a key channel.

input : SK, $ChIdx$, Msg
output: Sig, SK

1 $SigIdx \leftarrow (ChIdx||C.SeqNo)$ // $C = SK.C[ChIdx]$
2 $R \leftarrow PRF(SigIdx, SK.SK_2)$
3 $d \leftarrow H_{msg}(R||SK.Root||C.SeqNo||Msg)$
4 $a_O \leftarrow a_0.SetOTSAddr(C.ChainSeqNo)$
5 $a_O \leftarrow a_O.SetLayerAddr(C.Layers)$
6 $a_O \leftarrow a_O.SetTreeAddr(ChIdx)$
7 $s \leftarrow PRF(SK.SK_1, a_O)$
8 $sk, \_ \leftarrow wotst.KeyGen(s, a_O, SK.PS)$
9 $\sigma \leftarrow wotst.Sign(d, sk, a_O, SK.PS)$
10 **if** $c = 0$ **then**
11 $ct \leftarrow$ `GenChainTree(`*SK.P.h + (C.Layers-1) * SK.P.g*`)`
12 $AuthNode \leftarrow ct.node(ct.height - 2 - C.ChainSeqNo, 0)$
13 **else**
14 $AuthNode \leftarrow C.Cache[C.ChainSeqNo]$
15 **end**
16 $Sig \leftarrow (C.ChainSeqNo||C.SeqNo||C.Layer||R||ChIdx||\sigma||AuthNode)$
17 $C.SeqNo \leftarrow C.SeqNo + 1$
18 $C.ChainSeqNo \leftarrow C.ChainSeqNo + 1$
19 **return** (Sig, SK)

Channel Growth

Once all but one keys in a key channel are used, the last key (represented by the grey nodes with index $(0, 0)$ in Figure 4.1) is used to sign the root node of a newly generated chain tree, effectively growing the channel. The algorithm used for this procedure, $GrowChannel(SK, ChIdx) \rightarrow (GrowSig, SK)$, takes a channel index as input, and returns a growth signature on the next chain tree root in the specified key channel. The pseudocode for this algorithm is specified in Procedure 5.

Initially, a chain tree of height $h + (l-1)g$ is generated.[1] If key channel caching is enabled, the existing cache is overwritten with the internal nodes of this chain tree. Afterwards, the root node of the chain tree is signed with the WOTS-T key corresponding to the left-side leaf at the bottom layer of the current chain tree. Since the authentication node of this signature is the previously used leaf, it is not added to the signature. The first three values of the growth signature are to determine the hash addresses for the verifier. The last value, σ, is the WOTS-T signature on the root node (*Root*) of the newly appended chain tree. Subsequently, the state of the key channel is updated by increasing the number of layers in the channel with one, and the chain sequence counter is reset to 0 (because there are 0 leaves

[1] In the pseudocode, the -1 in $(l-1)$ is implicit, because l is increased only after the new chain tree is generated on line 10.

used in the new chain tree). Finally, the algorithm returns the growth signature and the private key with the updated key state.

Procedure 5: GrowChannel: Generates a chain tree, and signs its root node.

input : SK, $ChIdx$
output: $GrowSig$

1 $ct \leftarrow$ `GenChainTree(`$SK.P.h$ + $C.Layers$ * $SK.P.g$`)`
2 **if** $c = 1$ **then**
3 $\quad C.Cache \leftarrow ct.Cache$ `// Overwrite the cache`
4 **end**
5 $Root \leftarrow ct.node(ct.height - 1, 0)$
6 $a_O \leftarrow a_0.SetOTSAddr(C.ChainSeqNo)$
7 $a_O \leftarrow a_O.SetLayer(C.Layers)$
8 $a_O \leftarrow a_O.SetTree(ChIdx)$
9 $s \leftarrow PRF(SK.SK_1, a_O)$
10 $sk, \_ \leftarrow wotst.KeyGen(s, a_O, SK.PS)$
11 $\sigma \leftarrow wotst.Sign(Root, sk, a_O, SK.PS)$
12 $GrowSig \leftarrow (C.ChainSeqNo||ChIdx||C.Layers||\sigma)$
13 $C.Layers = C.Layers + 1$
14 $C.ChainSeqNo = 0$
15 **return** $(GrowSig, SK)$

4.5.3 Verification

Channel Verification

Before a key channel can be used to verify its corresponding signatures, the verifier needs to verify the authenticity of the key channel by verifying the signature on its root node. For this procedure, the algorithm $VerifyChannel(PK, ChanRt, Sig) \rightarrow accept/deny$, as shown in pseudocode in Procedure 6, is used. The algorithm takes the public key, channel root and a root signature as input, and returns whether the signature is accepted for the given channel root.

First, the hash chain of the WOTS-T signature is finished using the channel root and constructed OTS hash address. Then, the corresponding leaf node is computed and together with the authentication path hashed up until the root node is reached, according to the technique used in XMSS-T. The resulting hash digest is compared with the root hash from the public key. If they are equal, the channel root node is verified to be signed by the owner of the private key. Consequently, this verified node becomes the anchor node in the key channel.

Procedure 6: VerifyChannel: Verifies the root signature on a key channel.

input : PK, $ChanRt$, Sig
output: *boolean*

```
1  a_O ← a_0.SetOTSAddr(Sig.Idx)
2  pk = wotst.PkFromSig(ChanRt, Sig.σ, a_O)        // Finishing the WOTS-T chain
   function
3  a_L ← a_0.SetAddrType(1)
4  a_L ← a_L.SetLtreeAddr(Sig.Idx)
5  L ← xmsst.GenLeaf(pk, a_L, PK.PS)
6  RootNode ← xmsst.HashUpTree(L, Sig.AuthPath)
7  if RootNode = PK.Root then
8      return accept
9  else
10     return deny
11 end
```

Message Signature Verification

The algorithm $VerifyMsg(PK, Sig, Msg, Anchor) \rightarrow accept/deny$ takes the public key, a signature, a message, and anchor node as input, and returns whether the signature is valid for the given message. In Procedure 7, the specification of the algorithm is given in pseudocode.

First, the hash digest of the message is computed using the randomization elements derived from the signature. Subsequently, the WOTS-T hash chain function is finished, resulting in a verification key. This key is compressed in a leaf and hashed together with the authentication node from the signature. If the result equals the anchor node, the signature is accepted, and otherwise denied. If the signature is accepted, and the preceding chain of signatures is valid, the verifier has the guarantee that the signature belongs to the public key.

Channel Growth Verification

Before a message signature, signed with a key belonging to a new chain tree, can be verified, the signature on the root node of this new chain tree must be verified. The algorithm used for this, $VerifyGrowth(PK, Sig, ChainRt, Anchor) \rightarrow accept/deny$, shown in pseudocode in Procedure 8, specifies this. It takes the public key, growth signature, root node of the chain tree, and an authentication node as input, and returns whether the signature is valid for the given values.

To this end, first, the hash chain of the WOTS-T signature from the growth signature is finished to retrieve the one-time verification key corresponding to the signature. Subsequently, this verification key is compressed into a leaf and compared with the given anchor node. In this case, the anchor node should be the sibling node of the leaf last used to sign a message in the last used chain tree. Namely, this node is used in the previous signature verification, and can be trusted if that signature is valid. If the anchor node equals the created leaf, the new chain tree can be trusted for future message signature verifications.

Procedure 7: VerifyMsg: Verifies a signature on a message.

input : PK, Sig, Msg, $Anchor$
output: $boolean$

$SigIdx \leftarrow (Sig.ChIdx||Sig.SeqNo)$
$d \leftarrow H_{msg}(Sig.R||PK.Root||SigIdx||Msg)$
$a_O \leftarrow a_0.SetOTSAddr(Sig.ChainSeqNo)$
$a_O \leftarrow a_O.SetLayerAddr(Sig.Layers)$
$a_O \leftarrow a_O.SetTreeAddr(Sig.ChIdx)$
$pk = wotst.PkFromSig(Msg, Sig.\sigma, a_O)$
$a_L \leftarrow a_0.SetAddrType(1)$
$a_L \leftarrow a_L.SetLayerAddr(Sig.Layers)$
$a_L \leftarrow a_L.SetTreeAddr(Sig.ChIdx)$
$a_L \leftarrow a_L.SetLtreeAddr(Sig.ChainSeqNo)$
$L \leftarrow xmsst.GenLeaf(pk, a_L, PK.PS)$
$a_T \leftarrow a_0.SetAddrType(2)$
$a_T \leftarrow a_T.SetLayerAddr(Sig.Layers)$
$a_T \leftarrow a_T.SetTreeAddr(Sig.ChIdx)$
$a_T \leftarrow$
  $a_T.setTreeHeight(PK.P.h + PK.P.g * (Sig.Layers - 1) - 2 - Sig.ChainSeqNo)$
$Parent \leftarrow xmsst.H_k(Sig.Authnode, L, a_T)$
if $Anchor = Parent$ **then**
    return *accept*
else
    return *deny*
end

Procedure 8: VerifyGrowth: Verifies a signature on a new chain tree.

input : PK, Sig, ChainRt, Anchor
output: boolean

$a_O = a_0.SetLayerAddr(Sig.Layers)$
$a_O = a_O.SetTreeAddr(Sig.ChIdx)$
$a_O = a_O.SetOTSAddr(Sig.ChainSeqNo)$
$pk = wotst.PkFromSig(ChainRt, Sig.\sigma, a_O)$
$a_L \leftarrow a_0.SetAddrType(1)$
$a_L \leftarrow a_L.SetLayerAddr(Sig.Layers)$
$a_L \leftarrow a_L.SetTreeAddr(Sig.ChIdx)$
$a_L \leftarrow a_L.SetLtreeAddr(Sig.ChainSeqNo)$
$L \leftarrow xmsst.GenLeaf(pk, a_L, PK.PS)$
if $L = Anchor$ **then**
    return *accept*
else
    return *deny*
end

4.6 Security Considerations

The security of MBPQS is based on the security of XMSS-T, which has a classical security proof in the standard model (SM) and a quantum-security proof in the quantum random oracle model (QROM) [50, 12]. The proofs show that an attacker has to break the generic security properties of a hash function to break the security (EU-CMA) of the scheme, and only breaking the collision resistance is not sufficient. For the signature scheme to be secure, the internal functions must meet certain security requirements [48]. First of all, the keyed hash function used in WOTS-T and the one used to compute the internal tree nodes must be multi-function multi-target second-preimage resistant. In MBPQS, these functions are implemented identically to those in RFC 8391, which meet the aforementioned requirements. This is also the case for the PRF used both to (pseudo)randomize the initial message hash function (H_{msg}) and to pseudorandomly generate WOTS-T seeds, which must be post-quantum secure. Finally, the initial message hash function H_{msg} must be a post-quantum multi-target extended target collision-resistant keyed hash function. In MBPQS, a slightly changed version of the initial message compression function from XMSS-T is used, which is covered in Section 4.6.2. Furthermore, MBPQS uses the PRF from RFC 8391 to generate bitmasks and keys to randomize hash function calls. Therefore, the security proof of XMSS in the standard model from [14] does not fully apply. Namely, due to technicalities in this security proof, the PRF is modeled as a random oracle in the security proof [50]. The fact that the chain trees in MBPQS are essentially unbalanced Merkle trees does not invalidate the aforementioned security proofs. The proofs do not concern the shape of the trees, but rather the way in which they are constructed. Chaining stacked trees in MBPQS is also done in XMSS-T^{MT}, which is proven EU-CMA in the standard model in [49] by combining the security proofs in [14] and [65].

As discussed, the methods yielding the security properties for XMSS-T are copied where possible in MBPQS. Nonetheless, some of these methods are adapted to fit our design. For each of these adaptations, the rationale is discussed and it is explained how the security properties of XMSS-T are retained.

4.6.1 Hash Addressing

To achieve multi-target resistance, each hash function call in MBPQS is made unique, according to the multi-function multi-target security definitions in Definition 2.1.2. Therefore, the hash function addressing schemes from XMSS-T^{MT}, explained in Section 2.3.1, are adopted and slightly changed in MBPQS. In Table 4.1, these changes compared to the hash addressing schemes in XMSS-T are shown.

The channel index word differentiates the hash address in function calls between key channels. Subsequently, the chain level word differentiates them for each chain tree within a key channel, since each chain tree has a unique level. Finally, the hash function calls for the leaf generation and WOTS-T computations are made unique by using the chain tree sequence number, which is unique within a chain tree. The rest of the hash addressing works identical to XMSS-T. For the root tree, the chain level address is set to 0, which differentiates its hash addresses to those in the key channels, as chain tree levels start at 1.

Table 4.1: Changed hash address words in MBPQS compared to hash addresses in XMSS-T.

OTS Hash Address	**L-tree Address**	**Hash Tree Address**
chain level	chain level	chain level
channel index	channel index	channel index
-	-	-
chain sequence no	chain sequence no	-
-	-	-
-	-	-
-	-	-

4.6.2 Initial Message Hash

As mentioned, the initial message hash function H_{msg} from XMSS-T as specified in RFC 8391 is slightly changed for MBPQS. The function is used to compress messages before signing, as WOTS-T needs a fixed-size input. The function is defined as follows in RFC 8391:

$$H_{msg} = HashF(toByte(2, n)||KEY||M),$$

where $HashF$ is $SHA2$-$\{256, 512\}$ or $BLAKE$-$\{128, 256\}$, depending on the chosen hash function for XMSS. For MBPQS, currently only the $SHA2$ variants are implemented. The function $toByte(x, y)$ returns a y-byte string containing the binary representation of x in big-endian byte order. Furthermore, KEY is computed as

$$KEY = (R||PK.root||toByte(idx, n)), \text{where}$$
$$R = PRF(toByte(3, n)||SK_2||toByte(idx, n)).$$

Here, $PK.root$ is the root node of the authentication structure, which is part of the public key, idx is the index of the current signature, and SK_2 is the n-byte PRF seed, to randomize the message hashing. The root node is added to the KEY to prevent multi-user attacks, and the idx to prevent multi-target attacks against H_{msg}. In this way, an attacker can only attack a specific hash value, belonging to a predefined public key.

In MBPQS, the H_{msg} function works identical to XMSS-T, but to guarantee the uniqueness of idx for each message signature, it is defined as

$$idx = ChIdx||seqNo,$$

where $ChIdx$ is the key channel index, corresponding to the index of the root leaf which signed the root node of the first-level chain tree, and $seqNo$ is the sequence number of the message

signature, unique in the key channel. In the current implementation, idx is implemented as a 64-bit value where the 32 most significant bits encode the channel index, and the 32 least significant bits encode the sequence number of the message signing keys. For practical reasons, in the current implementation, the maximum number of key channels and message signatures within them is both 2^{32}. Nonetheless, since idx is encoded as an n-byte variable, it is trivial to increase these numbers to virtually unlimited numbers.

Chapter 5

Performance Analysis

In the following chapter, the performance characteristics of MBPQS are analyzed and compared with the state-of-the-art stateful HBS scheme XMSS-T. First, the sizes of the keys and signatures are considered. Subsequently, a model for the complexity of the algorithms is built and verified, which is used as an aid to interpret results from the performance analysis. Lastly, the practical performance of MBPQS is examined by timing and benchmarking the execution times of the implemented algorithms.

5.1 Context for Analyses

As mentioned in the introduction of this chapter, the results from the MBPQS analyses are compared with XMSS-T to put the numbers found into perspective. XMSS-T is used as a benchmark, because it is one of the only two stateful HBS schemes both formalized in an RFC and proposed for standardization by the NIST [48, 85]. Moreover, MBPQS and XMSS-T share a significant part of their internal structures, and have similar security properties, resulting in a fair comparison. Nonetheless, it is stressed that MBPQS is a digital signature scheme specifically designed for certain use cases, while XMSS-T is a universally applicable stateful HBS scheme. Since the other stateful HBS scheme proposed to be standardized, LMS, uses both a different OTS scheme and tree construction method, which require stronger security assumptions, comparing it with MBPQS is not as useful [68].

In the comparison between MBPQS and XMSS-T, the parameter set used in the RFC for comparisons is used here as well [48]. The Winternitz parameter $w = 16$, and security parameter $n = 32$ are used, with SHA2-256 as underlying hash function for the schemes. The XMSS-T variants considered are the single-tree variant XMSS-SHA256-20 and the multi-tree variants XMSS-MT-SHA256-20/2, XMSS-MT-SHA256-40/2, and XMSS-MT-SHA256-40/4. In the first two schemes, a maximum of 2^{20} signatures can be created with one key pair, while the latter two schemes support a maximum of 2^{40} signatures per key pair. For reference, suppose a key pair is used to sign just a single signature per second, it would take only about 12 days before running out of signatures for the first two schemes, and approximately 35,000 years for the last two variants. Hence, the first two schemes are selected to show how MBPQS compares to XMSS-T variants with a fairly limited amount of signatures. The second two schemes are chosen because they result in the smallest signatures for a default XMSS-T variant with arguably enough signatures for most use cases of MBPQS. XMSS-T variants supporting up to 2^{60} signatures per key pair require larger signature sizes and longer algorithm execution times, which would put them at a disadvantage compared to MBPQS. Even though MBPQS does in fact support up to 2^{84} signatures, such numbers will most likely not be required in practice, considering the use cases of MBPQS. Therefore, comparing the schemes for more than 2^{40} supported signatures contributes not much of use.

5.2 Key and Signature Sizes

In this section, the key and signature sizes of MBPQS are analyzed and compared with the aforementioned XMSS-T variants.

5.2.1 Public Key

The composition of the MBPQS private and public keys are shown in tables 5.1 and 5.2, respectively. As shown, the public key length is $18 + 2n$ bytes, where the 18 bytes are due

to the size of a full parameter set. Furthermore, the public key consists of an n-byte public seed, and the n-byte root node of the root tree, where n is the security parameter.

Table 5.1: Public key fields and their sizes in bytes.

Size (B)	18	n	n
Fields	P	PS	Root

5.2.2 Private Key

The private key contains the same fields as the public key plus two n-byte seed values, 4 bytes for the state of the root tree, and 12 bytes for each key channel, denoting its state. The number of key channels in use for the private key is denoted as $|c|$. The total length of the private key is $22 + 4n + |c| \times 12$ bytes.

The key channel cache is not considered part of the private key, because it can be re-computed at any time, and therefore, it does not have to be stored in persistent storage. Furthermore, the cache does not contain secret information, nor does it influence the security of the scheme in any other way. It is merely a space-time trade-off for the computation of the authentication node in signatures.

Table 5.2: Private key fields and their sizes in bytes.

Size (B)	18	4	n	n	n	n	$\|c\| \times 12$
Fields	P	Idx	SK_1	SK_2	PS	Root	C

5.2.3 Signatures

There exist three types of signatures in MBPQS: root signatures, message signatures, and growth signatures. For every key channel, a single root signature is required to sign the root of the top-most chain tree. Furthermore, for every additional chain tree in the key channel, a growth signature is used to sign its root node. Each of these signature types have their own unique set of fields. In Tables 5.3, 5.4, and 5.5, these fields and their sizes in bytes are shown for each signature type. Here, n is the security parameter, H is the height of the root tree, and ℓ is the number of elements in the WOTS-T signature, as explained in Section 2.3.2. The composition of each signature type is discussed in Section 4.5.2.

In Table 5.6, the sizes in bytes for each signature type are given for specified values of the Winternitz parameter w and the security parameter n. The values for $w \in \{4, 16\}$ are chosen because $\log(w)$ bits are signed at the same time in WOTS-T. Ergo, the values $w = 4$ and $w = 16$ result in respectively 2 and 4 bits being signed simultaneously. According to [48], "these values yield optimal trade-offs and easy implementation". Additionally, the value $w = 256$ is chosen to show what happens if the message is signed per byte, effectively

Table 5.3: Root signature fields and their sizes in bytes.

Size (B)	4	$\ell \times n$	$H \times n$
Fields	SeqNo	σ	AuthPath

Table 5.4: Message signature fields and their sizes in bytes.

Size (B)	4	4	4	4	n	$\ell \times n$	n
Fields	ChIdx	Layer	ChainSeqNo	SeqNo	R	σ	AuthNode

Table 5.5: Growth signature fields and their sizes in bytes.

Size (B)	4	4	4	$\ell \times n$
Fields	ChIdx	Layer	ChainSeqNo	σ

doubling the number of bits signed simultaneously compared to $w = 16$. It shows that a quadratic increase in the value of w approximately cuts the signature size in half. The values for n correspond to the digest length for the hash functions SHA-256 and SHA-512, which are used in our implementation. These hash functions are used because these are also used in the XMSS-T reference code, BPQS reference code, and their corresponding papers. This makes for easier comparison between the different schemes. In the comparison, the root tree height $H = 14$ is used, but it is trivial to see from Table 5.3 that any other value of H, defined as H', would only change the size of root signatures with $(H' - H)n$ bytes, because H only influences the size of the authentication path for the root signatures.

5.2.4 Comparison to XMSS-T

Public Key Size

The public key size in MBPQS is almost identical to the one in XMSS-T^{MT}. The only difference is that the parameter set in XMSS-T^{MT} is encoded in a 4-byte object identifier (OID), while MBPQS currently uses 18 bytes, which could easily be reduced to 4 bytes as well. However, this method would require predefined parameter sets, reducing the flexibility of the scheme.

Private Key Size

Assuming the pseudorandom generation of keys, the XMSS-T^{MT} private key consists of the SK_1, SK_2, PS, *Root* fields from Table 5.2, and a $\lceil \frac{T}{8} \rceil$-byte index specifying the first available signing key, where T is the total height of the XMSS-T^{MT} tree. In the MBPQS private key, an additional $|c| \times 12$ bytes are added to maintain the state of the key channels.

Table 5.6: Signature sizes in bytes for different values of the Winternitz parameter (w) and security parameter (n) for $H = 14$.

	$w = 4$	$w = 16$	$w = 256$
$n = 32$			
Root signature	4708	2596	1540
Message signature	4336	2224	1168
Growth signature	4268	2156	1100
$n = 64$			
Root signature	17604	9284	5124
Message signature	16848	8528	4368
Growth signature	16716	8396	4236

Signature Sizes

The signature sizes for XMSS-T and MBPQS are shown in Table 5.7 for the aforementioned parameters. It is clear to see that MBPQS message signatures are significantly smaller than those in XMSS-T. However, for each key channel, a single MBPQS root signature is required, and after each $(h + g(l-1) - 1)$ message signatures, a MBPQS growth signature is added. The total size of a MBPQS signature, combining all three signature types, in a chain tree of a key channel is:

$$\frac{|RootSig| + |GrowSig|(L-1)}{S} + |MsgSig|,$$

where $|...Sig|$ denotes the size of the signature of a certain type, S the total number of message signatures in a key channel, and $L \leq S$ the number of layered chain trees in a key channel, which is the smallest upper-bound of l, such that

$$\sum_{l=1} (h + g(l-1) - 1) > S.$$

For the parameters considered earlier, this would result in an average signature size of

$$\frac{440 + 2156L}{S} + 2224.$$

For a sufficient large number of signatures in combination with large chain trees, the total signature size in MBPQS converges towards 2224 bytes. Of course, this is unrealistic in practice, as this would result in either very long signing times or huge key channel caches. Nonetheless, even in the worst-case scenario, where for each signature a chain tree is added, the average signature size converges to $2156 + 2224 = 4380$ bytes, cutting more than 20% of the signature size compared to the multi-tree XMSS-T variant with the smallest signature size. Furthermore, already for a total chain tree height of 10, the average signature size

Table 5.7: XMSS-T and MBPQS signature sizes for $w = 16$ and $n = 32$.

Scheme	Signature size (B)
MBPQS (Root Sig)	2596
MBPQS (Message Sig)	2224
MBPQS (Growth Sig)	2156
XMSS-T-SHA256-20	2820
XMSS-T^{MT}-SHA256-20/2	4963
XMSS-T^{MT}-SHA256-40/2	5605
XMSS-T^{MT}-SHA256-40/4	9893

in MBPQS converges to 2464 bytes. This is approximately 12.6% less than the cheapest XMSS-T variant supporting only up to 2^{20} signatures, and less than half of the smallest signature size for a variant supporting up to 2^{40} signatures. For a chain tree height of 100, the signature size converges to 2250. With the size of the WOTS-T signature being 2144 bytes, this means that more than 95% is due to the OTS.

5.3 Complexity

In this section, the complexity of MBPQS is analyzed, and a model of the computational costs of its algorithms is provided. This complexity model can be used to compare the costs of subroutines within the scheme, or to compare the scheme with other HBS schemes. In this thesis, the complexity model is used to validate the results found in the practical performance analysis, and to interpret these results in the discussion in Chapter 6.

In other work, the theoretical performance of HBS schemes is compared using the number of required hash function operations for each algorithm [55]. To be more precise, the complexity is expressed as the number of compression operations within the used hash function. This metric is used because it turned out that the vast majority of the computational effort in HBS schemes is due to compression operations. Furthermore, the cost of other computations is implementation-specific, while the number of hash compression operations is the same for all implementations using the same hash function.

5.3.1 Model Verification

Before adopting the number of hash compression operations as a metric to model the complexity of MBPQS, first its applicability is verified. To this end, the percentage of the total computation times caused by hash operations is measured for each algorithm in the MBPQS implementation. These measurements are obtained using the Golang CPU profiler[1] after

[1] https://golang.org/pkg/runtime/pprof/

benchmarking each algorithm using all 4 cores of an Intel Core i7-4500U CPU, clocked at 3.0GHz. Subsequently, the average percentage is taken from benchmarks for all combination of the parameters: $w = \{4, 16\}$, $n = 32$, $h = \{2, 10^1, 10^2, 10^3, 10^4\}$, $H = \{8, 12, 16\}$, $c = \{0, 1\}$. The number of algorithm executions (N) is dynamically determined by the Golang benchmark tool, which is described as follows: "During benchmark execution, N is adjusted until the benchmark function lasts long enough to be timed reliably".[2] Hence, the number of algorithm executions range from 5 to 300,000, from which the average percentage is calculated. The results from these measurements are shown in Table 5.8.

Indeed, the vast majority – about 85% – of the total computation time of the algorithms in MBPQS is due to hash function operations. This is a conservative estimate, as the computation times of memory moves, which accounted for approximately 5% of the total computation times, are assumed to be completely unattributable to hash function operations. Nonetheless, 85% corresponds closely with findings in other literature where similar HBS schemes were tested [55, 86].

Table 5.8: Percentage of total computation time due to hash function operations for different MBPQS algorithms.

Algorithm	Hash Operations
KeyGen	83.9%
AddChannel	85.6%
SignMsg	84.7%
GrowChannel	85.3%
VerifyChannel	83.8%
VerifyMessage	84.6%
VerifyGrowth	85.6%

5.3.2 Complexity Model

To calculate the number of hash compression operations in each MBPQS algorithm, first the number of compression operations is determined for several subroutines. Hereinafter, hc denotes the unit specifying the number of hash compression operations. In the calculations, the keyed hash function implementations from RFC 8391 using SHA2-256 are considered, as these are used in MBPQS by default. Furthermore, these implementations also resemble the default for XMSS-T [48], defining the benchmark against which the performance of MBPQS is measured.

SHA2-256 is based on the Merkle-Damgård construction, in which the input is divided in blocks that are processed sequentially [70, 73]. Therefore, the state of the hash function, after

[2] https://golang.org/pkg/testing/

processing a certain number of blocks, can be stored. Whenever multiple hash function invocations share an identical block-sized (64 bytes) input prefix, the state of the hash function after processing this prefix can be stored and reused. In MBPQS and XMSS-T, many hash invocations share the same prefix, and therefore, using this optimization obviates many hash compression operations. Both in the implementation of MBPQS and the XMSS-T implementations used in the practical performance analysis, this optimization is used. Therefore, in our model, it is assumed that hash compression operations on input with a common prefix are precomputed, except in the initial message compression function. Precomputing this prefix would add n bytes to the scheme state, and saves only 2 hc during signing, and 1 hc during verification, which is considered negligible.

The subroutines considered in our model are defined as follows:

(i) Creating a WOTS-T signing key takes $1 + 2\ell$ hc: Pseudorandomly generating a seed from SK_1 takes 1 hc since the PRF has a common prefix. Additionally, expanding the seed in an $\ell \times n$ key takes 2ℓ hc, because the unique seed is part of the function prefix.

(ii) A single Winternitz chain step in WOTS-T takes 4 hc: Generating the bitmask and key cost 1 hc each, since their PRFs have a common prefix. Additionally, the function $\mathcal{F}$ costs 2 hc, because the generated key makes its prefix unique.

(iii) Computing a WOTS-T verification key from a signing key takes $4\ell(w-1)$: Each of the ℓ key elements go through $w-1$ Winternitz chain steps, costing 4 hc each.

(iv) Hashing two tree nodes in a parent node in any MBPQS tree takes 6 hc: Generating the key and two bitmasks takes 3 hc, as the used PRFs have a common prefix. Additionally, the keyed hash function takes 3 hc due to the random key in the prefix and the $2n$ input (3rd block compression is due to SHA2-256 padding [73]).

(v) Compressing a WOTS-T verification key in a leaf using an L-tree takes $6(\ell-1)$ hc because $(\ell-1)$ L-tree nodes are generated.

(vi) Generating a leaf from SK_1 takes $1 + 2\ell + 4\ell(w-1) + 6(\ell-1)$ hc, as the subroutines listed in (i), (iii), and (v) are performed consecutively.

(vii) Creating a WOTS-T signature from a signing key takes $\frac{4\ell(w-1)}{2}$ hc: $\frac{w-1}{2}$ Winternitz chain steps are taken on average[3] for each of the ℓ key elements.

(viii) Computing a WOTS-T verification key from the signature to verify the signature takes $\frac{4\ell(w-1)}{2}$ hc, because on average, the other 'half' of the Winternitz chain is finished up to step $(w-1)$.

(ix) Compressing the initial message before signing takes $4 + \lceil \frac{m+9}{64} \rceil$ hc during signing, and $2 + \lceil \frac{m+9}{64} \rceil$ hc during verification, where m is the message length in bytes, 64 is the SHA2-256 block size in bytes, and 9 the minimal padding in $SHA2-256$ [73].

[3]This includes the WOTS-T checksum, of which the average number of steps cannot be assumed, but it is close [55].

Computing the randomization element R takes 2 hc, plus 3 hc to compress the n-byte function padding and the $3n$ byte key. During verification, R is derived from the signature, hence the $2hc$ difference.

5.3.3 Complexity Model Verification

To verify the proposed complexity model, the cost ratio between subroutines according to the model are compared with the cost ratios according to real life measurements. First, the cost of different sets of the earlier stated operations are calculated according to our model for $w = 16$, $\ell = 67$, $n = 32$, and $m = 512000$. The results from this are shown in the second column of Table 5.9. The considered operation sets are shown in the first column for each experiment. The operations (ii) and (iv) from our model are not covered separately, as these are embedded in other operations. Subsequently, the computation times for each experiment are measured, and the results are presented in the third column. Again, these measurements are from tests on an Intel Core i7-4500U CPU clocked at 3.00GHz, using 4 cores and 4 threads and using the aforementioned measuring methods, except this time for a single parameter set in order to make comparing possible. Finally, the relative complexity ratio between the routines is shown in the fourth and fifth columns, according to the model and the measurements, respectively. Here, the first operation set from the first experiment is used as point of reference to compute the ratios in other experiments. As shown in the table, the modeled complexity ratio corresponds very closely to the measured complexity ratio between different routines. In the right-most column, the "accuracy" of the model according to the measured ratios is given for each experiment. Even in the worst case, the modeled ratio is 94.3% of our measured ratio.

Table 5.9: Modeled and measured costs and ratios of several operation sets for $w = 16$, $n = 32$, and $m = 512000$.

Operations	Model hc	Measured time	Model ratio	Measured ratio	Model accuracy
(i)	135	0.038 ms	1.0	1.0	100%
(i)+(iii)	4155	1.231 ms	30.8	32.7	94.3%
(v)	396	0.112 ms	2.9	2.9	100%
(vi)	4551	1.357 ms	33.7	35.7	94.4%
(vii)+(i)	2145	0.621 ms	15.9	16.3	97.5%
(viii)	2010	0.581 ms	14.9	15.3	97.4%
(ix) - signing	805	0.227 ms	6.0	6.0	100%

5.3.4 Complexity Analysis

Using the model from Section 5.3.2, the complexity of the MBPQS algorithms, as presented in Section 4.5, are determined below.

Initial Key Generation
To generate the MBPQS key pair, 2^H leaves are generated from SK_1, and then $(2^H - 1)$ internal tree nodes are computed up to the root node, for a total cost of 2^H(vi)+$(2^H - 1)$(iv) *hc*.

Key Channel Addition
To generate a key channel, h leaves are generated from SK_1, and subsequently $(h - 1)$ internal tree nodes are computed up to the root node, which is signed using a generated WOTS-T signing key, resulting in a total cost of h(vi)+$(h - 1)$(iv)+(i)+(vii) *hc*.

Message Signing
To sign a message with caching enabled, the message is compressed, after which a WOTS-T signing key is generated and the signature on the compressed message is computed, resulting in a total cost of (ix)+(i)+(vii) *hc*.

When caching is disabled, additionally the chain tree needs to be created up to the sibling node of the leaf node used for signing. A worst-case scenario is considered in which the entire chain tree needs to be created, corresponding to computing the sibling node of the highest leaf node. The additional cost for signing when caching is disabled is h(vi)+$(h - 2)$(iv) *hc*.

Channel Growing
To append a new chain tree to a key channel, a chain tree, consisting of $h + g(l - 1)$ leaves, is generated from SK_1, after which the $h + g(l - 1) - 1$ internal tree nodes are computed up to the root node. Subsequently, a WOTS-T signature is computed over this root node using a generated WOTS-T signing key, resulting in a total cost of $(h + g(l - 1))$(vi)(iv)-(iv)+(i)+(vii).

Key Channel Verification
To verify a key channel, the WOTS-T verification key is computed from the signature, which is compressed in a leaf. Using this leaf and the authentication path from the signature, the tree is hashed up to the root node, which is compared to the root node from the public key. The total cost to verify a channel signature is (viii)+(v)+H(iv) *hc*.

Message Verification
To verify a message signature, first, the message is compressed using the initial message hash function. Subsequently, the WOTS-T verification key is computed from the signature, using the compressed message. Then, this verification key is compressed using an L-tree, and hashed together with the authentication node. Finally, this result is compared with the anchor node for the key channel. The total cost to verify a message signature is (xi)+(viii)+(v)+(iv) *hc*.

Growth Verification
To verify a growth signature, first the WOTS-T verification key is computed from the signature, which is compressed using an L-tree. The compressed key and the authentication node, retrieved from the signature, are hashed together to generate their parent node. Finally, this node is compared to the anchor node, resulting in a total cost to verify a growth signature of (viii)+(v)+(iv) hc.

5.4 Practical Performance

To evaluate the practical performance of MBPQS, a standalone implementation[4] is written in the Go programming language. The WOTS-T implementation, and parts of the XMSS-T tree generation, are taken from the Go implementation of XMSS-T^{MT} according to RFC 8391 by Bas Westerbaan.[5] The MBPQS implementation covers the parameter sets $n \in \{32, 64\}$, and $w \in \{4, 16, 256\}$, using SHA2-256, and SHA2-512. For the parameters H, h, and g, the values can theoretically be chosen to be up to 2^{32}, but this would lead to years of key generation times in both the root tree and chain trees. Practical values for these parameters can be derived from Table 5.10.

In the implementation of MBPQS, the user can specify the number of threads used during the program execution. In the benchmarks, all threads in the CPU were used because it resembles how the scheme would typically be used in practice. More specifically, all benchmark results are retrieved using the 'testing' package from the Golang standard library on an Intel Xeon E5-2697A v4 CPU clocked at 2.60GHz, using 4 cores, and 8 threads.

5.4.1 Key Generation

In Table 5.10, the initial key generation and chain tree generation times of MBPQS are given for different combinations of the Winternitz parameter w, the root tree height H, and initial chain tree height h. For the chain tree generation times, the growth factor g is not considered, as it only changes the total height of the chain tree, which is already variable in h. One could consider $h' = h$ as the result of the total chain height function $h + g(l-1)$ for all compositions where $h + g(l-1) = h'$.

In both the initial key generation, and the chain tree generation, the generation times increase with a factor of almost 2 for $w = 16$ instead of $w = 4$, and increase with approximately a factor 9 for $w = 256$ instead of $w = 16$.

The left plot in Figure 5.1 shows the relationship between the root tree height H, and the initial key generation time. The vertical axis is in logarithmic scale, because the root tree height is logarithmic proportional to the number of keys generated. The plot shows a relationship between the number of keys generated and the key generation time, which appears to be linear after $H = 4$.

In the right plot in Figure 5.1, the relationship between the chain tree height, and the chain tree generation time is presented. For a chain tree, the number of associated keys is

[4] https://github.com/Breus/mbpqs
[5] https://github.com/bwesterb/go-xmssmt

equal to its height. The axes of the plot are in logarithmic scale to fit the measurement points in the graph. For the chain tree generation, there appears to be a linear correlation between the number of keys generated and the chain tree generation time after $h = 10$.

In both measurements, the times corresponding to the lowest values for H and h are higher than expected, and their points do not seem to fit in the further, apparently linear plot. The reason for this can be found in the set-up for the measurements. As mentioned before, the practical performance analysis is conducted on a CPU with 4 cores and 8 threads. In the MBPQS code, the generation of multiple keys and their leaves is equally divided over multiple threads, to use the full capacity of the CPU. Since for $H = 2$, 4 keys are generated, and for $h = 2$, only 2 keys are generated, in their corresponding measurements the full capacity of the CPU is not used yet, explaining the deviating results. Indeed, in single-threaded measurements, the plot was linear throughout the entire measurement domain.

The measurements for both the initial key generation times and chain tree generation times correspond to the complexity model. According to the model, the cost of the initial key generation is $2^H(vi) + (2^H - 1)(iv)\ hc$, indicating a linear relationship between the computational cost and the number of keys generated, logarithmic proportional in H. The tree chain generation was modeled to cost $h(vi) + (h-1)(iv)\ hc$, showing the linear relationship between h and the total computational cost.

Table 5.10: Key generation times for different values of w for $n = 32$.

	$w = 4$	$w = 16$	$w = 256$
Initial Key Generation			
$H = 8$	0.05 s	0.08 s	0.67 s
$H = 12$	0.72 s	1.30 s	10.86 s
$H = 16$	11.54 s	20.55 s	173.43 s
$H = 20$	187.39 s	331.31 s	2778.79 s
Tree Chain Generation			
$h = 2$	1.04 ms	1.95 ms	17.98 ms
$h = 10$	2.39 ms	4.00 ms	49.96 ms
$h = 100$	18.82 ms	33.56 ms	0.34 s
$h = 1000$	0.18 s	0.33 s	2.77 s
$h = 10000$	1.77 s	3.08 s	27.06 s

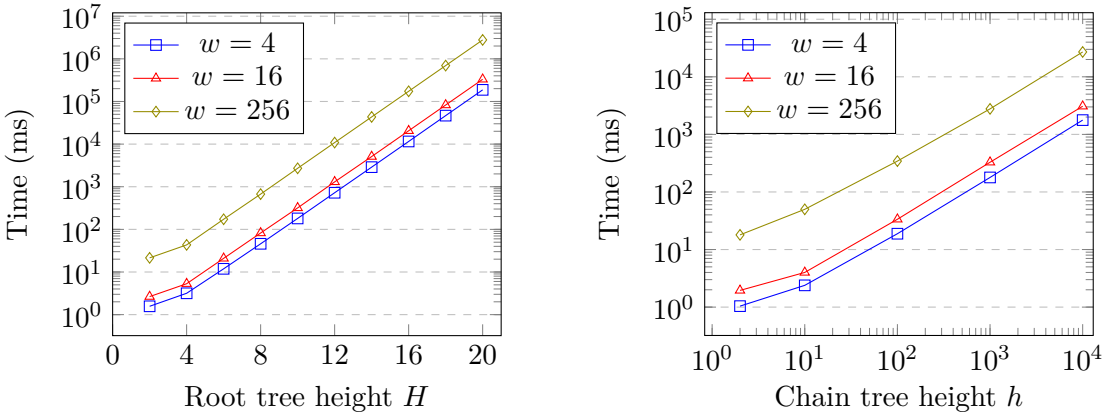

Figure 5.1: Left: Measured initial key generation times.
Right: Measured chain tree generation times.

5.4.2 Signing

In Table 5.11, the measured times to sign a 512KB message using MBPQS with SHA2-256 are presented. For both signing with caching disabled ($c = 0$) and enabled ($c = 1$), the results are presented for all implemented options of the Winternitz parameter. The ratios between different values of the Winternitz parameter are slightly smaller during signing with caching enabled than during key generation, discussed in Section 5.4.1. Instead of a factor 2 increase in timings between $w = 4$ and $w = 16$, and a factor 9 increase between $w = 16$ and $w = 256$, the increase in timings is about 1.6 and 6, respectively. This can be explained by the fact that for a single instance, half the steps of the WOTS-T chaining function are computed to create a WOTS-T signature. Therefore, the overhead of computations independent of the Winternitz parameter, such as pseudorandomly generating a WOTS-T seed, is larger than during key generation relatively speaking. During signing with caching disabled, the ratios are similar to those during key generation, because many full WOTS-T chains are computed to generate the authentication node of the signature.

Comparing the signing times for $c = 0$ with the tree generation times from Table 5.10, one can see that the majority of the computational cost of signing without caching stems from the generation of the authentication node. This corresponds to the complexity analysis, where the cost of signing without caching is modeled as (ix)+(i)+(vii)+h(vi)+$(h-2)$(iv) hc. Here, (iv) represents the cost of generating a tree leaf, which is the most expensive subroutine for signing without caching, with a cost more than twice as expensive as computing the WOTS-T signature (see (vii) in Table 5.9).

In Figure 5.2, the relationship between the signing time and the chain tree height h is shown for the results in Table 5.11. The solid lines represent the signing times when caching is enabled, and the dashed lines when caching is disabled. The deviation from the linear growth in the dashed lines for the first measurements is, again, due to the method of measuring, as explained in Section 5.4.1.

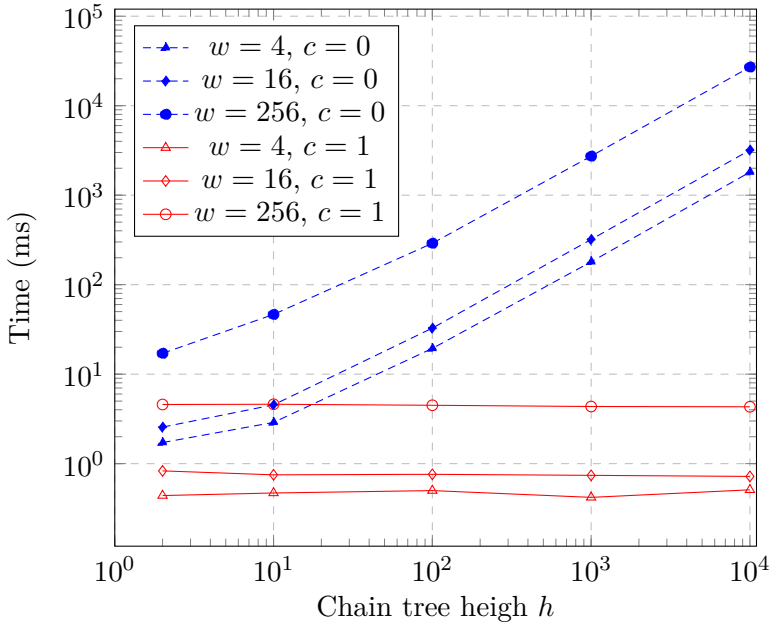

Figure 5.2: Measured signature generation time on a 512KB message using MBPQS.

Table 5.11: 512KB message signing times comparison for $n = 32$.

	$c = 0$			$c = 1$		
	$w = 4$	$w = 16$	$w = 256$	$w = 4$	$= 16$	$w = 256$
$h = 2$	1.72 ms	2.56 ms	17.08 ms	0.44 ms	0.83 ms	4.58 ms
$h = 10$	2.89 ms	4.54 ms	46.49 ms	0.47 ms	0.75 ms	4.61 ms
$h = 100$	19.38 ms	32.71 ms	0.29 s	0.50 ms	0.76 ms	4.49 ms
$h = 1000$	0.18 s	0.32 s	2.73 s	0.42 ms	0.74 ms	4.35 ms
$h = 10000$	1.81 s	3.19 s	27.02 s	0.51 ms	0.72 ms	4.32 ms

5.4.3 Verification

In Table 5.12, the verification times for each signature type are shown. For the message signature, two measurements are executed, verifying signatures on 32-byte and 512-kilobyte (KB) messages, respectively. In MBPQS, using SHA2-256, the root signature and growth signature sign 32 bytes of input, being the root nodes of chain trees. Therefore, a 32-byte message was verified to show the similarity in execution times between different verification algorithms.

Table 5.12: MBPQS-SHA2-256 verification times for all signatures types, for $H = 20$.

	$w = 4$	$w = 16$	$w = 256$
Root Signature	0.45 ms	0.64 ms	5.85 ms
Message Signature (32B)	0.44 ms	0.64 ms	5.77 ms
Message Signature (512KB)	1.83 ms	2.03 ms	7.21 ms
Growth signature	0.42 ms	0.66 ms	5.46 ms

Corresponding to the complexity analysis, the expected times for different signature types should be close to each other, for the same input length. Namely, the complexity of the message signature and growth signatures are the same for an equal input length, and the verification time of root signatures depends only marginally on the height of the root tree. Moreover, the computation of internal nodes is a relatively cheap operation, and only $(H-1)$ of such operations are added to verify a root signature compared to verifying signatures of the other two types. Indeed, the measurements in Table 5.12 show that the verification times for all signature types for the same input length are similar. A root tree height of $H = 20$ is used for the benchmarks to show just how marginal the extra operations in the root signature verification are compared to the cost to verify the WOTS-T signature. Since the verification times are independent from key caching, it is not considered in the results.

The time to verify a signature on a 512-KB message is added to the table to show the impact of the message length on the total verification time. For reference, the 512-KB message is 16,000 times the size of the 32-byte message. According to the complexity model, the extra message length of 512-KB messages, compared to the 32-byte messages, adds approximately 8,140 hash compression operations to the verification time, independent of the Winternitz parameter. Indeed, for all measurements, the additional verification time is approximately equal. For $w = 4$ and $w = 16$, approximately 76% and 68% of the total verification time is due to the initial hash message, respectively. This shows that, especially for low values of the Winternitz parameter, the cost to verify signatures on large messages is to a great extent due to the initial message hash.

5.4.4 Implementation Comparison

To put the measurements of the MBPQS implementation into perspective, the results are compared with results from measurements of XMSS-T implementations for similar parameters. The implementations used as benchmarks are the earlier mentioned XMSS-T implementation of Bas Westerbaan, and the Go implementation of XMSS-T from the Aidos Kuneen cryptocurrency team[6], which are referred to as BW\XMSS-T and AK\XMSS-T, respectively. The reference implementation of XMSS-T is not used as benchmark, because it is missing many optimizations, such as hash precomputations and multi-threading, and it is written in

[6] https://github.com/AidosKuneenOld/xmss

a different programming language.

In Table 5.13, the results are shown for each implementation. The 'MT' prefix in front of some XMSS-T names in the first column denotes that it concerns the multi-tree variant XMSS-T^{MT}. The XMSS-T^{MT} scheme names have a postfix, which should be interpreted as '-H/d', where H denotes the total height of the authentication structure, and d denotes the number of stacked XMSS tree layers.

In the comparison, the Winternitz parameter $w = 16$, a message length of 32 bytes, and SHA2-256 as underlying hash function were used in each implementation. These parameters are chosen because $w = 16$ and SHA2-256 are the default and required parameters for implementations of XMSS-T according to RFC8391. The message length of 32 bytes is chosen to be able to observe more subtle differences in the signing and verification algorithm between implementations. Namely, in Section 5.4.3 it was shown that the verification time of signatures on large messages is largely due to the initial message hash function. Therefore, comparing the different schemes for large messages would mainly be a comparison of the initial hash message implementation.

Because the algorithm sets of MBPQS and XMSS-T are not identical, and the considered implementations of XMSS-T use different optimizations and design, a fair comparison is challenging. Therefore, for each algorithm comparison, an explanation is given on the rationale behind the measurements.

Key Generation

In the second column of Table 5.13, the initial key generation times of the different implementations are listed. In the BW\XMSS-T implementation, only the XMSS trees directly required to sign messages are generated during the initial key generation. Hence, only a single (left-most) XMSS tree on each layer in the multi-tree variants is generated, and the entire authentication structure for single-tree variants. In the AK\XMSS-T implementation, only the top-most XMSS tree is generated during the initial key generation. The additional trees, required to sign messages, are generated during the first signing operation. This explains the shorter key generation times, but the larger signing times compared to BW\XMSS-T. For MBPQS, the more expensive requirement, generating all keys required for signing, is included in the key generation times. To have a fair comparison in key generation times, a root tree height of $H = 10$, and a chain tree height of $h = 1024$ is used for two measurements of MBPQS. This results in 2^{11} keys to be generated, corresponding to BW\XMSS-T-20/2, where the top-most tree, and left-most tree of the bottom layer are generated, both consisting of 2^{10} keys, for a total of 2^{11} keys.

Consequently, the initial key generation times of MBPQS-H10-h1024 and BW\XMSS-T-20/2 are almost identical, as expected, since the two implementations use identical optimizations. The key generation times in AK\XMSS-T-20/2 and AK\XMSS-T-40/4 are less than half of those times, mostly due to the aforementioned key generation scheduling decision. Besides, extra optimizations are implemented in AK\XMSS-T, such as multi-threading the

WOTS-T chaining function and fast XOR operations[7], using pointer arithmetic in Go.[8]

The key generation times of MBPQS are significantly shorter than both single-tree XMSS-T implementations, because for single-tree XMSS-T, all 2^{20} keys must be generated at once. By comparing the single-tree XMSS-T implementations, the additional optimizations of AK\XMSS-T compared to BW\XMSS-T results in a performance increase of about 16%.

Signing

To compare the signing times between the different implementations, in each implementation exactly 1,000 messages are signed, and the average timing for one signing operation is calculated from this. This number of messages was chosen to achieve a fairer comparison, for the following two reasons: Firstly, it decreases the penalty from generating the XMSS trees required for signing in the signing algorithm in AK\XMSS-T, instead of during key generation. Namely, the additional time for key generation during the signing algorithm is divided over 1,000 signature creations. Secondly, in the schemes MBPQS-h1024, BW\XMSS-T{-20/2, -40/4}, and AK\XMSS-T{-20/2, -40/4}, 1,000 is about[9] the maximal number of signatures that can be created before new keys need to be generated. Therefore, the time to generate keys is excluded in the measurements, resulting in a more genuine comparison of signing times.

The results of the measurements are shown in the third column of Table 5.13. The signing time of MBPQS-h10-c0 is approximately equal to the signing times in single-tree AK\XMSS-T, and twice as long compared to any variant of BW\XMSS-T. Furthermore, signing in MBPQS-h1024-c0 is approximately 150 times slower than BW\XMSS-T. The signing time of AK\XMSS-T-40/2 is significantly slower than the other implementations. This is because in the first signature generation, the bottom-layer XMSS tree is generated. The extra time for this is divided over only 1,000 signing operations, adding an approximately 279ms to the signing time. Finally, the results show that MBPQS with caching enabled outperforms all other schemes by a large amount in terms of signing times. It is about 3.5 times faster than all BW\XMSS-T variants, and about 8 times faster than the fastest AK\XMSS-T implementation.

Verification

In the fourth column of Table 5.13, the verification times of the different scheme implementations are shown. It is shown that the signature verification times in MBPQS are similar to the single-tree XMSS-T implementations. In the XMSS-T implementations, only a single WOTS-T signature is verified per signature. In MBPQS, additionally one WOTS-T signature must be verified for each h message signatures, assuming no chain tree growth. Therefore, to compute the total verification times for MBPQS variants, the verification time of a WOTS-T

[7] https://golang.org/src/crypto/cipher/xor_generic.go

[8] https://golang.org/pkg/unsafe/

[9] The actual maximum is $2^{10} = 1024$, but this is harder to use in quick conversions for comparison and makes a negligible difference for the timings.

Table 5.13: Algorithm execution times for different implementations of XMSS-T and MBPQS with $w = 16$, $n = 32$, and 32B messages.

Scheme	KeyGen	Sign	Verify
MBPQS			
SHA256-H10-h10-c0	0.32 s	4.54 ms	0.70 ms
SHA256-H10-h10-c1	0.32 s	0.59 ms	0.70 ms
SHA256-H10-h1024-c0	0.65 s	0.33 s	0.64 ms
SHA256-H10-h1024-c1	0.65 s	0.59 ms	0.64 ms
BWesterb\XMSS-T			
SHA256-20	336.28 s	2.19 ms	0.64 ms
MT-SHA256-20/2	0.66 s	2.27 ms	1.31 ms
MT-SHA256-40/2	672.31 s	1.94 ms	1.35 ms
MT-SH256-40/4	1.41 s	2.47 ms	2.63 ms
AidosKuneen\XMSS-T			
SHA256-20	281.03 s	4.61 ms	0.49 ms
MT-SHA256-20/2	0.27 s	6.58 ms	0.95 ms
MT-SHA256-40/2	278.78 s	291.94 ms	0.96 ms
MT-SHA256-40/4	0.28 s	7.78 ms	1.79 ms

signature is divided by h, and added to the message verification time. This is why the verification times for MBPQS with $h = 10$ are higher than those with $h = 1024$. For $h = 1024$, the verification time due to grow/root signature verification is 0.00064 ms, which is negligible. This explains why the signature verification times in MBPQS with $h = 1024$ and the single-tree BW\XMSS-T implementation are equal. The additional operations in XMSS-T to compute internal nodes to verify a signature are negligible, as shown in Section 5.4.3. The verification time in the single-tree AK\XMSS-T implementation is approximately 20% lower than the other two, due to the additional optimizations in this implementation, mentioned before.

To verify a signature in XMSS-T^{MT}, for each layer of XMSS trees, a WOTS-T signature is verified, which is clearly visible in Table 5.13. In MBPQS, signatures used to connect chain trees are only verified once, assuming a blockchain is used to keep track of which signatures are verified. This is why the signature verification times in MBPQS are lower compared to all multi-tree XMSS-T implementations.

Chapter 6

Discussion and Conclusion

In this chapter, the work from this thesis is first shortly reviewed, and then discussed. With the results from the previous chapter, the performance of the signature scheme proposed in Chapter 4 is appraised, and the requirements to implement this scheme are examined. The final topic of the discussion is the future work, where practical optimizations for the scheme, and suggestions to mitigate research limitations of this thesis, are being discussed. After the discussion, a conclusion is drawn.

6.1 Discussion

In this thesis, research has been conducted on post-quantum hash-based signatures (HBSs) for multi-chain blockchain technologies. The practical use of HBSs for blockchain has already been shown by others [88]. In addition, HBS schemes that leverage the blockchain structure, reducing the signature sizes compared to universally applicable HBS schemes, have been proposed [86, 23]. These existing proposals are mainly targeted at public, single-chain blockchains, while this research is focused on private or consortium multi-chain blockchain technology [92, 1]. To show the applicability of stateful HBSs for multi-chain blockchain, Multi-Blockchain Post-quantum Signatures (MBPQS), an HBS scheme based on BPQS is proposed in Chapter 4. MBPQS can be used to create post-quantum secure digital signatures on messages in multiple segregated blockchains using a single key pair.

6.1.1 Performance

In Chapter 5, performance analyses conducted on MBPQS are given. The results from these analyses, in combination with the use cases of MBPQS, will be discussed to appraise the performance of the scheme. As mentioned before, MBPQS is meant to be used in combination with a multi-chain blockchain. Typically, many signatures are stored in such a system, as each transaction is signed by one or multiple peers [92]. Furthermore, each peer maintains an instance of the blockchain;thus, each signature is stored multiple times. Increased signature sizes result in increased storage requirements for each peer. In addition, larger signatures sizes decrease the number of transactions that fit in a block, limiting the throughput of the blockchain network. Therefore, arguably one of the most important performance factors for schemes such as MBPQS is the size of the signatures.

Signature Size

In the key and signature sizes analysis, it is shown that the total signature size in MBPQS depends mostly on the height of the chain trees. Namely, a larger chain tree results in a lower average signature size for the same number of signatures in a key channel. To simplify the discussion, it is assumed for now that there is no chain tree growth, and thus the chain tree height is defined as h. In Section 5.2, it is shown that already for $h = 10$, 87% of the signature size can be attributed to a single OTS, and for $h = 100$, this is 95%. Since HBS schemes combine multiple OTSs, the size of a single OTS is considered the absolute minimum necessary size of a signature. Therefore, to significantly reduce the signature size in MBPQS, the size of the OTS must be reduced. To this end, a higher value of w could be used, but this results in longer key generation, signing, and verification times.

Key Generation

In the results from Section 5.4.4, it is shown that the total cost of key generation in MBPQS is virtually identical to XMSS-T when similar optimizations are implemented. However, for the key generation in the single-tree XMSS-T variant, all keys have to be generated at once, resulting in long initial key generation times. This issue is tackled in XMSS-T^{MT}, where

smaller individual XMSS trees are generated once they are required for signing. However, the reduced initial key generation times in XMSS-T^{MT} comes with a large penalty for the signature size, as shown in Section 5.2.4. Using MBPQS, the best of both worlds can be achieved: a small signature size, and a short initial key generation time. This is possible because the signature that connects chain trees in MBPQS is added only once every $h-1$ signatures, instead of in every signature like in XMSS-T^{MT}. Nevertheless, the approach in MBPQS has additional requirements for the implementation as a consequence, which will be discussed later.

Signing and Verification

As shown in Section 5.4.4, the signing times in MBPQS are about 4 times lower compared to XMSS-T with caching enabled in both schemes. Even without caching, MBPQS can be used to sign hundreds of signatures per second when a chain tree with less than a height of 15 is used. Nonetheless, considering the use cases of MBPQS, the verification times are more important. Namely, a signature created by a single peer is verified by all peers in the blockchain. Furthermore, when a peer joins a blockchain, it needs to verify all the signatures before it can participate. The average verification times in MBPQS do not depend on the cache, but on the height of the chain tree. For each signature, a single OTS signature is verified, but once every $h-1$ signatures, an additional one must be verified, again assuming a growth factor of 0. Even for the minimal value $h = 2$, verifying a signature in MBPQS is at least as fast as in the fastest XMSS-T^{MT} variants. For higher values of h, the verification time converges to that of a single WOTS-T signature verification. In [23], it was already shown that verifying a WOTS-T signature is either faster or as fast as verification in classical signature schemes used in blockchain, such as ECDSA, EdDSA, and RSA.

6.1.2 Usability

As mentioned before, to implement MBPQS in blockchain, additional requirements must be met, besides the need to keep state. First of all, signatures belonging to the same key channel can only be verified in a chronological order. Namely, each signature verification updates the anchor node for the corresponding key channel. Subsequently, this updated anchor node is used to verify the next signature in the key channel. Secondly, there can be no missing chain tree nodes (links) in the key verification chain. If previous signatures belonging to a key channel are not (yet) included in the blockchain, the signer must include the missing links in the signature, up to the anchor node in the key channel. In Section 4.1, it was shown that this only adds n bytes per missing link to a signature.

Since MBPQS is meant to be used in private or consortium multi-chain blockchains, there are two general use cases for which the scheme can be used. First of all, signing of transactions by participants, and secondly, signing of blocks by block writers.

Block Signing

According to the definition of blockchain in Section 2.2.1, blocks are created in an ordered sequence. Furthermore, to verify the validity of the blockchain, no intermediate blocks can be missing. These requirements correspond closely with the aforementioned implementation requirements for MBPQS. When multiple block writers are active in the same blockchain channel, a reference to the previous block created by the same orderer should be included.

Transaction Signing

For transaction signing, MBPQS can be used as well, but adhering to the aforementioned requirements is less trivial. The chronological ordering of transactions is required in a blockchain for the synchronization of the ledger state among the peers. Namely, if transactions would be applied to the blockchain in a different sequence by peers, this can result in contradicting states of the blockchain. Therefore, transactions are ordered to prevent this from happening, making the first requirement fairly easy to adhere to during implementation.

The second requirement, stating that no missing links can occur in the verification chains might be more difficult to adhere to. For example, consider a signed transaction which is sent to the block writer, but is not added in a block for any reason. Since keys can only be used once, the signer cannot use that key again, but its verification key needs to be included in the blockchain to verify the consecutive signatures. The signer should be notified, and include the missing link in the next signature it creates. Inserting a missing link is indeed cheap in MBPQS, but an extra mechanism might be required in the system to adhere to this requirement. Especially for systems with an extensive transaction flow mechanism, it might be non-trivial to implement this.

6.1.3 Future Work

Practical Optimizations

In the current MBPQS implementation, for each signature, the entire chain tree is generated to compute the authentication node. However, on average, only half of the chain tree needs to be generated. Considering that most of the signing time for MBPQS without caching is due to the chain tree generation, the number of signatures generated per second can almost be doubled using this improvement.

Another optimization for MBPQS is multi-threading the WOTS-T chaining function. As shown by others [86], this optimization cuts the execution time of the WOTS-T chaining function approximately in half for $w > 16$, using more than 4 CPU threads. In the complexity analysis in Section 5.3, it is shown that the vast majority of the computational cost for key generation, verification, and signing in MBPQS is due to the WOTS-T chaining function. Therefore, this optimization could partly mitigate the increased algorithm execution times caused by increasing w for smaller signatures. Nonetheless, for the key generation, multi-threading the chaining function will result most likely only in a minor speed-up, as the generation of multiple keys is already evenly divided over the available cores.

Research Limitations

In our research, the practicality of the scheme for multi-chain blockchains is only tested for the data structure of a multi-chain blockchain. In practice, there might be additional hurdles to overcome to implement MBPQS, depending on the system requirements. Nonetheless, because multi-chain blockchain frameworks are vastly different from each other, no one-size-fits-all solution can be implemented. Therefore, requirements are provided in this work to which must be adhered to implement MBPQS.

A more practical limitation of our research is that only two caching strategies are implemented in MBPQS. The first strategy is the most storage-efficient one, where the chain tree is recomputed for every signing operation. This method results in large signing times if large chain trees are used. The second strategy is the fastest alternative for signing, where all internal nodes in a chain tree are cached. This method results in fast signing for all chain tree heights, but for each key channel, $h \times n$ bytes must be cached. Besides these two obvious caching strategies, other approaches exist for unbalanced hash trees, as shown in [82]. Implementing different caching strategies adds more flexibility to MBPQS. For instance, a caching strategy that is a compromise between signing speed and cache size might be a good addition to the scheme.

As mentioned in Section 4.1, the internal structures of MBPQS are based on XMSS-T for their strong security properties. Nonetheless, a scheme similar to MBPQS could be created using the constructions from LMS. As discussed in Section 3.1.8, the algorithms in LMS are 3 to 5 faster compared to XMSS-T, but the security of the scheme relies on the assumption that the Merkle-Damgård structure behaves (pseudo)randomly. Building a MBPQS variant based on LMS could be useful for systems where algorithm speed is more important than the reliance on weaker security assumptions. Furthermore, with faster algorithms, a higher value of w can be used, resulting in smaller signatures for algorithm execution times similar to the current implementation based on XMSS-T

Growth Factor

As shown in Chapter 5, increasing the chain tree height in MBPQS reduces the total number of growth signatures per key channel. A smaller number of growth signatures results in smaller average signatures sizes and less algorithm operations, as these signatures can be considered overhead. However, larger chain trees also result in longer signing times or larger caches, depending on whether caching is enabled.

Consider a multi-chain blockchain, using MBPQS with caching enabled, where the user rarely signs messages in some channels, while his signing rate in other channels is high. Optimally, the caches of the channels where the user barely signs messages are small, while the other caches are larger. In this way, the available cache space is optimally used to prevent as much overhead from growth signatures as possible.

A trivial solution to achieve this would be to let the user determine the chain tree height per channel, but this has two major disadvantages. First of all, it might be hard to estimate the rate in which messages will eventually be signed in a channel. Therefore, the estimated value might be (much) too high, resulting in an unnecessarily large cache, and redundant

initial key generations. Alternatively, when the estimated value is too low, an unnecessary number of growth signatures is added. Secondly, having to specify a chain tree height for each channel adds extra complexity to the usability of the scheme.

Instead, MBPQS implements an optional growth factor parameter, which is used to dynamically determine the chain tree height in a key channel. However, for MBPQS without caching, using the growth factor results in signing times that gradually increase for each consecutive chain tree, which is considered impractical. For MBPQS with caching however, the growth factor tackles both problems of the aforementioned solution, where the user had to determine the chain tree height for each key channel. With the growth factor, caches grow automatically according to the signing rate in a channel, thus the user does not have to specify a chain tree height for each channel. Furthermore, the growth factor makes estimation of the number of signatures per channel easier, as a conservative low value can be chosen as initial chain tree height, while the growth factor enlarges the caches of channels with high signing rates automatically in a faster rate than other channels.

However, during the analyses for Chapter 5, two major disadvantages of the growth factor where discovered. First of all, the channel caches will grow indefinitely, taking a lot of storage space, while at some point, the extra cache size barely provides any additional benefits. Secondly, the approach bears the implicit assumption that channels where the signing rate used to be high, will also be high in the future as well. Even though this might be likely, this does not have to be the case. To solve the first problem, two approaches for mitigation are proposed. Firstly, the maximum chain tree height could be determined either by using a different growth function, which converges to a limit, or this maximum can be specified with an additional parameter. Secondly, a caching strategy such as first-in-first-out could be used, in combination with a fixed cache size.

6.2 Conclusion

In this thesis, the post-quantum secure stateful HBS scheme MBPQS is presented. The scheme is designed to sign messages in a multi-chain blockchain under a single public key. More specifically, the design is focused on private and consortium multi-chain blockchains in which blocks are signed by the block writer, and no forks can happen. The scheme is based on BPQS, using the internal structures from XMSS-T. The security of the scheme is not based on any number-theoretic or structured hardness assumptions, but on the security properties of hash functions, which are well-understood both in the classical and the quantum model.

When used in combination with a blockchain, MBPQS provides smaller signatures and better performance compared to universally applicable HBS schemes with similar security properties. Furthermore, MBPQS can be used to sign virtually unlimited messages, and supports distributed key generation. When using MBPQS without a cache, the signing times are comparable to XMSS-T with a cache, for comparable signature sizes. The verification times for MBPQS with chain trees larger than 100 are equal to the verification times in single-tree XMSS-T. For smaller chain trees, the verification time is, even in the worst-case, equal to the best performing multi-tree XMSS-T variants.

The main drawbacks of MBPQS compared to non-quantum secure signature schemes, currently used in blockchain, are the larger signature sizes and the need to keep state of the private key. However, the large signatures are the main drawback of stateful HBSs in general, even for OTS schemes. Compared to universally applicable stateful HBS signature schemes, MBPQS adds the requirements that signatures in a blockchain can only be verified in a chronological order, and that missing links in the key hash chain must be added to subsequent signatures. In the systems targeted by MBPQS, digital signatures are used to sign blocks and transactions. To sign blocks, these requirements are easily met as they correspond with the nature of the blockchain. However, to use MBPQS to sign transactions, implementing the scheme in existing blockchain frameworks might be more challenging as a mechanism needs to be in place such that peers insert the missing links.

Bibliography

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 30:1–30:15, New York, NY, USA, 2018. ACM. 13, 25, 77

[2] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013. 38

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. 12

[4] Jean-Philippe Aumasson, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe. SPHINCS+ – Submission to the NIST post-quantum project, April 2019. 38

[5] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 431–448, 1999. 47

[6] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making uowhfs practical. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 470–484, 1997. 15

[7] Daniel J. Bernstein. Post-quantum cryptography. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 949–950. Springer, 2011. 12, 22

[8] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In *Cryptographic Hardware and Embedded Systems -*

CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings, pages 124–142. Springer, 2011. 11

[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. 12, 37

[10] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017. 12

[11] Daniel Bleichenbacher and Ueli M. Maurer. Directed acyclic graphs, one-way functions and digital signatures. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, pages 75–82, 1994. 20

[12] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 41–69, 2011. 55

[13] Johannes A. Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the Security of the Winternitz One-Time Signature Scheme. In *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, pages 363–378, 2011. 20, 35

[14] Johannes A. 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. 35, 36, 37, 45, 55

[15] Johannes A. Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle Signatures with Virtually Unlimited Signature Capacity. In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, pages 31–45, 2007. 34

[16] Johannes A. Buchmann, Erik Dahmen, and Michael Szydlo. Hash-based digital signature schemes. In *Post-Quantum Cryptography*, pages 35–93. Springer, 2009. 12

[17] Johannes A. Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. CMSS - an improved merkle signature scheme. In *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings*, pages 349–363, 2006. 34, 36

[18] Johannes A. Buchmann, Luis Carlos Coronado García, Martin Döring, Daniela Engelbert, Christoph Ludwig, Raphael Overbeck, Arthur Schmidt, Ulrich Vollmer, and Ralf-Philipp Weinmann. Post-quantum signatures. *IACR Cryptology ePrint Archive*, 2004:297, 2004. 12

[19] Ahto Buldas, Denis Firsov, Risto Laanoja, Henri Lakk, and Ahto Truu. A New Approach to Constructing Digital Signature Schemes (Extended Paper). *IACR Cryptology ePrint Archive*, 2019:673, 2019. 41, 43

[20] Ahto Buldas, Risto Laanoja, and Ahto Truu. A Server-Assisted Hash-Based Signature Scheme. In *Secure IT Systems*, pages 3–17. Springer International Publishing, 2017. 40, 43

[21] Ahto Buldas, Risto Laanoja, and Ahto Truu. A blockchain-assisted hash-based signature scheme. In *Secure IT Systems - 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018, Proceedings*, pages 138–153, 2018. 40, 41, 43

[22] Feng Ca, Haojin Lv, Zhanfeng Ma, Kai Zhen, Zhihong Chong, Zhenjie Zhang, and Ruosong Xu. The first native mutli-chain system supporting evm in the world. making large-scale blockchain applications possible. Position paper, PCHAIN, 2019. https://www.pchain.org/js/generic/web/viewer.html Accessed 28 June 2019. 13, 25

[23] Konstantinos Chalkias, James Brown, Mike Hearn, Tommy Lillehagen, Igor Nitto, and Thomas Schroeter. Blockchained post-quantum signatures. In *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData 2018, Halifax, NS, Canada, July 30 - August 3, 2018*, pages 1196–1203, 2018. 13, 39, 43, 77, 78

[24] N. Chaudhry and M. M. Yousaf. Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities. In *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, pages 54–63, Dec 2018. 25

[25] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016. 12

[26] Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, pages 109–123, 2008. 35

[27] Dipankar Dasgupta, John M. Shrein, and Kishor Datta Gupta. A survey of blockchain from security perspective. *Journal of Banking and Financial Technology*, 3(1):1–17, Apr 2019. 11

[28] Ana Karina D. S. de Oliveira, Julio López, and Roberto Cabral. High performance of hash-based signature schemes. *International Journal of Advanced Computer Science and Applications*, 8(3), 2017. 37

[29] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, January 4-7, 2016*, pages 13:1–13:10, 2016. 25

[30] Dorothy E. Denning. Digital signatures with RSA and other public-key cryptosystems. *Commun. ACM*, 27(4):388–392, 1984. 11

[31] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976. 20

[32] Chris Dods, Nigel P. Smart, and Martijn Stam. Hash based digital signature schemes. In *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, pages 96–115, 2005. 21

[33] Edward Eaton. Leighton-micali hash-based signatures in the quantum random-oracle model. In *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, pages 263–280, 2017. 37

[34] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. In *Conference on the Theory and Application of Cryptology*, pages 263–275. Springer, 1989. 21

[35] Aleksey Fedorov, Evgeny O. Kiktenko, and Alexander I. Lvovsky. Quantum computers put blockchain security at risk. *Nature*, 563:465–467, November 2018. 12

[36] Scott R. Fluhrer. Further analysis of a proposed hash-based signature standard. *IACR Cryptology ePrint Archive*, 2017:553, 2017. 37

[37] World Economic Forum. Deep Shift Technology Tipping Points and Societal Impact. page 24. Global Agenda Council on the Future of Software & Society, September 2015. 11

[38] LC Coronado Garcıa. On the security and the efficiency of the merkle signature scheme. Technical report, Technical Report 2005/192, Cryptology ePrint Archive, 2005. 34

[39] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. 18

[40] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. 35

[41] Gideon Greenspan. Multichain private blockchain. Whitepaper, Coin Sciences Ltd., November 2014. https://www.multichain.com/download/MultiChain-White-Paper.pdf Accessed 2 August 2019. 25

[42] Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 41–59, 2006. 15

[43] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. 35

[44] Mike Hearn. Corda: A distributed ledger. Whitepaper 0.5, R3, November 2016. https://docs.corda.net/releases/release-V3.1/_static/corda-technical-whitepaper.pdf Accessed 27 June 2019. 13

[45] Jordi Herrera-Joancomartí and Cristina Pérez-Solà. Privacy in Bitcoin Transactions: New Challenges from Blockchain Scalability Solutions. In *Modeling Decisions for Artificial Intelligence - 13th International Conference, MDAI 2016, Sant Julià de Lòria, Andorra, September 19-21, 2016. Proceedings*, pages 26–44, 2016. 25

[46] Jeremy Hsu. CES 2018: Intel's 49-Qubit Chip Shoots for Quantum Supremacy. https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy, January 2018. Accessed 18 June 2019. 12

[47] Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, pages 173–188, 2013. 12, 21, 35

[48] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: extended merkle signature scheme. RFC 8391, May 2018. 22, 25, 26, 35, 36, 37, 43, 45, 55, 59, 60, 64

[49] Andreas Hülsing, Lea Rausch, and Johannes A. Buchmann. Optimal Parameters for XMSS-MT. In *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*, pages 194–208, 2013. 36, 55

[50] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, pages 387–416, 2016. 15, 16, 17, 29, 35, 36, 37, 38, 43, 55

[51] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, pages 387–416, 2016. 20, 21, 37

[52] Tzonelih Hwang, Yi-Ping Luo, Prosanta Gope, and Zhi-Rou Liu. Forward/backward unforgeable digital signature scheme using symmetric-key crypto-system. In *2016 International Computer Symposium (ICS)*, pages 244–247. IEEE, 2016. 18

[53] Shuxian Jiang, Keith A Britt, Alexander J McCaskey, Travis S Humble, and Sabre Kais. Quantum annealing for prime factorization. *Nature*, 8(1):17667, 2018. 12

[54] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Sec.*, 1(1):36–63, 2001. 11

[55] Panos Kampanakis and Scott R. Fluhrer. LMS vs XMSS: A comparison of the stateful hash-based signature proposed standards. *IACR Cryptology ePrint Archive*, 2017:349, 2017. 37, 43, 63, 64, 65

[56] Jonathan Katz. *Digital Signatures*. Springer, May 2010. 17, 18

[57] Jonathan Katz. Analysis of a proposed hash-based signature standard. In *Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings*, pages 261–273, 2016. 37

[58] Julian Kelly. A Preview of Bristlecone, Google's New Quantum Processor. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, March 2018. Accessed 18 June 2019. 12

[59] Evgeny O. Kikento, N. O. Pozhar, M. N. Anufriev, A. S. Trushechkin, R. R. Yunusov, Y. V. Kurochkin, A. I. Lvovsky, and A. K. Fedorov. Quantum-secured blockchain. *CoRR*, abs/1705.09258, 2017. 11

[60] Will Knight. IBM Raises the Bar with a 50-Qubit Quantum Computer. https://www.technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/, November 2017. Accessed 17 June 2019. 11

[61] Tommy Koens and Erik Poll. *The Drivers Behind Blockchain Adoption: The Rationality of Irrational Choices: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers*, pages 535–546. January 2019. 11

[62] Philip Lafrance and Alfred Menezes. On the security of the WOTS-PRF signature scheme. *Adv. in Math. of Comm.*, 13(1):185–193, 2019. 35

[63] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979. 20, 33

[64] Frank T. Leighton and Silvio Micali. Large provably fast and secure digital signature schemes based on secure hash functions, U.S. Patent 5432852A, Sept. 1993. https://patents.google.com/patent/US5432852 Accessed 25 July 2019. 20, 33, 36, 37

[65] Tal Malkin, Daniele Micciancio, and Sara K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 400–417, 2002. 43, 55

[66] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. The impact of quantum computing on present cryptography. *CoRR*, abs/1804.00200, 2018. 11

[67] David McGrew and Michael Curcio. Hash-Based Signatures. Internet-Draft draft-mcgrew-hash-sigs-04, Internet Engineering Task Force, March 2016. Work in Progress. 33, 37

[68] David McGrew, Michael Curcio, and Scott Fluhrer. Leighton-Micali Hash-Based Signatures. RFC 8554, April 2019. 37, 59

[69] Ralph Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1979. 21, 33, 37

[70] Ralph Merkle. Secrecy, authentication, and public key systems. *PhD Thesis Department Electrical Engineering, Stanford University*, 1979. 64

[71] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008. 24, 25

[72] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989. 38

[73] National Institute of Standards and Technology. *FIPS PUB 180 4: Secure Hash Standard*. pub-NIST, August 2015. Supersedes FIPS PUB 180-4 2012 May 03. 64, 65

[74] Qutech online news article. Microsoft and TU Delft collaboration started. https://qutech.nl/microsoft-and-tu-delft-collaboration-started/, June 2019. Accessed 19 June 2019. 12

[75] Edwin Pednault, John Gunnels, Dimitri Maslov, and Jay Gambetta. On "quantum supremacy". https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/, October 2019. Accessed October 30 2019. 12

[76] Xinhua Peng, Zeyang Liao, Nanyang Xu, Gan Qin, Xianyi Zhou, Dieter Suter, and Jiangfeng Du. Quantum Adiabatic Algorithm for Factorization and Its Experimental Implementation. *Phys. Rev. Lett.*, 101:220405, Nov 2008. 12

[77] Adrian Perrig. The biba one-time signature and broadcast authentication protocol. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 28–37, 2001. 20

[78] John Proos and Christof Zalka. Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves. volume 3, pages 317–344, Paramus, NJ, July 2003. Rinton Press, Incorporated. 11

[79] Leonid Reyzin and Natan Reyzin. Better than biba: Short one-time signatures with fast signing and verifying. In *Information Security and Privacy, 7th Australian Conference, ACISP 2002, Melbourne, Australia, July 3-5, 2002, Proceedings*, pages 144–153, 2002. 20

[80] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, pages 371–388, 2004. 15, 35

[81] 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. 35

[82] Berry Schoenmakers. Explicit optimal binary pebbling for one-way hash chain reversal. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, pages 299–320, 2016. 80

[83] Marc Stevens. Fast collision attack on MD5. *IACR Cryptology ePrint Archive*, 2006:104, 2006. 34

[84] Michael Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 541–554, 2004. 34

[85] NIST Online News Updates. Request for Public Comments on Stateful Hash-Based Signatures (HBS) . February 4 2019. https://www.nist.gov/news-events/news/2019/02/request-public-comments-stateful-hash-based-signatures-hbs Accessed 25 July 2019. 35, 37, 59

[86] Wouter van der Linde. Post-quantum blockchain using one-time signature chains. Master's thesis, Radboud University, August 2018. 13, 38, 39, 43, 64, 77, 79

[87] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 17–36, 2005. 34

[88] Peter Waterland. Quantum Resistant Ledger (QRL). Whitepaper, The QRL Foundation, October 2016. https://github.com/theQRL/Whitepaper/blob/master/QRL_whitepaper.pdf Accessed 16 June 2019. 13, 33, 43, 77

[89] Tao Xie, Fanbao Liu, and Dengguo Feng. Fast collision attack on MD5. *IACR Cryptology ePrint Archive*, 2013:170, 2013. 34

[90] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. In *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, pages 163–181, 2017. 12

[91] Hao Zhang, Chun-Xiao Liu, and Sasa Gazibegovic et al. Quantized Majorana conductance. *Nature*, 556(7699):74–79, April 2018. 12

[92] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *IEEE International Congress on Big Data, BigData Congress, Honolulu, HI, USA, June 25-30, 2017*, pages 557–564, June 2017. 11, 13, 24, 25, 38, 77

[93] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: a survey. *International Journal of Web and Grid Services*, 14(4):352–375, October 2018. 11