Digital Signature Schemes Based on Hash Functions

by

Philip Lafrance

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Cryptographers and security experts around the world have been awakened to the reality that one day (potentially soon) large-scale quantum computers may be available. Most of the public-key cryptosystems employed today on the Internet, in both software and in hardware, are based on number-theoretic problems which are thought to be intractable on a classical (non-quantum) computer and hence are considered secure. The most popular such examples are the RSA encryption and signature schemes, and the Elliptic Curve Diffie-Hellman (ECDH) key-exchange protocol employed widely in the SSL/TLS protocols. However, these schemes offer essentially zero security against an adversary in possession of a large-scale quantum computer. Thus, there is an urgent need to develop, analyze and implement cryptosystems and algorithms that are secure against such adversaries. It is widely believed that cryptographic hash functions are naturally resilient to attacks by a quantum adversary, and thus, signature schemes have been developed whose security relies on this belief.

The goal of this thesis is to give an overview of hash-based cryptography. We describe the most important hash-based signature schemes as well as the schemes and protocols used as subroutines within them. We give a juxtaposition between stateful and stateless signature schemes, discussing the pros and cons of both while including detailed examples. Furthermore, we detail serious flaws in the security proof for the WOTS-PRF signature scheme. This scheme had the feature that its security proof was based on minimal security assumptions, namely the pseudorandomness of the underlying function family. We explore how this flawed security argument affects the other signature schemes that utilize WOTS-PRF.

Acknowledgments

First and foremost I would like to thank my supervisor Dr. Alfred Menezes for his world-class guidance, tutelage, and conversations over the course of these past two years. Alfred's unique insights and razor sharp eyes were invaluable to my growth and success throughout this program. His experience with the so-called "real-world" has kept me grounded and on a path that I am not only overjoyed to be walking, but am proud to be on.

I would like to thank the department of Combinatorics and Optimization for all it has provided me during my time here. The administrative staff have been nothing but friendly and helpful. In particular, I would like to thank Melissa Cambridge for everything she has done for not only myself, but also for all my peers and colleagues. Melissa was always available to give me advice or to lend a hand with any number of problems. There is no doubt in my mind that this whole ordeal would have been much more grueling if it were not for Melissa.

I am grateful for my readers, Dr. Mosca and Dr. Jao for their helpful insights and diligence. Furthermore, I want to sincerely thank CryptoWorks21 for their financial support and for the valuable training sessions they continue to provide. Additionally, my thanks and appreciation extend to NSERC, not only for their financial support, but also because of all they do for academia.

Finally, I wish to thank my dear friends that have made this all possible. I fear that no words could truly convey how much good they have brought into my life over these years. Just by knowing these people I have become a wiser, happier, and better person. From the bottom of my heart, thank you my friends.

Dedication

This thesis is dedicated to the person who has loved me, supported me, and believed in me more than any other. This is for my Mother. I can only hope to one day pay her back for all she has done for me.

Table of Contents

List of Figures

Chapter 1

Introduction

The field of Cryptography has experienced dramatic changes since its infancy. From the primitive encryption schemes of the Romans, to the paradigm changing advent of public-key cryptography and so very much in between, the search for secure and efficient methods of protecting our data has been ongoing. However, security experts worldwide are acutely aware that we may be approaching another paradigm shift in the field.

We are entering the Quantum Era. Researchers around the world are consistently working to develop functional quantum computers; computers which are capable of calculations — which although are not theoretically impossible for our classical computers, would take infeasible amounts of time and memory to complete — that are devastating to modern cryptographic infrastructures. Previously secure systems such as the famed RSA encryption scheme, the Elliptic Curve Digital Signature Algorithm, or even the Diffie-Hellman key agreement scheme all provide essentially zero security against an adversary equiped with a large-scale quantum computer.

It is to this end that the National Institute of Standards and Technology (NIST) announced their challenge to produce new quantum safe cryptographic standards [33]. Of particular interest are quantum safe encryption schemes, signature schemes and key agreement protocols. This thesis focuses on the second item. In particular, we focus on what seems to be the most likely candidate for standardization in this regard: hash-based digital signature schemes. That is, signature schemes (a notion we introduce in Section 2.2) based on cryptographic hash functions (Section 2.1).

NIST is clear that the intent of their challenge is not to pick one clear, ultimate winner in each of the above mentioned categories, but rather to select multiple secure schemes and protocols as each individual scheme is likely to have its own strengths and weaknesses.

So, even though we believe that hash-based signature schemes will "win" the challenge, it remains to discuss *which* hash-based schemes may be selected. Hence, the purpose of this work is to collect and describe all of the potential candidates and describe in detail their strengths and weaknesses. Moreover, we collect and discuss the techniques used in the field of hash-based cryptography, why these techniques are used and what they actually accomplish. We analyze the obstacles to practicality that hash-based schemes might suffer from and present the best known remedies to these issues. Furthermore, we discuss the security of these schemes and their key sizes, key generation times and signing/verification times.

The schemes we discuss in this work are of the following two types. The first type are one-time signature schemes. These are schemes whose security is only guaranteed if at most one message is signed with any particular key pair. The second type are full signature schemes based on binary hash trees which utilize the schemes of type one.

Organization: Chapter 2 of this work introduces the requisite background information to understand the later chapters. We introduce hash functions and discuss their desired properties and presumed security against classical and quantum adversaries. We introduce a basic hash-based signature scheme which we use as a starting point for our discussion of the topics mentioned above. Chapter 3 introduces each of the one-time signature schemes which are of interest in and of themselves, but which in some sense serve as building blocks for the more complete schemes discussed later on. We present the flaws that we discovered in the security proof and analysis of the WOTS-PRF signature scheme from [5, 6]. In Chapter 4, we discuss a variety of full signature schemes based on binary hash trees which implement the schemes from Chapter 3. This includes both stateful and stateless schemes. Finally, in the last chapter, Chapter 5 we conclude with a summary of what we have accomplished and outline potential projects for future work.

Notation: Much notation will be introduced in later chapters, but here we mention some of the more standard notation which we shall be employing. We will write $x \xleftarrow{\$} X$ to denote that the value x is selected uniformly at random from the set X. Also we will write $\log(x)$ to represent the logarithm base-2 of x, i.e., $\log(x) = \log_2(x)$.

Chapter 2

Background

This chapter aims to give the reader a gentle introduction to hash-based cryptography. In Section 2.1 we introduce hash functions and the security properties which are required of them as well as the fastest known generic classical and quantum attacks on them. Then, in Section 2.2 we concretely define digital signature schemes, and in particular, what a hash-based digital signature scheme is. Section 2.3 introduces a fundamental hash-based signature scheme and systematically describes its drawbacks and provides modifications that resolve these issues in a clear, rigorous, but not-too-technical way.

2.1 Hash Functions

There exists some debate amongst cryptographers as to the precise definition of a hash-function. Some will claim that because security properties are not required in the definition that simply saying "a hash function is just a function" is satisfactory. Others may argue that more substance is necessary. The definition we give below is one which is suitable for our needs and is seemingly the most common definition given.

Definition 2.1.1. *Let n be a fixed, positive integer. A hash function g is a deterministic function mapping elements of $\{0,1\}^*$ to elements of $\{0,1\}^n$. That is,*

$$g : \{0,1\}^* \to \{0,1\}^n.$$

Strictly speaking we do not need the domain of g to be all bitstrings, but this is the most general form. Most often the domain will either be $\{0,1\}^*$ or $\{0,1\}^n$. Again,

it is not strictly necessary for the definition, but in practice we also need that $g(x)$ is efficiently computable for each x in the domain; we will implicitly assume this from now on. Additionally, if we are working with an arbitrary hash function we will also assume that it is uniformly random, but deterministic.

Next, we describe the most well-known and desired properties of hash functions as well as the fastest known generic classical and quantum attacks on those properties. We remark that we will always assume that any attacker we discuss in this work is computationally bounded. That is to say we assume every attacker has as an upper bound on their computational abilities some polynomial in the security parameter, i.e., they cannot perform more than polynomially many operations. The security parameter for the hash functions we discuss is a positive integer n.

2.1.1 Preimage Resistance

Definition 2.1.2. *A hash function $g : \{0,1\}^* \to \{0,1\}^n$ is said to be* preimage resistant *if given a random element $y \xleftarrow{\$} \{0,1\}^n$, it is computationally infeasible to produce an element $x \in \{0,1\}^*$ such that $g(x) = y$.*

In other words, if one is given a random image under the hash function, one cannot feasibly produce a value in the domain which yields that image. Sometimes this property is also called *one-wayness*, and g is referred to as a *one-way function*.

The fastest classical attack on preimage resistance against a generic hash function is simply brute force. Suppose that an attacker is given a uniformly random $y \in \{0,1\}^n$, and is challenged to produce a preimage of y. Assuming of course that g is random (but deterministic) the best that the attacker can do is to select arbitrary elements from the domain and check if the hash of any of these values equals the challenge y. If the attacker selects an element $x \in \{0,1\}^*$, there is only a 2^{-n} probability that x is a preimage of y. Hence, the expected number of attempts before the attacker successfully finds a preimage is 2^n. We say that the hash function has n-bits of security against this attack.

It is necessary to also consider the security of hash functions against a quantum adversary, i.e., an attacker with access to a quantum computer. Unfortunately, the best known generic quantum attack against preimage resistance is a little better than brute force. To see why this is so, we give a high level exposition of Grover's search algorithm [16]. Let n be a positive integer, and suppose that there is a function $h : \{0,1\}^n \to \{0,1\}$ such that $h(x) = 1$ for exactly one $x \in \{0,1\}^n$ and $h(x) = 0$ for all other inputs. We also require that h is efficiently computable. Then, with high probability, Grover's search algorithm returns

the value x such that $h(x) = 1$ in $2^{n/2}$ operations. Generally, Grover's algorithm is thought of as a probabilistic method to search an unsorted database, but we will think of it as a way to invert functions. More generally, if there exist p distinct values in the domain that have image 1, then Grover's algorithm needs only $2^{n/2}p^{-1/2}$ operations to produce one of them.

Now let's see how this algorithm can be used to attack preimage resistance. Let g be as above, and let $y \in \{0,1\}^n$ be randomly selected. Define a function $h_y : \{0,1\}^n \to \{0,1\}$ by

$$h_y(x) = \begin{cases} 0 & \text{if } g(x) \neq y \\ 1 & \text{if } g(x) = y. \end{cases}$$

As we assumed that g is efficiently computable, it is clear that so is h_y. However, observe that g has as its domain all non-empty bitstrings, and that Grover's algorithm only works with binary functions of fixed input length; of course $\{0,1\}^n$ is only a small subset of all non-empty bitstrings. And so, if we apply Grover's algorithm to h_y, if we get an answer, it will be an n-bit string. One may think that is is not satisfactory; however, if we assume that g is uniformly random then there is a high probability that a preimage of y does indeed exist in $\{0,1\}^n$. If this is the case, then if we apply Grover's algorithm to h_y, we will obtain a value x such that $g(x) = y$ in $O(2^{n/2})$ operations. Note that because the algorithm is not guaranteed to succeed, one may have to run it more than once, and we might choose to modify h_y to act on $\{0,1\}^{n+1}$ instead.

Hence, to obtain say 128-bit security for preimage resistance against a quantum adversary, we need only extend the outputs of our hash function to be at least 256 bits. In this way, the advent of quantum computers does not greatly affect the security of one-way hash functions. We will see that this theme continues for the remaining properties.

If g is a hash function such that any attacker making at most t_{OW} operations has success probability at most ϵ_{OW} of producing a preimage, then we call g a $(t_{\mathrm{OW}}, \epsilon_{\mathrm{OW}})$-preimage resistant hash function.

2.1.2 Second-preimage Resistance

Definition 2.1.3. *A hash function $g : \{0,1\}^* \to \{0,1\}^n$ is said to be* second-preimage resistant *if given a random element $x' \xleftarrow{\$} \{0,1\}^*$, it is computationally infeasible to produce an element $x \in \{0,1\}^*$ such that $x \neq x'$ and $g(x) = g(x')$.*

In other words, given a random preimage it is computationally infeasible to produce a second, distinct value which has the same image under g as the first.

The fastest classical generic attack against second-preimage resistance is exactly the same as for preimage resistance. The best that the attacker can do is to try arbitrary inputs until they find one that works. So again, we expect about 2^n attempts before a second preimage is found.

Similar to the preimage resistance case, Grover's algorithm can be used to attack second-preimage resistance. Suppose that $x' \in \{0,1\}^m$ is a given second-preimage challenge, where m is a positive integer. Compute $y = g(x')$. Define a function $h_y : \{0,1\}^n \to \{0,1\}$ by

$$h_y(x) = \begin{cases} 0 & \text{if } g(x) \neq g(x') \text{ or } x = x' \\ 1 & \text{if } g(x) = g(x') \text{ and } x \neq x'. \end{cases}$$

We can apply Grover's algorithm to h_y to find a second preimage of x' in $O(2^{n/2})$ operations.

If g is a hash function such that any attacker making at most t_{SPR} operations has success probability at most ϵ_{SPR} of producing a second-preimage of g, we call g a $(t_{\text{SPR}}, \epsilon_{\text{SPR}})$-second-preimage resistant hash function.

2.1.3 Collision Resistance

Definition 2.1.4. *A hash function $g : \{0,1\}^* \to \{0,1\}^n$ is said to be* collision resistant *if it is computationally infeasible to produce two distinct values $x, x' \in \{0,1\}^*$ such that $g(x) = g(x')$. Such a pair (x, x') is called a* collision.

The key difference between collision resistance and second-preimage resistance is that in second-preimage resistance the attacker is challenged with finding a second preimage of a given value, whereas in collision resistance the attacker is challenged with finding *any* pair which have the same hash value. Observe that by the pigeonhole principle, collisions are guaranteed to exist if the domain is strictly larger than the range.

It may not be surprising to hear that because an attacker has more "freedom" in an attack on collision resistance that the best generic classical attack is faster than 2^n operations. Consider for instance the well known Birthday Paradox: if 23 people are gathered in one room, then there is about a fifty percent chance that at least one pair of the people share the same birthday. We calculate this probability by first calculating its complement as follows. Label the people as person 1, person 2 and so on. Then the probability that no two people share a birthday is given as the product of the probabilities of the following events: person 1 has a birthday, person 2's birthday is different from

person 1's, person 3's birthday is different from both person 1's and person 2's and so on. Explicitly we have that

$$Pr(\text{no two people share a birthday}) = 1 \times \frac{364}{365} \times \frac{363}{365} \times \cdots \times \frac{343}{365} \approx 0.4927.$$

Thus, by complements, the probability that at least one pair of people share a birthday is about 0.5073.

How this translates to an attack on collision resistance is clear. If an attacker queries the hash function on m distinct inputs, then the probability that at least two of those inputs collide is

$$1 - \prod_{i=0}^{m-1} \frac{2^n - i}{2^n}.$$

We can obtain an approximation for this value by treating the events as independent. That is to say we can assume that the probability that any particular pair of inputs do not collide is $(2^n - 1)/2^n$. Thus, by observing that there are $m(m-1)/2$ such events the above probability can be approximated as

$$1 - \left(\frac{2^n - 1}{2^n}\right)^{\frac{m(m-1)}{2}} = 1 - \left(1 - \frac{1}{2^n}\right)^{2^n \frac{m(m-1)}{2 \cdot 2^n}} \approx 1 - \exp\left(\frac{-m(m-1)}{2^{n+1}}\right) \approx 1 - \exp\left(\frac{-m^2}{2^{n+1}}\right).$$

Inverting the above function yields the formula for the approximate number of queries (denoted $m(p)$) needed to find a collision with probability p.

$$m(p) \approx \sqrt{2^{n+1} \ln\left(\frac{1}{1-p}\right)}.$$

If we set $p = 0.5$, we get that $m(0.5) \approx 1.1774 \times 2^{n/2}$. Hence, we need about $m = 2^{n/2}$ queries to find a collision with 50% probability. Exploiting this fact of probability is called a *birthday attack*.

In practice birthday attacks have enormous storage requirements. There do exist more sophisticated methods for finding generic collisions in hash functions, but their expected number of operations is not any less than a birthday attack. For example, in 1999 van Oorschot and Wiener [36] devised a highly parallelizable method for finding generic collisions which has the advantage of having very small storage requirements. Brassard et al. in 1997 developed a quantum collision finding algorithm that runs in time $O(2^{n/3})$ [4]. However, as observed by Bernstein [3], when one takes into account the storage requirements of this algorithm, the true cost is not better than the van Oorschot-Wiener method.

If g is a hash function such that any attacker making at most t_{CR} operations has success probability at most ϵ_{CR} of producing a collision, then we call g a $(t_{\mathrm{CR}}, \epsilon_{\mathrm{CR}})$-collision resistant hash function.

2.1.4 Subset Resilience

Subset resilience is a less studied property of hash functions, but nonetheless it is still an important one, especially for our purposes. We give essentially the definition from [28], but first we need to define what it means for a function to be negligible in some parameter.

Definition 2.1.5. *Let $\epsilon_n : \mathbb{N} \to [0,1]$ be a function. We say that ϵ_n is* negligible in n *and write $\epsilon_n = \mathsf{negl}(n)$ if, for any positive integer $c \in \mathbb{N}$, there exists an $n_c \in \mathbb{N}$ such that $\epsilon_n < n^{-c}$ for all $n > n_c$.*

Let $\mathcal{H} = \{H_{i,t,k}\}$ be a collection of functions, where $H_{i,t,k}$ maps an input of arbitrary length to a subset of size at most k of the set $\{0, 1, \ldots, t-1\}$, and i is an index. One can think of $\mathcal{H}$ as an indexed collection of families of functions. Moreover, assume that there is a polynomial-time algorithm such that given $i, t, k \in \mathbb{N}$ and $M \in \{0,1\}^*$, computes $H_{i,t,k}(M)$.

Definition 2.1.6. *We say that $\mathcal{H}$ is* r-subset-resilient *if, for every probabilistic polynomial-time adversary A,*

$$\Pr_i\left((M_1, M_2, \ldots, M_{r+1}) \leftarrow A(i, 1^t, 1^k) : H_{i,t,k}(M_{r+1}) \subseteq \bigcup_{j=1}^{r} H_{i,t,k}(M_j)\right) < \mathsf{negl}(t,k).$$

In other words, the probability that the adversary outputs $r+1$ messages such that the hash of the last message is contained in the collection of the other r hash values is negligible in the security parameters.

2.2 Signature Schemes

Theoretically every person has a unique hand-written signature; and in a perfect world, no one would be able to successfully replicate another's signature. Under this assumption, if you sign a contract then any person who views that contract could verify that indeed it is your signature on the contract, and hence that you agree to the terms and conditions

of said contract. Similarly, if you sign a cheque you are verifiably agreeing to that exchange of funds, and so on. The purpose of your signature is for some form of *data origin authentication*.

This same concept exists mathematically and electronically. Suppose that you look at your cellphone and see a notice for a software update. How do you know that this software update is coming from a reliable source and not a malicious attacker? Signatures! Your phone receives a signature from the company trying to give you the update and then verifies that indeed this signature is that of the legitimate company. Assuming that signatures cannot be forged — which we prove is the case — then you can rest assured that you are receiving a legitimate, safe, software update. In actuality the process is much more involved and nuanced than this; there are plenty of security considerations and technical details, but this high level idea is all we need. We refer to these sorts of signatures as *digital signatures*.

Digital signatures are also used for *non-repudiation*. Because signatures cannot be forged, each entity can only produce the signature for itself. Suppose that a document is signed by some user. Since that signature could only have come from that user, they cannot later claim that they did not sign it. Verifiably, we know that this user signed the document; they cannot lie.

The third major use of digital signatures is for *data integrity*. One key difference between physical signatures and digital signatures is that a physical signature is the same regardless of what document it is on; your signature in principle always looks the same. However, in the case of digital signatures, each signature is theoretically unique to the document it is signing. Thus, if a document is signed by a legitimate user, then when this document/signature pair is received by someone else they can check to see if the contents of the document changed at any point after it was signed. As the signature placed on the document is unique to the document the signature will not verify if the contents of the document were changed. The signature would be rejected.

We give a formal definition of a digital signature scheme (DSS) now.

Definition 2.2.1. *A* digital signature scheme *is a triple of algorithms* (Kg, Sign, Vrfy) *where:*

- Kg *is called the* key generation algorithm. *On input of a security parameter* n*,* Kg *outputs a key pair* (sk, pk)*, where* sk *is known as the* secret key, private key, or signing key*, and* pk *is known as the* public key, or verification key*.*
- Sign *is called the* signing algorithm. *On input of a signing key* sk *and a message* M*,* Sign *outputs a digital signature* σ*.*

- Vrfy *is called the* verification algorithm. *On input of a verification key* pk*, a signature* σ *and a message* M*,* Vrfy *outputs* 1 *or "accept" if* σ *is a valid signature on* M*, and outputs* 0 *or "reject" otherwise.*

- *We also require the correctness condition that if* $(\mathsf{sk},\mathsf{pk})$ *is a valid key pair output by* Kg*, then for any message* M*,*

$$\mathsf{Vrfy}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, M), M) = 1.$$

  In other words, any signature gained from signing a message with sk *will verify using* pk*.*

Each key pair corresponds to an *instance* of the scheme; so when we say an "instance" of a DSS, we mean an instantiation using one specific key pair.

The concept of a DSS was introduced in the same paper as both public-key cryptography and the Diffie-Hellman key-exchange protocol were in 1976 by the name-sakes Diffie and Hellman [13], perhaps the most famous paper in the history of cryptography. However, the first example of a DSS was not introduced until 1978 by Rivest, Shamir, and Adleman in [34], perhaps the second most famous paper in cryptography.

Most signature schemes in use today (classical signature schemes) rely on hard number theoretic problems such as factoring large integers or the discrete logarithm problem for their security. Unfortunately, due to the work of Shor [35] all such schemes are broken in polynomial time by an adversary who has access to a quantum computer. The purpose of this thesis is to study signature schemes whose security relies solely on the security of hash functions; as we previously discussed, quantum adversaries do not greatly affect the security of hash functions. We give our first example of a *hash-based digital signature scheme* in Section 2.3.

Before we proceed with our discussion of digital signature schemes, we need to rigorously define what we mean by a *secure DSS*. It is necessary to build up the security vocabulary we will need. Although this vocabulary is universal in cryptography, we will only use it in the context of digital signature schemes.

By a *forgery* of a signature, we mean a valid message/signature pair which was not produced by the honest user, but rather by an adversary. In other words, a signature created by an attacker that is accepted by the verification algorithm. There are various ways in which an attacker can go about attempting to produce a forgery, but we shall discuss our security under the assumption that an attacker performs a so-called *chosen message attack* (CMA), the most general and liberal of the classical attack types. In this

attack model the adversary is allowed to query the honest user (or an oracle) for the valid signatures of some number of messages of the adversary's choosing, and the honest user must oblige. This can be done *adaptively*, i.e., the attacker queries one message at a time and can choose their next query using the knowledge they gained from their previous query, or the attacker can query all of their messages all at once, non-adaptively.

Finally, if we say that a signature scheme is *existentially forgeable*, then we mean that an attacker is able to feasibly forge a valid message/signature pair (M, σ) for *some* message M. Conversely, if a signature scheme is *existentially unforgeable* then no attacker can feasibly forge a valid message/signature pair for any message. To put this all together, we say that a signature scheme is *existentially unforgeable under a chosen message attack* (EU-CMA) if no attacker can feasibly forge a valid message/signature pair for any message using a chosen message attack. We formalize this below in terms of a game (sometimes called an *experiment*).

Definition 2.2.2. *Let* (Kg, Sign, Vrfy) *be a digital signature scheme with security parameter n. Let $\mathcal{A}$ be an adversary against this DSS and let $\mathcal{C}$ be the challenger. Then the* EU-CMA game *is the following.*

1. *$\mathcal{C}$ runs* Kg *with input n and receives* (pk, sk).
2. *$\mathcal{C}$ gives* pk *to $\mathcal{A}$.*
3. *$\mathcal{A}$ selects messages $M_1, M_2, \ldots, M_q$ and gives these messages to $\mathcal{C}$.*
4. *$\mathcal{C}$ returns the valid signatures $\sigma_1, \sigma_2, \ldots, \sigma_q$ to $\mathcal{A}$.*
5. *$\mathcal{A}$ outputs (M^*, σ^*).*

$\mathcal{A}$ wins the game if σ^* is a valid signature for the message M^* and $M^* \notin \{M_1, M_2, \ldots, M_q\}$.

Note that written as is the above game is non-adaptive, but it is clear how it could be modified to become adaptive. The two versions of the game are almost always distinguished. We also reiterate that $q = poly(n)$. For convenience we will notate the success probability of the adversary $\mathcal{A}$ in this game as

$$\mathrm{Exp}^{\text{EU-CMA}}_{\mathrm{DSS}(1^n)}(\mathcal{A}).$$

Definition 2.2.3. *Let* DSS *be a digital signature scheme and $n \in \mathbb{N}$ a security parameter. We say that* DSS *is* EU-CMA*-secure if for all q, t polynomial in n, the maximum success probability denoted* $\mathrm{InSec}^{\text{EU-CMA}}(\mathrm{DSS}(1^n); t, q)$ *of all (possibly probabilistic) adversaries $\mathcal{A}$*

running in time at most t and making at most q queries in the EU-CMA *game is negligible in n. Symbolically,*

$$\mathrm{InSec}^{\text{EU-CMA}}(\mathrm{DSS}(1^n); t, q) := \max_{\mathcal{A}} \left\{ \mathrm{Exp}^{\text{EU-CMA}}_{\mathrm{DSS}(1^n)}(\mathcal{A}) \right\} = \mathsf{negl(n)}.$$

We refer to this value as the *Insecurity function* of the DSS.

If DSS is a digital signature scheme that is existentially unforgeable against any adversary running in time at most t and making at most q queries to the signing algorithm and who has a success probability at most ϵ, then we say the DSS is (t, q, ϵ)-EU-CMA secure. In the next section, where we introduce one-time signature schemes, we add the restriction that $q = 1$.

2.3 The Lamport-Diffie One-Time Signature Scheme

First proposed in [24], the Lamport-Diffie one-time signature scheme (LD-OTS) is an important signature scheme, one which we shall use as a launching point for our discussion of provably secure, practical and efficient hash-based digital signature schemes. The scheme takes as input a security parameter n, and uses a one-way function

$$f : \{0,1\}^n \to \{0,1\}^n$$

and a collision resistant hash function

$$g : \{0,1\}^* \to \{0,1\}^n.$$

In what follows, we use superscripts to denote a secondary index.

Key generation: On input of security parameter n do as follows.

1. Select $2n$ bitstrings, $x_0^0, x_0^1, x_1^0, x_1^1, \ldots, x_{n-1}^0, x_{n-1}^1 \xleftarrow{\$} \{0,1\}^n$.
2. For each pair (i, j) with $i \in \{0, 1, \ldots, n-1\}$ and $j \in \{0, 1\}$, compute $y_i^j = f(x_i^j)$.

The secret key is $X = (x_0^0, x_0^1, \ldots, x_{n-1}^0, x_{n-1}^1)$, and the public key is the collection of images under f of these secret bitstrings: $Y = (y_0^0, y_0^1, \ldots, y_{n-1}^0, y_{n-1}^1)$.

Signature generation: To sign a message $M \in \{0,1\}^*$ using secret key X do as follows.

1. Compute the message digest $d = g(M)$.

2. Parse d as $d = d_0 \| d_1 \| \ldots \| d_{n-1}$.

The signature on M is

$$\sigma = (x_0^{d_0}, x_1^{d_1}, \ldots, x_{n-1}^{d_{n-1}}).$$

In other words, the signer reveals the n coordinates of X that correspond to the n bits of d.

Signature verification: To verify a signature σ on a message M using verification key Y do as follows.

1. Compute the message digest $d = g(M)$ as before.
2. Check if
$$(f(\sigma_0), f(\sigma_1), \ldots, f(\sigma_{n-1})) = (y_0^{d_0}, y_1^{d_1}, \ldots, y_{n-1}^{d_{n-1}}),$$
where σ_i is the i^{th} coordinate of σ.

The signature is accepted if and only if the above comparison holds.

The security of the above scheme relies on the one-wayness of the function f and the collision resistance of g. We require f to be one-way, for else an attacker could simply recover the secret key from the public key. We require g to be collision resistant, for else if an attacker had two distinct messages M and M' such that $g(M) = g(M')$ they could query for the signature of M and then produce a successful forgery by simply swapping M' for M.

2.3.1 One-Time Use

LD-OTS is a so-called *one-time signature scheme*. It is a signature scheme which is (provably) secure only if it is used to sign at most one message. Trivially a signature scheme becomes insecure if the signing key is known to an attacker; and a single LD-OTS signature reveals *half* of the secret key. Reasonably one would expect that if a second message were signed, about three-quarters of the secret key would be revealed if you include the first signature. We see that security degrades rapidly as more messages are signed. And so, the LD-OTS scheme should only be used to sign at most one message per instance.

A solution to this problem of one-time use is to reveal a smaller fraction of the secret key than one half. To this end we introduce the hash-based signature scheme HORS (Hash to Obtain a Random Subset) [28]. HORS was conceived as an improvement on the "Balls and Bins signature scheme" [32]. HORS has parameters $n, k, t \in \mathbb{N}$ where t is a power of 2, and utilizes a one-way function

$$f : \{0,1\}^n \to \{0,1\}^n$$

and an r-subset-resilient hash function

$$g : \{0,1\}^* \to \{0,1\}^{k \log(t)},$$

where r is the maximum number of messages the user wishes to sign securely with a single key pair.

Key generation: On input of parameters n, k, t do as follows.

1. Select t bitstrings, $x_0, x_1, \ldots, x_{t-1} \xleftarrow{\$} \{0,1\}^n$.
2. Compute $y_i = f(x_i)$ for $i = 0, 1, \ldots, t-1$.

The secret key is $X = (k, x_0, x_1, \ldots, x_{t-1})$ and the verification key is $Y = (k, y_0, y_1, \ldots, y_{t-1})$.

Signature generation: To sign a message $M \in \{0,1\}^*$ using secret key X as above do as follows.

1. Compute the message digest $d = g(M)$.
2. Parse d as k bitstrings $d_0, d_1, \ldots, d_{k-1}$, where each d_j is of length $\log(t)$. Interpret each d_j as a non-negative integer.

The signature on M is

$$\sigma = (x_{d_0}, x_{d_1}, \ldots, x_{d_{k-1}}).$$

Signature verification: To verify a signature σ on a message M using verification key Y as above,

1. Compute the message digest $d = g(M)$.
2. Parse d as in signature generation. Check if

$$(f(\sigma_0), f(\sigma_1), \ldots, f(\sigma_{k-1}) = (y_{d_0}, y_{d_1}, \ldots, y_{d_{k-1}}).$$

The signature is accepted if and only if the above comparison holds.

Observe that in the HORS signature generation algorithm at most k coordinates of the t-element secret key are revealed (the d_j are not necessarily distinct). And so, if k is much smaller than t it seems intuitively clear that the degradation of security is much slower in HORS than in LD-OTS as more messages are signed. In fact, HORS can be used to securely sign more than one message. It is an example of a *few-time signature scheme* (FSS). The exact number of messages which can be signed securely depends on the value of r. In other words, it depends on the subset-resilience of the hash function employed.

For example, if $k = 32$ and $t = 2^{16}$, then the fraction of the secret key revealed by a signature is at most $32/2^{16} = 0.00048828125$. In other words, a signature would reveal at most about 0.0489 percent of the secret key. It is for this reason that we require g to be subset-resilient. Signatures are made by parsing the hash values of messages and revealing those corresponding pieces of the secret key. Hence, if g is r-subset-resilient then it is with negligible probability that the hash of a new message is contained in the collection of previously output hash values (from the r previously produced signatures). Hence, an attacker cannot simply look at r previous message/signature pairs and construct a forgery from them.

We also describe another solution to the problem of one-time use in Section 2.3.3.

2.3.2 Large Private Keys

In order to be useful in practice, a signature scheme needs to have a reasonably small private key. However, we see that both LD-OTS and HORS suffer from massive secret keys. The size of an LD-OTS secret key is $2n^2$ bits and a HORS secret key is of length tn bits. In general, for meaningful values of n and t these keys can become too large and greatly effect the practicality of the scheme. However, we can mitigate this flaw with the use of a deterministic pseudorandom function. Instead of storing a massive secret key, we can store a relatively short secret seed and then generate the secret key on the fly. This cuts down on the large storage requirements, and assuming one makes a careful choice of pseudorandom function, the key generation is done efficiently. This technique is used commonly in the schemes we discuss in later chapters.

2.3.3 Large Public Keys

Just as having large secret keys can be burdensome, having large public keys detracts from the practicality of a signature scheme. Again, both LD-OTS and HORS suffer from this deficiency. Unfortunately, we cannot use pseudorandom functions to generate public keys on the fly because the public keys are derived from the secret keys, and of course we would like the secret keys to remain secret. Instead we introduce a powerful tool: the *Merkle Signature Scheme (MSS)*.

In 1979 Ralph Merkle (one of the inventors of public-key cryptography) patented the so-called *Merkle tree* [30]; a binary tree where the internal nodes are the hash values of the concatenation of their children. Merkle's intent was to use the root node of this tree as a way to check the validity of many one-time public keys. Let $g : \{0,1\}^* \to \{0,1\}^n$ be

a one-way, and collision resistant hash function, and let OTS be any one-time signature scheme.

Key generation: To generate the Merkle tree, one first selects a parameter $h \in \mathbb{N}$. This parameter dictates how many one-time keys we can verify, namely 2^h. Next, generate 2^h OTS key pairs (X_i, Y_i), with indexing starting at 0. The leaf nodes of the tree are the hashes of the public keys, $g(Y_i)$. The tree will have $h+1$ levels, level h will contain just the root node and level 0 will contain the leaves. We denote the i^{th} node on level j by $v_j[i]$ indexing from left to right starting at 0; see Figure 2.1. Then, for $i = 0, 1, \ldots, 2^h - 1$, $v_0[i] = g(Y_i)$, and

$$v_j[i] = g(v_{j-1}[2i] \| v_{j-1}[2i+1]), \quad 1 \leq j \leq h,\ 0 \leq i \leq 2^{h-j}.$$

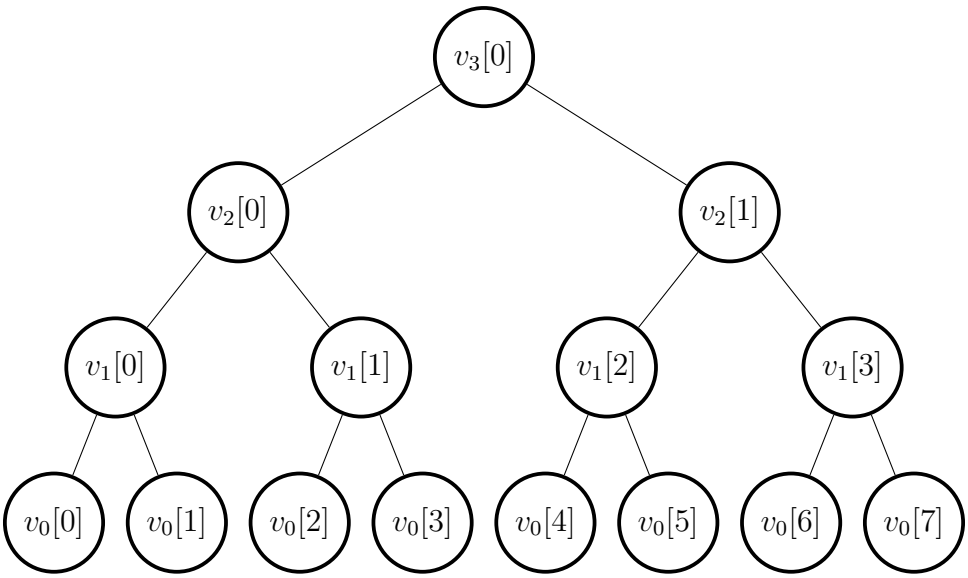

Figure 2.1: A Merkle tree with $h = 3$.

The MSS public key is the root node $v_h[0]$, and the secret key is the collection of one-time secret keys $\mathcal{X} = \{X_0, X_1, \ldots, X_{2^h-1}\}$. Again, these can be generated on the fly via pseudorandom functions. See the survey by Buchmann et al. [1, Chapter 2] for a more in-depth analysis of how this can be done efficiently. Moreover, the authors of [1] note that if a so-called forward secure pseudorandom function is used, then in fact the Merkle signature scheme itself becomes forward secure. A signature scheme is said to be *forward*

secure if when a current secret key is obtained by an adversary, the adversary still cannot feasibly forge signatures corresponding to past keys. Similarly, a stateful pseudorandom function is forward secure if an adversary who knows the current secret internal state of the function cannot feasibly distinguish outputs corresponding to an earlier state from random. In the context of the MSS forward security is desirable because, if the secret key of a current node is compromised, then one needn't worry about the security of earlier signatures, but rather only that of signatures not yet produced. In other words, such a compromise causes no harm to past signatures.

Signature generation: To sign message $M \in \{0,1\}^*$ using secret key $\mathcal{X}$ as above first compute $d = g(M)$. If this is the s^{th} message to be signed using this instance of the MSS, then we sign d using the s^{th} OTS secret key X_s; call this signature σ_{OTS}. The signature will also contain the s^{th} public key, so this too must be calculated. It remains to calculate the *authentication path*. The authentication path $\mathsf{Auth}_s = (a_0, a_1, \ldots, a_{h-1})$ corresponding to σ_{OTS} is the ordered collection of sibling nodes along the tree which are required to compute the root node. The a_j are computed as follows:

$$a_j = \begin{cases} v_j[\lfloor s/2^j \rfloor - 1] & \text{if } \lfloor s/2^j \rfloor \text{ is odd} \\ v_j[\lfloor s/2^j \rfloor + 1] & \text{if } \lfloor s/2^j \rfloor \text{ is even.} \end{cases}$$

The MSS signature on M is

$$\sigma = (s, Y_s, \sigma_{OTS}, \mathsf{Auth}_s).$$

Signature verification: To verify an MSS signature σ on a message M we must first use the one-time verification key Y_s to verify the one-time signature σ_{OTS} on M. The verification algorithm proceeds if and only if the OTS verification succeeds. Secondly, we must confirm the validity of the public key Y_s. To do so, we use Y_s together with the authentication path to construct the root node of the Merkle tree as follows. We compute a path $P_s = (p_0, p_1, \ldots, p_h)$ by setting $p_0 = g(Y_s)$, and

$$p_j = \begin{cases} g(a_{j-1} \| p_{j-1}) & \text{if } \lfloor s/2^{j-1} \rfloor \text{ is odd} \\ g(p_{j-1} \| a_{j-1}) & \text{if } \lfloor s/2^{j-1} \rfloor \text{ is even} \end{cases}$$

for $j > 0$; see Figure 2.2. The algorithm accepts the signature if and only if p_h equals the public key (the root node).

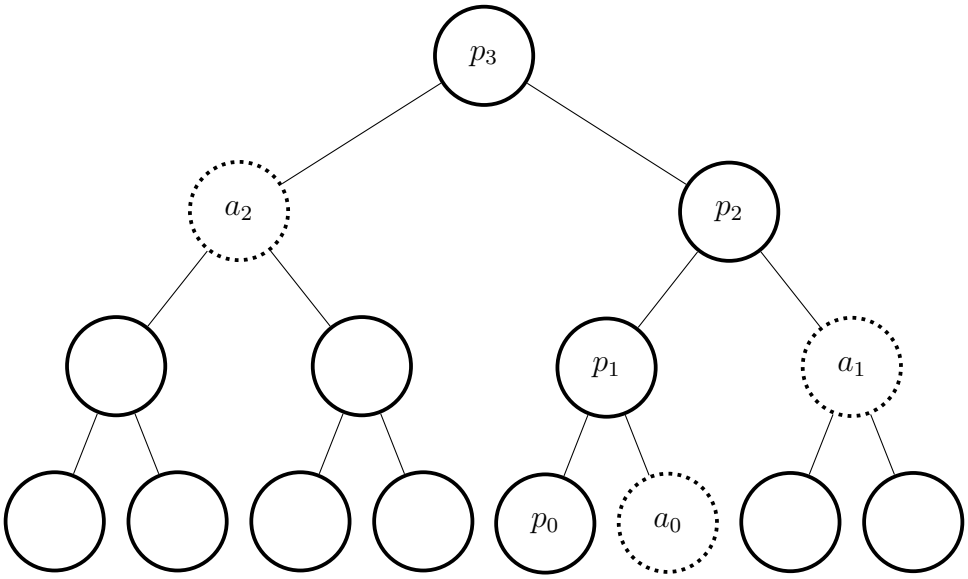

Figure 2.2: Authentication path for $s = 4$, $h = 3$.

To be explicit, by using a Merkle tree one can produce a secure signature scheme where the public key size is only n bits as opposed to say $2n^2$ bits. Furthermore, this signature scheme can be used to sign 2^h messages as opposed to just a single message. However, this construction still has some drawbacks which we continue to discuss below.

2.3.4 Reliance on Collision Resistance

In the security proof for MSS we require the assumption that the hash function used is collision resistant. Intuitively, one can think that if the underlying one-time signature scheme is secure, then if a forgery is successfully produced it must be the case that a collision occurred somewhere within the tree. However, in the real-world, collision resistance is a very strong property for a hash function to have. For example, previously popular hash functions such as MD4, MD5 and SHA-1 have all fallen from the good graces of cryptographers because modern day attacks on these algorithms makes them question or completely abandon the assumption that the functions are collision resistant. For practical purposes we would like to be able to relax the need for collision resistance to second-preimage resistance. To accomplish this goal we introduce bitmasks, and describe a slight modification to the MSS due to Dahmen et al. [12].

For each level of the tree with the exception of the leaf level, we uniformly at random select a bitmask $Q_j \in \{0,1\}^{2n}$. Instead of hashing the concatenation of two sibling nodes to create the parent node, we first take the **xor** of the concatenation of the sibling nodes with the bitmask corresponding to that level and then hash. Symbolically,

$$v_j[i] = g(Q_j \oplus (v_{j-1}[2i] \| v_{j-1}[2i+1])).$$

Suppose for now that the public keys of the underlying OTS consist of $L = 2^\ell$ n-bit strings for some fixed ℓ. In the previous incarnation of the MSS, the leaves of the tree were the hashes of the OTS public keys. Here, the leaves will be the bitstrings which comprise the 2^h OTS public keys — in the natural order. So instead of having a tree with $h+1$ levels and 2^h leaves, we have a tree with $h+\ell+1$ levels, and $2^h 2^\ell = 2^{h+\ell}$ leaves.

Each OTS public key Y_i corresponds to its own subtree, and the same bitmasks are used for each such tree. There is a slight problem with this construction if L is not a power of two. However, this is easily fixed by creating an unbalanced subtree; a left node which has no sibling is lifted to a higher level until it has one. The result is that the full tree has $h + \lceil \log(L) \rceil + 1$ levels. We'll refer to these subtrees as *L-trees*.

The rest of the signature scheme is exactly the same. In particular, we mention that the authentication paths have the same length here as in the previous incarnation of MSS. In other words, they begin on level $\lceil \log(L) \rceil$; the level with the root nodes of the L-trees. The authors of [12] refer to this construction as SPR-MSS, for "second-preimage resistant Merkle signature scheme".

We claim that these modifications relax the need for collision resistance in the Merkle signature scheme. First, it is important to understand why second-preimage resistance is not sufficient in the vanilla version of the MSS. Recall that in the second-preimage challenge, one requires that the input (the first-preimage) of the hash function be chosen uniformly at random from its domain. Then the adversary attempts to produce a second element of the domain which differs from the first, but results in the same hash value. If we attempted to prove security of the MSS in the standard model, we lose the assumption that the hash function is a random function. The leaves of the tree are the hashes of public keys, and hence, they are not uniformly random as they are the outputs of a non-random function. Moreover, the concatenation of pairs of these leaves are also not uniformly random. However, these concatenations are the inputs into the hash function when we produce the next level; and so, we have non-uniformly random inputs into the hash function for each higher level, and so we fail to fulfill the premise for a second-preimage challenge.

Notice that if r is a uniformly random n-bit string, then $r \oplus x$ is also uniformly random for any n-bit string x. Hence, even though $(v_{j-1}[2i] \| v_{j-1}[2i+1])$ may not be uniformly

random for some (i, j), we have that $(Q_{j-1} \oplus v_{j-1}[2i] \| v_{j-1}[2i+1])$ is indeed uniformly random; and so, inputs into the hash function are in fact uniformly random in SPR-MSS.

Notice that in this construction leaf nodes are not **xor**'ed with any bitmasks. Suppose that an attacker produced a valid forgery for some message, but such that the OTS verification key Y_s given in the signature is not the original key produced by the user. This implies that a leaf node has changed. However, since the signature is still valid, this means that there must have been a collision while computing the parent node of that leaf. The security reduction for SPR-MSS in [12] accounts for this possibility, and provably second-preimage resistance suffices. Thus, we need not include bitmasks for the leaf nodes.

One trick which various authors employ to get better security reductions with these sorts of schemes is to replace functions with families of functions parameterized by some set, where each member of the family has the same security property: one-way, collision resistant, second-preimage resistant, etc. MSS and SPR-MSS can be modified in this way by replacing the hash functions with a family of hash functions and then randomly selecting a member of this family to use in any particular instance. LD-OTS is often presented with this modification.

We omit the technical details, but we have the following theorem from [12].

Theorem 2.3.1. *If $\mathcal{H}_K = \{H_k : \{0,1\}^{2n} \to \{0,1\}^n\}_{k \in K}$ is a family of $(t_{\text{SPR}}, \epsilon_{\text{SPR}})$-second-preimage resistant hash functions with $\epsilon_{\text{SPR}} \leq 1/(2^{h+\ell+1} - 2)$ and the employed one-time signature scheme is a $(t_{\text{OTS}}, \epsilon_{\text{OTS}}, 1)$ signature scheme with $\epsilon_{\text{OTS}} \leq 1/2^{h+1}$, then the SPR-MSS is a $(t, \epsilon, 2^h)$ signature scheme with*

$$\epsilon \leq 2 \cdot \max\{(2^{h+\ell} - 1) \cdot \epsilon_{\text{SPR}}, 2^h \cdot \epsilon_{\text{OTS}}\}$$
$$t = \min\{t_{\text{SPR}}, t_{\text{OTS}}\} - 2^h \cdot t_{\text{SIG}} - t_{VER} - t_{\text{GEN}},$$

where $t_{\text{SIG}}, t_{\text{VER}}, t_{\text{GEN}}$ are the number of operations needed to sign a message, verify a message, and generate a key pair for the underlying OTS.

Although not mentioned in the above theorem, the proof actually gives the stronger result of strong unforgeability.

With the introduction of SPR-MSS, the condition of collision resistance for the Merkle signature scheme has been successfully relaxed. However, for each of the one-time schemes we have introduced so far, our security relies on collision resistance. In particular, the MSS no longer needs collision resistance, but we still need collision resistance in LD-OTS. Clearly if we don't address this issue as well then we really haven't made any significant improvements. Thankfully, relaxing the need for collision resistance in LD-OTS is fairly

simple. The modification is due to Leighton and Micali [23] and was originally intended for the *Winternitz signature scheme* which we discuss in Chapter 3, but works equally well for any hash-based signature scheme. Suppose that a signer wishes to sign a message M. Instead of just hashing M down to n bits with g, they first select a uniformly random bitstring r and sign the message digest $d = g(r\|M)$. Everything else remains exactly the same except the value of r is included in the signature.

Before, if an attacker knew of a collision $g(M) = g(M')$ for some $M \neq M'$, they could query for a signature of M and receive (M, σ) and then produce a forgery as (M', σ). Now, if we include some randomness, finding an arbitrary collision with g is insufficient to produce a forgery. Since the attacker has no knowledge of what the random value r will be before they query, they have no way of knowing ahead of time what the resulting message digest will be, and so knowing a collision is useless to them. They would need to produce values M' and r' such that $g(r', M') = g(r, M)$ — assuming of course that r is long enough to not be guessed feasibly. Suppose that the attacker had the values M and r. If they evaluated the hash function on say q random inputs, then they only have probability $q/2^n$ of finding a pair (r', M') which collide with (r, M).

This modification has the added benefit of allowing us to halve the hash length of g because we no longer need to worry about birthday attacks.

2.3.5 Small Bound on the Number of Signatures

One can sign 2^h messages with SPR-MSS; this hardly seems like a small upper bound. It is hard to imagine one signing say 2^{40} messages within the lifetime of one key — that's over a trillion messages! Theoretically this is all quite fine, but a problem arises in implementation. We would need to generate 2^{40} OTS key pairs and then construct the Merkle tree and all the L-trees. The storage requirements and generation times required would be absurd. We need a way to make the implementation of SPR-MSS practical. It is to this end that we introduce *hyper trees*. We first describe a hyper tree with two layers, and then generalize to d layers.

Select two tree heights h_0 and h_1; the total height of the hyper tree will be $h = h_0 + h_1$. The first layer (the top layer) consists of a single tree of height h_1, the root of which will be the public key for the scheme. Layer zero will consist of one tree for each leaf of the tree on the layer above; and so there will be 2^{h_1} trees on this level. Each tree on layer zero will have height h_0. All trees are constructed using the SPR-MSS construction; that is their leaves are root nodes of L-trees (which we will interchangeably refer to as *compressed OTS public keys*) and each level has a corresponding, uniformly random bitmask. We use

the same bitmasks for each tree on level zero. We give a high-level description of the key generation, signature and verification algorithms.

Key generation: Given as input h_0, h_1 and security parameter n, first generate 2^{h_1} OTS key pairs (X_i, Y_i), $0 \leq i \leq 2^{h_1} - 1$. Next, randomly select $h = h_0 + h_1$ bitmasks $Q_j \in \{0,1\}^{2n}$ and bitmasks for the L-trees. To produce a public key we need only generate the tree on layer one. Do this according to the SPR-MSS construction using bitmasks $Q_{h_0+1}, Q_{h_0+2}, \ldots, Q_h$ for the tree itself. We use the same bitmasks for all L-trees. The root node will be the public key for the scheme, the secret key is the seed used to generate the OTS secret keys.

Signature generation: Given a message M and an index s, we wish to first sign M with the s^{th} leaf on layer zero. Determine which tree this leaf belongs to; if $0 \leq s \leq 2^{h_0} - 1$, then we use the first tree — indexing starting from 0. If $2^{h_0+k} \leq s \leq 2^{h_0+k+1} - 1$, for some positive integer k, then it belongs to tree k. The leaf we use in this tree is at index $s \bmod 2^{h_0}$. Sign M with this leaf to get σ_0, and then produce the authentication path Auth_0 to the root node $v_{h_0}[0]$ of the k^{th} tree on layer 0. Observe that to do this we need to generate this entire tree, but for reasonable choices of h_0 this is practical. Next, treat the root node of this tree as a message and sign it with the corresponding leaf on layer one to obtain σ_1; if we are using the k^{th} tree on layer zero then sign the root node with the k^{th} leaf on layer one. Compute the authentication path Auth_1 for this leaf. The signature is then: $\Sigma = (s, \sigma_0, \mathsf{Auth}_0, \sigma_1, \mathsf{Auth}_1)$.

Signature verification: Given a message M and a signature Σ, we verify Σ as follows. First, verify σ_0 on M. The algorithm proceeds if and only if the OTS verification algorithm succeeds. Next, as described in Section 2.3.3 use Auth_0 to produce the root node of the tree on layer zero. The next step is to verify the signature σ_1 on this root node. The algorithm proceeds if and only if the OTS verification algorithm succeeds. Next, use Auth_1 to compute the root node of the tree on layer one. The algorithm accepts the signature if and only if the root node equals the public key.

During key generation we need only construct the topmost tree. To sign a message we need only generate a tree on the layer below. Once we have signed 2^{h_0} messages, we remove the bottom tree from memory and then generate the tree corresponding to the second leaf, and so on. Hence, we need only generate/store $2^{h_1} + 2^{h_0}$ OTS key pairs. Suppose for concreteness that $h_0 = h_1 = h/2$ for some even height h. Then we generate $2^{1+h/2}$ key pairs. Without the use of this two-layer hyper tree we would have needed to generate 2^h such pairs; so this is nearly a square root improvement. Figure 2.3 shows a two-layer hyper tree where both layers are of height 3. For clarity we omit the L-trees from Figure 2.3, but remind the reader that they lie below each leaf node.

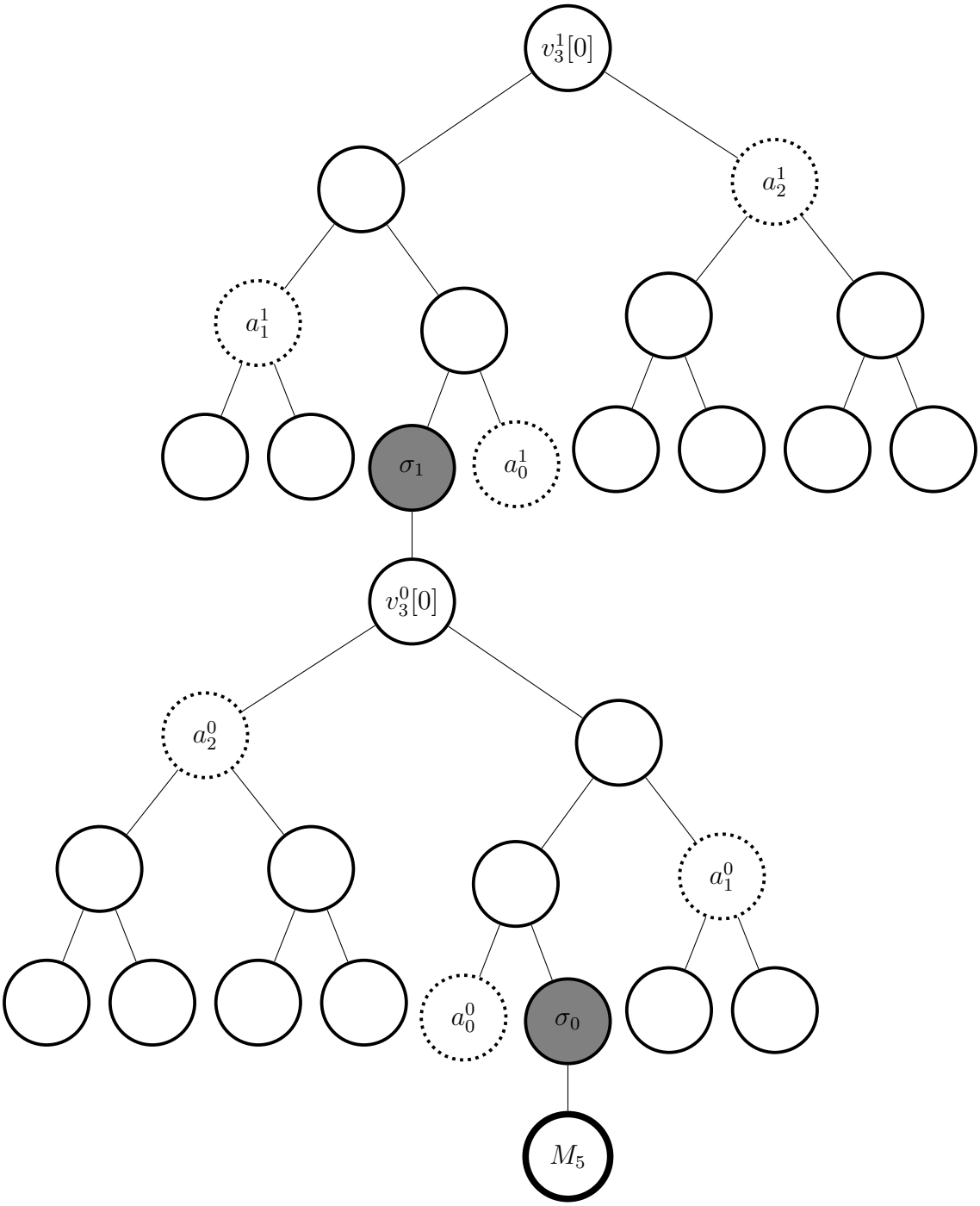

Figure 2.3: A two-layer hyper tree with $h_0 = h_1 = 3$.

It is worthwhile to make an obvious, yet important observation. Recall that each leaf on the topmost tree is a compressed OTS verification key, i.e., it can only securely sign one message. However, we use it to sign 2^{h_0} times. The observation here is that every time it is used, it signs the exact same message; namely the root node of the tree below it. Hence, even though it is used many times, the attacker gains no new knowledge by us repeatedly signing with it. That is of course assuming a deterministic signature algorithm for the underlying OTS scheme. It is conceivable that with a non-deterministic scheme that some information could be gained by the attacker if they see multiple signatures.

In general, to construct a d-level hyper tree, we first select d heights $h_0, h_1, \ldots, h_{d-1}$. Again, there will be a single tree on layer $d-1$ (the topmost layer). Layer $d-2$ will be exactly as we described layer 0 in the 2-layer case. However, now instead of signing messages with the leaves of the trees on layer $d-2$, we continue to construct an SPR-MSS type tree per each leaf of each of these trees (only one at a time of course), and so on. Messages are signed with the leaves of the trees on layer 0. In this construction, a signature will look like,

$$\Sigma = (s, \sigma_0, \mathsf{Auth}_0, \sigma_1, \mathsf{Auth}_1, \ldots, \sigma_{d-1}, \mathsf{Auth}_{d-1}).$$

Verification is done in a similar manner. We verify an OTS signature, then use the corresponding authentication path to obtain the root of a tree, and then repeat, but this time using that root as the new message we wish to verify. The signature verifies if and only if the root node on layer $d-1$ equals the public key.

2.3.6 Weakness Against Multi-user Attacks

Often not taken into consideration when discussing the security of signature schemes is the potential threat of a multi-user attack. Often when discussing security only the security of a single instance of a particular scheme is considered. However, these security arguments lose tightness when one considers multiple instances of a particular scheme, i.e., if there is more than one user of a scheme, each with their own sets of public and private keys. In this setting an attacker is challenged to produce a forgery for *at least one* of these users. So, they have a larger number of potential targets.

Let us return to our specific discussion on LD-OTS. We previously claimed that a particular instance of LD-OTS is EU-CMA assuming the one-wayness of a function (family) and the collision resistance of a hash function (family). While this certainly remains true, conceivably an attacker could get "lucky" and produce a forgery for some instance of the scheme if there are sufficiently many instances of the scheme currently in use. Let us clarify by using a trivial example. There are $N = 2^{2n^2}$ possible distinct LD-OTS public keys, and

of course in practice it is impossible to actually generate all of these keys for any meaningful value of n. However, suppose for the sake of argument that each of these N public keys is in use across some number of users. That means then that if the attacker selects any set $(x_0^0, x_0^1, \ldots, x_{n-1}^0, x_{n-1}^1)$ of n-bit strings and computes $d = g(M)$ for any message M, then $\sigma = (x_0^{d_0}, x_1^{d_1}, \ldots, x_{n-1}^{d_{n-1}})$, is a valid forgery for *some* instance of LD-OTS; of course the attacker is only successful if they can also tell you exactly what instance it is a forgery for. Obviously this is a silly example, but it should highlight something important: the more instances of LD-OTS that are in use at any particular time, the higher the probability that an attacker can find a forgery for at least one of them.

Let us make this a bit more concrete. Suppose that we have exactly one instance of LD-OTS, with public key Y. Since the one-way function is public, the attacker could just select a random $X \in \{0,1\}^{2n^2}$ and see if $f(X) = Y$. In other words, they can just make a guess at what the secret key is. After q such guesses, their probability of successfully finding the secret key is $q/2^{2n^2}$. Hence, to achieve k-bit security for one particular instance (where $k = \log(q)$), we need $k \geq 2n^2$. Now, suppose that N distinct instances of LD-OTS are in use for some positive integer N, with public keys $Y_1, Y_2, \ldots, Y_N$. In this case, instead of checking if $f(X) = Y$ for a fixed key Y, the attacker can check to see if $f(X) = Y_i$ for some $i = 1, 2, \ldots, N$. In this case, after q guesses, the probability of successfully finding a secret key for at least one instance of the scheme is $Nq/2^{2n^2}$. Thus, to achieve k-bit security for this multi-user setting we need $k + \log(N) \geq 2n^2$. Therefore, to achieve the same level of security, we must increase the output length n of the hash function as we increase the number of instances.

Additionally, in a similar manner, even if we concatenate our messages with a random string before hashing (as suggested in Section 2.3.4), we still get a degradation in security as more instances are added.

It is another suggestion by Leighton and Micali from [23] which fixes this problem. Each signer is given a distinct identity, and if they are using more than one instance of LD-OTS, they are also given a unique instance number for each one — instance numbers are never re-used. The user's identity and instance numbers are included with their public keys, and are used in the verification algorithm. Depending on the OTS used, this modification can be implemented in a number of ways, but for the LD-OTS scheme what one could do is compute message digests as $d = g(r \| \text{id} \| M)$, where r is a random string and id includes both the user's identity and their instance number. This makes it so that each query the attacker makes to g can be used to attack exactly one instance of the scheme since each query must be unique to some id.

Furthermore, it has been suggested by Katz [22] that we also include (for example)

identifiers for the functions we use. For example, instead of computing $y = f(x)$ for some x, we could compute $y = f(x\|00)$, and similarly $z = g(x\|01)$. The purpose of this modification is that potentially, f and g could be the same function, and these identifiers allow us to treat them each as independent functions. Katz also suggests further modifications if the functions need to be iterated, but we omit those details for now and revisit them in Section 3.2.

2.3.7 Statefulness

The final drawback we shall discuss in this section is that of *statefulness*. The modifications discussed thus far have been stateful, i.e., they need to keep an internal state or index which needs to be updated after each new message is signed. This is not entirely a bad thing as we can use statefulness to prove properties such as forward security. However, there are numerous difficulties with managing states. One difficulty is that there is some overhead associated with maintaining a state such as storage requirements. However, this is perhaps the least important problem associated with stateful signature schemes. McGrew et al. in 2016 discussed these problems in detail [25]. For example, a difficulty arises if a system needs to restore itself to some earlier point. Here it is possible for the system to restore itself to a point when it was using an earlier state. New messages get signed using earlier states and hence with previously used OTS keys. This could be disastrous for security. A similar issue arises if keys are backed up or copied onto other devices. Similarly, if multiple devices are all signing with the same keys (say one's desktop computer, phone, and laptop are all used) then an issue arises with *key synchronization*. See [25] for a more complete list and analysis of these state management problems.

We describe a stateless hash-based signature scheme due to Goldreich [14, 15]. Goldreich's construction uses a deterministic pseudorandom generator G and a collision resistant hash function $g : \{0,1\}^* \to \{0,1\}^n$. And of course, the construction uses an underlying OTS.

Key generation: Uniformly at random select a secret seed for G. Then G along with this seed will be used to generate $2^{n+1} - 1$ OTS key pairs; note that this is computationally infeasible for a meaningful value of n, but we clarify this at the end of this subsection. Each node in the entire tree will correspond to one such key pair, and as such, we need to maintain some sort of coherent ordering of the key pairs. Unlike the nodes of Merkle trees, nodes in this tree are not constructed using their children, but rather are the OTS verification keys themselves. The secret key will be the seed for G. The public key of the scheme is the OTS public key of the root node.

Signature generation: Goldreich described two possible ways to sign a message. We give both. Notice that the outputs of g are n-bits, and there are 2^n leaf nodes in the tree ($2^{n+1}-1$ nodes in total). So, Goldreich's first suggestion was to sign in a deterministic way: given a message M, compute $d = g(M)$ and treat the n-bit string d as an integer between 0 and $2^n - 1$. And so, d in base ten (denoted $(d)_{10}$) is the index of the leaf which we use to sign the message M, producing signature σ_0. Note that the underlying OTS should use a different hash function than g — or at least use identifiers as suggested by Katz. Next, take the OTS verification key corresponding to index $(d)_{10}$, and concatenate it with its sibling's verification key, then hash the result. Treat this new hash value as a message, and sign it using the parent node of the two siblings we just used. Call this new signature σ_1. Iterate this process until we use the root node to sign the hash of the concatenation of its childern to receive signature σ_n. The signature Σ on M consists of each of $\sigma_0, \sigma_1, \ldots, \sigma_n$, as well as each of the OTS public keys used to verify, and their respective sibling keys.

Goldreich's second suggestion was to randomly select a leaf node with which to sign M instead of deterministically selecting one; everything else remains the same. In the first method, we require g to be collision resistant, and in the second case, we need sufficiently many leaves so that the probability of selecting a leaf node to sign with more than once is acceptably small — recall that we are using one-time key pairs to sign with.

Signature verification: The verification algorithm should be pretty obvious; systematically verify signatures on messages as we build a path up to the root node. The algorithm accepts the signature if and only if each of the σ_i verifies for $i = 0, 1, \ldots, n$. In Figure 2.4 we show an example of the signature path from node 10, to the root node. Hatched nodes are used to sign and the corresponding sibling nodes are shown in bold. The signature produced is $(\sigma_0, \sigma_1, \sigma_2, \sigma_3, Y_{10}, Y_9, Y_4, Y_3, Y_2, Y_1)$. Note that node j's sibling is at $j+1$ if j is even, and at $j-1$ if j is odd. Similarly, the parent node of j is at index $(j/2) - 1$, if j is even, and at $\lfloor j/2 \rfloor$ otherwise.

The tree from this construction is what is known as a *virtual tree*. By that, we mean that the entire tree is never actually constructed. Consider the signing algorithm. If we use the deterministic method mentioned, then to achieve at least 128 bits of security we need the output of the hash function to be at least 256-bits so as to be secure against birthday attacks. However, the output length of the hash function corresponds to the number of levels in the tree. Hence, our tree would have at least 257 levels! This would require generating 2^{256} leaves, which is clearly infeasible. So suppose we use the method of randomized leaf selection. It is noted by Bernstein et al. [2] that if we take $n = 128$, then there is only a probability of about 2^{-30} of re-using a leaf node for signing, and that the scheme is likely to be broken within 2^{50} signatures. Even if this is an acceptable probability, the tree would still have 129 levels, and while this is surely an improvement over 257, it is

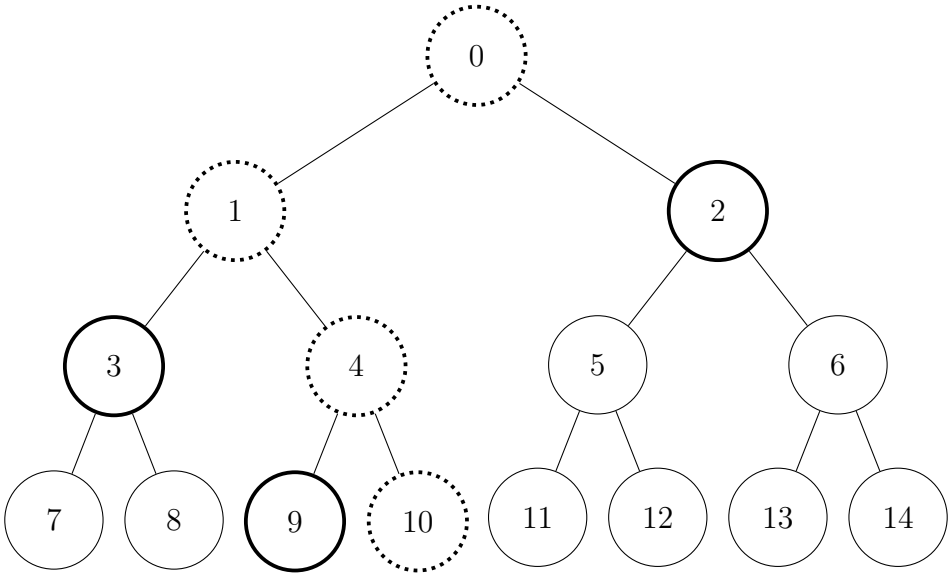

Figure 2.4: Goldreich's construction for $n = 3$, signing with leaf 10.

still infeasible to generate.

The solution is to not actually generate the entire tree. Instead, one could simply generate just the nodes that are required using the secret seed for the generator along with the indices of the required nodes.

Chapter 3

Winternitz One-Time Signature Schemes

In this chapter we introduce the *Winternitz one-time signature scheme* (WOTS) [31], and discuss its evolution and usage throughout the years. First proposed by Ralph Merkle in 1979, but not published until a decade later, this scheme was developed as an improvement to LD-OTS, and was suggested to Merkle by Robert Winternitz. These improvements allow us to substantially reduce signature sizes and gives us a way to establish a trade-off between signature generation time and storage space used. The other property of WOTS which we make extensive use of is that we can compute the verification key from any valid signature. Hence, public keys no longer need to be included in signatures, further reducing signature sizes.

In Section 3.1 we describe the most basic version of the scheme and discuss at a high level why it is secure. Next, in Section 3.2 we describe a version of the scheme that utilizes the Leighton-Micali improvements suggested in Sections 2.3.4 and 2.3.6 and give a full security proof due to Katz [22]. Then, in Section 3.3 we describe a version of WOTS which utilizes pseudorandom functions and give the corresponding security proof. However, we observe that the only currently known proof is flawed and incorrect. We include this version for its historical significance and because the ways in which the proof is flawed is of interest. Next, in Section 3.4 we describe another improved version of WOTS utilizing so-called *chaining functions* and give a high-level security argument, and discuss what these improvements actually accomplish. Finally, in Section 3.5 we discuss a version that is provably secure against multi-target attacks.

3.1 Basic WOTS

The scheme uses a security parameter $n \in \mathbb{N}$ and the *Winternitz parameter* $w \in \mathbb{N}$, which is typically a power of 2 and which determines the space-time trade-off that can be achieved. Additionally, we need a one-way function

$$f : \{0,1\}^* \to \{0,1\}^n$$

and a collision resistant hash function

$$H : \{0,1\}^* \to \{0,1\}^n.$$

For a message $M \in \{0,1\}^*$, we denote by $f^e(M)$ the e-fold iteration of f on M. In other words,

$$f^e(M) = f(f(\cdots(f(M))\cdots)),$$

where there are e applications of f. Let $f^0(\cdot)$ be the identity function. Set

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

Define a checksum $C : (\{0,1\}^w)^{\ell_1} \to \{0,1\}^{w\ell_2}$ as follows. Let d be an n-bit message digest. First compute a base-w representation of d so that each digit in d has value in $\{0,1,\ldots,w-1\}$. Next, parse d into its ℓ_1 constituent digits $b_0, b_1, \ldots, b_{\ell_1-1}$ and interpret each b_i as a nonnegative integer. Then the checksum is computed as

$$C(b_0, b_1, \ldots, b_{\ell_1-1}) = \sum_{i=0}^{\ell_1-1} (w - 1 - b_i).$$

The result is stored as an integer using *exactly* ℓ_2 bits, where we prepend 0's if necessary[1]. We claim that the base-w representation of the checksum has length at most ℓ_2. This is because $C(d) \leq \ell_1(w-1)$ for $d \in \{0,1\}^n$, and taking the base-w log of both sides yields $|C|_w \leq \log_w(\ell_1(w-1))$. Now we apply the change of base formula to the right hand side to get $\log(\ell_1(w-1))/\log(w)$. By construction, this value is bounded above by ℓ_2.

Observe that the larger the b_i are, the smaller is the checksum; this is crucial to the security proofs for such schemes. We now describe the scheme.

Key generation: The key generation algorithm takes as input the security parameter n.

[1] McGrew et al. [29] suggest a cyclic left shift of the checksum bits. The size of this shift is dependent on the parameters used; see [29] for their recommendations. In the interest of readability we do not shift the checksum in this thesis.

1. Select ℓ values $x_0, x_1, \ldots, x_{\ell-1} \xleftarrow{\$} \{0,1\}^n$.
2. For $i = 0, 1, \ldots, \ell - 1$, compute $y_i = f^{w-1}(x_i)$.
3. Compute $\mathsf{pk} = H(y_0 \| y_1 \| \ldots \| y_{\ell-1})$.

The secret key is the set of x_i and the public key is the hash value pk.

Signature generation: To sign message $M \in \{0,1\}^*$ using secret key $\{x_0, x_1, \ldots, x_{\ell-1}\}$, do as follows.

1. Compute the digest $d = H(M)$ and the checksum $c = C(d)$.
2. Let $B = d \| c$ be the concatenation of the base-w representations of d and c.
3. Parse B as a sequence of ℓ bitstrings of length w as $B = b_0 \| b_1 \| \ldots \| b_{\ell-1}$. Interpret these b_i as nonnegative integers.
4. For $i = 0, 1, \ldots, \ell - 1$, compute $\sigma_i = f^{b_i}(x_i)$.

The signature returned is $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{\ell-1})$.

Signature verification: To verify signature σ on message $M \in \{0,1\}^*$ using public key pk, do as follows.

1. Compute $d = H(M)$, $c = C(d)$, and $B = d \| c$ as in the signing algorithm.
2. Parse $B = b_0 \| b_1 \| \ldots \| b_{\ell-1}$ as in the signing algorithm.
3. For $i = 0, 1 \ldots, \ell - 1$, compute $y_i = f^{w-1-b_i}(\sigma_i)$.

The signature is accepted if and only if $\mathsf{pk} = H(y_0 \| y_1 \| \ldots \| y_{\ell-1})$.

Figure 3.1 gives a visual example of the Winternitz signature of a message with hash digest $d = 1011$ using $n = w = 4$. These parameters yield $\ell_1 = \ell_2 = 2$ and hence, $\ell = 4$. The base-w representation of d is 23, and the checksum (prepending a zero) is computed as 01. Thus, $B = 2301$ and the resulting signature is

$$\sigma = (f^2(x_0), f^3(x_1), f^0(x_2), f^1(x_3)).$$

The highlighted nodes of Figure 3.1 are the individual components of the signature.

The size of the public key is n bits, the size of the private key is ℓn bits, and a signature has size ℓn bits. We will see throughout this chapter that this version of WOTS has the smallest of these sizes. However, this version also has the strongest security assumptions which are difficult to achieve in practice.

An interesting and useful observation about WOTS is that the public key can be computed from a valid signature. We make use of this fact in Chapter 4.

We give an informal security argument for this scheme. Suppose that an adversary $\mathcal{A}$ wants to forge a WOTS signature, and that they are in possession of a valid message/signature pair (M, σ). Each component x_i in the secret key corresponds to a function "chain" $f^0(x_i), f^1(x_i), f^2(x_i), \ldots, f^{w-1}(x_i)$. As f is a one-way function, it is infeasible for $\mathcal{A}$ to recover any of the elements along these chains that occur before the σ_i which they possess, but they can efficiently produce any elements further along the chains. Let $B = b_0 \| b_1 \| \ldots \| b_{\ell-1}$ correspond to M. Suppose that $M' \neq M$ is a message whose signature $\mathcal{A}$ is trying to forge. And so $\mathcal{A}$ computes $B' = d' \| c' = b'_0 \| b'_1 \| \ldots \| b'_{\ell-1}$.

To successfully produce a signature $\mathcal{A}$ must produce $\sigma'_i = f^{b'_i}(x_i)$ for each i. Again, $\mathcal{A}$ cannot recover the elements along the chains earlier than the σ_i they got from σ. In particular they cannot compute the secret key. Thus, they can only produce a signature if $b'_i \geq b_i$ for each i. We claim that this can never happen. Consider just $d' = b'_0 \| \ldots \| b'_{\ell_1-1}$; if each of these b'_i equal their corresponding b_i then $d' = d$, and the adversary has found a collision in H. Because H is collision resistant, we can assume this is not the case. So suppose that each of these b'_i are at least equal to b_i, but with at least one $b'_i > b_i$. Then by construction of the checksum, $c' < c$ — the larger these b_i are, the smaller the checksum becomes. Finally, since $c' < c$, we have that $b'_i < b_i$ for at least one $i \in \{\ell_1, \ell_1+1, \ldots, \ell-1\}$. This gives the claim.

3.2 WOTS-LM

McGrew, Curcio and Fluhrer [29] put forth WOTS combined with the suggested improvements by Leighton and Micali from Chapter 2 as a proposed standard for hash-based digital signatures. Commissioned by the National Security Agency (NSA), Katz [22] analyzed the concrete security of this scheme in the multi-instance setting. Katz refers to this scheme as the Leighton-Micali OTS, but for the sake of consistency we refer to it as WOTS-LM. In the same paper, Katz analyzes the concrete security of the so-called full Leighton-Micali scheme described in [29], i.e., a version of the Merkle signature scheme where the underlying OTS is WOTS-LM. We discuss the full scheme in Section 4.1.

LM-OTS uses the same primitives as the vanilla version of WOTS, but with a couple of modifications. First, if i and b are nonnegative integers with $0 \leq i < 2^{8b}$, we denote by $[i]_b$ the b-byte representation of i. Furthermore, we make some modifications to the hash function H. Let s be a string, k a positive integer and u a nonnegative integer. We

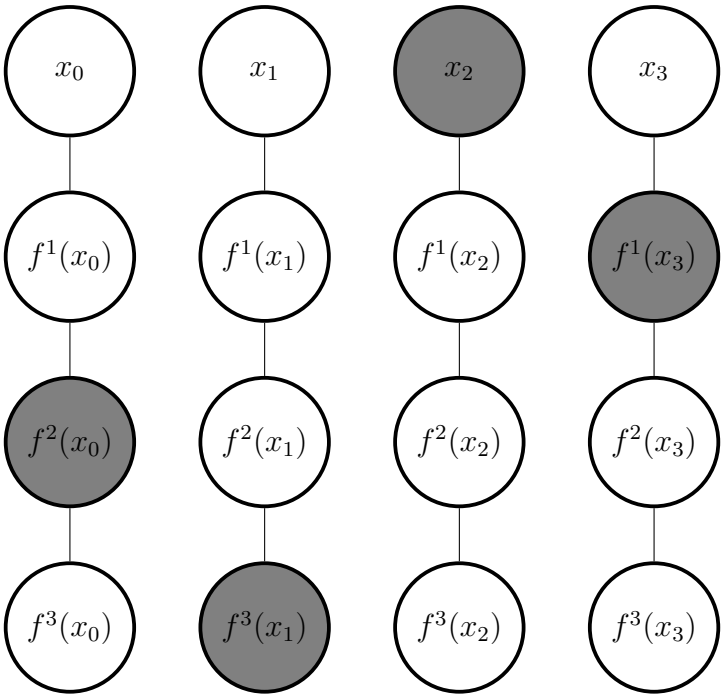

Figure 3.1: Winternitz signature for $w = n = 4$ and $d = 1011$.

recursively define

$$H_s^k(x;u) = H(H_s^{k-1}(x;u)\|s\|[u+k-1]_1\|0\text{x}00),$$

where 0xab denotes that ab is written in hexadecimal, and $H_s^0(x;u) = x$. We give the scheme's algorithms next.

Key generation: The key generation algorithm takes as input the security parameter n and a unique identity $\text{id} = (I, p)$, where I is a 64-byte identifier and p is a 4-byte instance number.

1. Select ℓ values $x_0, x_1, \ldots, x_{\ell-1} \xleftarrow{\$} \{0,1\}^n$.
2. For $j = 0, 1, \ldots, \ell - 1$, compute $y_j = H_{\text{id}\|[j]_2}^{w-1}(x_j; 0)$.
3. Compute $\mathsf{pk} = H(\text{id}\|y_0\|y_1\|\ldots\|y_{\ell-1}\|0\text{x}01)$.

The public key is pk and the secret key sk is the set of x_j.

Signature generation: To sign message $M \in \{0,1\}^*$ using secret key sk and identification $\text{id} = (I, p)$, do as follows.

1. Select $r \xleftarrow{\$} \{0,1\}^n$.
2. Compute $d = H(M\|r\|\text{id}\|0\text{x}02)$ and $c = C(d)$.
3. Let $B = d\|c$ be the concatenation of the base-w representations of d and c.
4. Parse B as a sequence of ℓ bitstrings of length w as $B = b_0\|b_1\|\ldots\|b_{\ell-1}$. Interpret these b_j as nonnegative integers.
5. For $j = 0, 1, \ldots, \ell - 1$, compute $\sigma_j = H_{\text{id}\|[j]_2}^{b_j}(x_j; 0)$.

The signature returned is $\sigma = (r, p, \sigma_0, \sigma_1, \ldots, \sigma_{\ell-1})$.

Signature verification: To verify signature σ on message $M \in \{0,1\}^*$ using identification $\text{id} = (I, p)$, do as follows.

1. Compute $d = H(M\|r\|\text{id}\|0\text{x}02)$ and $c = C(d)$.
2. Compute $B = d\|c$ and parse it as in signature generation.
3. For $j = 0, 1, \ldots, \ell - 1$, compute $y_j = H_{\text{id}\|[j]_2}^{w-1-b_j}(\sigma_j; b_j)$.

The signature verifies if and only if $H(\mathrm{id}\|y_0\|y_1\|\ldots\|y_{\ell-1}\|0\mathrm{x}01) = \mathsf{pk}$.

Sizes: The size of the public key is n bits, the size of the private key is ℓn bits, and a signature has size $\ell(n+1)+32$ bits.

Next, we give a full security proof for this scheme which very closely follows that from [22].

Theorem 3.2.1 ([22])**.** *For any adversary attacking arbitrarily many instances of WOTS-LM, making at most q hash queries of the form $H(\star\|m)$ with $m \in \{0\mathrm{x}00, 0\mathrm{x}01, 0\mathrm{x}02\}$, $\star$ arbitrary and H a random function, the probability with which they successfully forge a signature with respect to any of the instances of the scheme is at most $\frac{3q}{2^n}$.*

Proof. The proof proceeds as follows. We describe a game which we can play with the adversary, and then describe a slightly syntactically different game from the first, but which has the same probability space as the first. We play the second game with the adversary. We give an exhaustive list of events that could possibly occur during this game and then prove that if the adversary successfully produces a forgery then at least one of those events occurred. We then calculate upper bounds on the probabilities of these events and finally use a union bound to establish the above stated upper bound on the adversary's success probability.

Let t be an upper bound on the number of instances of WOTS-LM; we assume that there is a fixed set of distinct identifiers $\{\mathrm{id}^i = (I^i, p^i) \mid i = 1, 2, \ldots, t\}$. We assume for simplicity that all of these instances of WOTS-LM use the same parameters. It is not too difficult to alter the following proof if this assumption is not valid.

Game 1:

1. Select a uniformly random function $H : \{0,1\}^* \to \{0,1\}^n$.

2. For $i = 1, 2, \ldots, t$, run the key generation algorithm with input id^i to obtain $(\mathsf{pk}^i, \mathsf{sk}^i)$. Give $\{(\mathrm{id}^i, \mathsf{pk}^i) \mid i = 1, 2, \ldots, t\}$ to the adversary $\mathcal{A}$.

3. The adversary is given oracle access to H, and is given a signing oracle $\mathcal{O}_{\mathsf{Sign}}(\cdot,\cdot)$ with the property that for message $M \in \{0,1\}^*$ and index $i \in \{1, 2, \ldots, t\}$, $\mathcal{O}_{\mathsf{Sign}}(i, M)$ outputs $\mathsf{Sign}(\mathsf{sk}^i, M, \mathrm{id}^i)$, where Sign is the WOTS-LM signing algorithm. For each i, $\mathcal{A}$ may query this oracle at most once.

   We assume without loss of generality that $\mathcal{A}$ queries the signing oracle *exactly* once for each i (we could just query it for them). Additionally, we assume that whenever $\mathcal{A}$ is given a signature, they are also given all of the queries/answers to H needed to verify that signature.

4. The adversary outputs (i, M, σ), where (i, M) was not queried to the signing oracle previously. The adversary wins the game if σ is accepted by the verification algorithm as a signature for M in the i^{th} instance of the scheme. Just as in the above step, we assume without loss of generality that the attacker has all of the queries/answers to H needed for this verification.

Next we describe a second game which instantiates the first game along with the algorithms for the WOTS-LM scheme, but such that the probability space of the second game is the same as that of the first. The second game is the one which we play with the adversary.

Game 2:

1. Initialize an empty set H. The elements of H will be pairs (x, y) such that $H(x) = y$. In other words, H will contain the defined query/answer pairs for the function H from the first game.

2. For $i = 1, 2, \ldots, t$, do:
   (a) For $j = 0, 1, \ldots, \ell - 1$, select $x^i_{j,0} \xleftarrow{\$} \{0,1\}^n$.
   (b) For $j = 0, 1, \ldots, \ell - 1$ and $k = 0, 1, \ldots, w - 2$ select $x^i_{j,k+1} \xleftarrow{\$} \{0,1\}^n$ and add $(x^i_{j,k} \| \mathrm{id}^i \| [j]_2 \| [k]_1 \| 0\mathrm{x}00, x^i_{j,k+1})$ to H. Let $y^i_j = x^i_{j,w-1}$.
   (c) Select $\mathsf{pk}^i \xleftarrow{\$} \{0,1\}^n$ and add $(\mathrm{id}^i \| y^i_0 \| y^i_1 \| \ldots \| y^i_{\ell-1} \| 0\mathrm{x}01, \mathsf{pk}^i)$ to H.
   (d) Select $r^i, d^i \xleftarrow{\$} \{0,1\}^n$.
   (e) Give the adversary $(\mathrm{id}^i, \mathsf{pk}^i)$.

3. When $\mathcal{A}$ queries $H(x)$, respond as follows:
   (a) If $(x, y) \in H$ for some y, then return y.
   (b) Else, select $y \xleftarrow{\$} \{0,1\}^n$ and return y to the adversary; store (x, y) in H.

4. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Sign}}(i, M^i)$, respond as follows:
   (a) If $(M^i \| r^i \| \mathrm{id}^i \| 0\mathrm{x}02, d) \in H$ for some d, then redefine d^i by giving it the value d.
   (b) Let $c^i = C(d^i)$, set $B^i = d^i \| c^i$, and parse B^i as in the WOTS-LM signature generation algorithm.
   (c) Give the signature $(r^i, p^i, x^i_{0,b^i_0}, x^i_{1,b^i_1}, \ldots, x^i_{\ell-1,b^i_{\ell-1}})$ to the adversary.

5. The adversary outputs a triple (i, M, σ), with $M \neq M^i$. The adversary wins the game if $\mathsf{Vrfy}(M, \sigma, I^i) = \mathsf{pk}^i$. Here, Vrfy is the WOTS-LM verification algorithm.

The second game works by randomly selecting all of the input/output pairs of H including the message digests and randomness pairs (d^i, r^i). If the adversary queries for a hash value of a particular input, then we do a simple look-up in H to see if we already have a defined response for the query. If we do, then give the response to the adversary, and if we do not, then randomly select a response and give this to the attacker and store this new pair in H. When the adversary queries the signing oracle for $H(M^i \| r^i \| \mathrm{id}^i \| 0\mathrm{x}02)$, we redefine d^i if we already have a response to this stored. Otherwise, we essentially run the signature algorithm as usual. Having all of the hash query/response pairs randomized allows for the calculation of more elegant upper bounds on the probabilities of the events we define below.

We refer to the following as *collision events*. These events constitute an exhaustive list of possible occurrences in the second game that allow the adversary to produce a forgery.

- $\mathrm{Coll}_{1,i}$ is the event that $\mathcal{A}$ queries $H(I^i \| p \| y_0 \| y_1 \| \ldots \| y_{\ell-1} \| 0\mathrm{x}01)$ where $(p \| y_0 \| y_1 \| \ldots \| y_{\ell-1}) \neq (p^i \| y_0^i \| y_1^i \| \ldots \| y_{\ell-1}^i)$, but they receive the correct public key pk^i as a response.

- $\mathrm{Coll}_{2,i}$ is the event that $\mathcal{A}$ queries $H(\star \| r^i \| \mathrm{id}^i \| 0\mathrm{x}02)$ *before* the query $\mathcal{O}_{\mathsf{Sign}}(i, \star)$ is made. Observe that if this event does not occur, then we will not redefine d^i in Step 4(a) of the above experiment.

- $\mathrm{Coll}_{2,i}^*$ is the event that either $\mathrm{Coll}_{2,i}$ occurs, or either of the following events occur:

  - Before making the query $\mathcal{O}_{\mathsf{Sign}}(i, \star)$, $\mathcal{A}$ queries $H(\star \| \star \| \mathrm{id}^i \| 0\mathrm{x}02)$ and receives d^i as response, i.e., the adversary finds some message and a string which gives a collision for d^i under H before making the signing query, or
  - After making the query $\mathcal{O}_{\mathsf{Sign}}(i, M^i)$, $\mathcal{A}$ queries $H(M \| \star \| \mathrm{id}^i \| 0\mathrm{x}02)$ with $M \neq M^i$ and receives as response d^i, i.e., the adversary finds a collision for d^i sometime after they make the signing query.

- $\mathrm{Coll}_{3,i,j,k}$ is the event that $\mathcal{A}$ queries $H(x_{j,k}^i \| \mathrm{id}^i \| [j]_2 \| [k]_1 \| 0\mathrm{x}00)$ either before making the query $\mathcal{O}_{\mathrm{Sign}}(i, \star)$, or after making the query but with $k < b_j^i$. In the former case, $\mathcal{A}$ learns intermediate links in a Winternitz chain *before* they have even queried the signing oracle. In the latter case, $\mathcal{A}$ will receive a value along the Winternitz chain that occurs *before* the value they received as part of the response from the signing oracle.

- $\mathrm{Coll}_{3,i,j,k}^*$ is the event that either $\mathrm{Coll}_{3,i,j,k}$ occurs, or $\mathcal{A}$ queries $H(x \| \mathrm{id}^i \| [j]_2 \| [k]_1 \| 0\mathrm{x}00)$ with $x \neq x_{j,k}^i$ and receives the response $x_{j,k+1}^i$. In other words, the adversary finds a collision for $x_{j,k+1}^i$ under H.

The first claim we establish is that an upper bound on $\mathcal{A}$'s success probability is given by the probability that at least one of the collision events occur.

Claim 1: *If the adversary succeeds in producing a forgery, then at least one of the collision events occurred.*

Proof. Suppose that $\mathcal{A}$ outputs (i, M, σ), where σ is a valid signature on M with respect to I^i and pk^i, and with $M \neq M^i$. Recall that by assumption all of the H-queries needed to verify σ on M with respect to I^i and pk^i are defined when $\mathcal{A}$ outputs their forgery. Parse σ into its components as $\sigma = (r, p, \sigma_0, \sigma_1, \ldots, \sigma_{\ell-1})$ and set $\mathrm{id} = (I^i, p)$. Define $d = H(M\|r\|\mathrm{id}\|0\mathrm{x}02)$, and let $c = C(d)$. Run the verification algorithm on M with input id to obtain the values $d\|c = b_0\|b_1\|\ldots\|b_{\ell-1}$ and $y_j = H^{w-1-b_j}_{\mathrm{id}\|[j]_2}(\sigma_j; b_j)$. Since the adversary wins the game, we have that $H(\mathrm{id}\|y_0\|y_1\|\ldots\|y_{\ell-1}\|0\mathrm{x}01) = \mathsf{pk}^i$.

Suppose that neither $\mathrm{Coll}_{1,i}$ nor $\mathrm{Coll}^*_{2,i}$ have occurred. We show that $\mathrm{Coll}^*_{3,i,j,k}$ has occurred for some values of j and k. As $\mathrm{Coll}_{1,i}$ has not occurred, we have that

$$(p\|y_0\|y_1\|\ldots\|y_{\ell-1}) = (p^i\|y^i_0\|y^i_1\|\ldots\|y^i_{\ell-1}),$$

and thus $\mathrm{id} = \mathrm{id}^i$. As $\mathrm{Coll}^*_{2,i}$ has not occurred, d^i was not redefined during the game and the adversary has not received d^i as the response to a query in the forms described in the definition of $\mathrm{Coll}^*_{2,i}$; and as d is of this form, we can conclude that $d \neq d^i$. By the construction of the checksum we have that there exists some j such that $b_j < b^i_j$ — this was hinted at when we introduced the checksum in Section 3.1. Hence, b_j is earlier along its chain than is b^i_j. However, we also have that $y_j = y'_j$. And so, it is now clear that either $\mathrm{Coll}_{3,i,j,k}$ occurred for some k or the adversary found a collision in H in the form of event $\mathrm{Coll}^*_{3,i,j,k}$ for some k. □

Hence, to establish an upper bound on the adversary's success probability, we upper bound the probabilities of each of the collision events occurring and then take a union bound.

Claim 2: Let $q_{1,i}$ be the number of H-queries of the form $H(I^i\|\star\|0\mathrm{x}01)$. Then for all i, $Pr(\mathrm{Coll}_{1,i}) \leq q_{1,i}2^{-n}$.

Proof. Any query $H(I^i\|p\|y_0\|\ldots\|y_{p-1}\|0\mathrm{x}01)$ with $p\|y_0\|\ldots\|y_{p-1} \neq p^i\|y^i_0\|\ldots\|y^i_{p-1}$ returns a uniformly random value in $\{0,1\}^n$. Moreover, this output is independent of pk^i. Hence, for each query in this form, $\mathcal{A}$ has a $1/2^n$ probability of receiving pk^i as the response. The claim now trivially follows. □

Claim 3: Let $q_{2,i}$ be the number of H-queries of the form $H(\star \| \star \| \mathrm{id}^i \| 0\mathrm{x}02)$. Then for all i, $Pr(\mathrm{Coll}_{2,i}) \leq q_{2,i} 2^{-n}$.

Proof. In order for $\mathrm{Coll}_{2,i}$ to occur, the adversary must query $H(\star \| r^i \| \mathrm{id}^i \| 0\mathrm{x}02)$ prior to making the signing query $\mathcal{O}_{\mathsf{Sign}}(i, \star)$. However, r^i is a uniformly random value in $\{0,1\}^n$ which the adversary has no knowledge of until it queries $\mathcal{O}_{\mathsf{Sign}}(i, \star)$. The claim trivially follows. □

Claim 4: For all i, $Pr(\mathrm{Coll}^*_{2,i}) \leq 2 \cdot q_{2,i} 2^{-n}$.

Proof. We first observe that $Pr(\mathrm{Coll}^*_{2,i}) \leq Pr(\mathrm{Coll}_{2,i}) + Pr(\mathrm{Coll}^*_{2,i} | \neg \mathrm{Coll}_{2,i})$, where $\neg A$ means that event A did not occur. We apply Claim 3 to get a bound for the first term on the right hand side. As mentioned earlier, if $\mathrm{Coll}_{2,i}$ does not occur, then the value of d^i is not redefined in the game. Furthermore, every time $\mathcal{A}$ queries $H(\star \| \mathrm{id}^i \| 0\mathrm{x}02)$ before making the signing query $\mathcal{O}_{\mathsf{Sign}}(i, \star)$, and every time they query $H(M \| \star \| \mathrm{id}^i \| 0\mathrm{x}02)$ with $M \neq M^i$ after querying $\mathcal{O}_{\mathsf{Sign}}(i, M^i)$, the response they obtain is uniformly random in $\{0,1\}^n$ and is independent of d^i. Hence, the second term on the right hand side can be upper bounded by $q_{2,i} 2^{-n}$. This establishes the claim. □

Claim 5: Let $q_{3,i,j,k}$ be the number of H-queries of the form $H(\star \| \mathrm{id}^i \| [j]_2 \| [k]_1 \| 0\mathrm{x}00)$. Define $q_{3,i,j,-1} = 0$. For all i, j, k,

$$Pr\left(\mathrm{Coll}_{3,i,j,k} \mid \bigwedge_{\ell=0}^{k-1} \neg \mathrm{Coll}^*_{3,i,j,\ell}\right) \leq \frac{q_{3,i,j,k}}{2^n - q_{3,i,j,k-1}},$$

where $\bigwedge$ is the logical AND operator.

Proof. If $\mathrm{Coll}^*_{3,i,j,k-1}$ has not occurred, $\mathcal{A}$'s information about the uniformly random value $x^i_{j,k}$ is limited to the fact that $x^i_{j,k}$ was not the result of one of the attacker's previous queries of the form $H(\star \| \mathrm{id}^i \| [j]_2 \| [k-1]_1 \| 0\mathrm{x}00)$. That is of course assuming $k < b^i_j$, in the event that the adversary has already made the query $\mathcal{O}_{\mathsf{Sign}}(i, M_i)$. The claim follows. □

Claim 6: For all i, j, k,

$$Pr\left(\mathrm{Coll}^*_{3,i,j,k} \mid \bigwedge_{\ell=0}^{k-1} \neg \mathrm{Coll}^*_{3,i,j,\ell}\right) \leq \frac{q_{3,i,j,k}}{2^n - q_{3,i,j,k-1}} + \frac{q_{3,i,j,k}}{2^n}.$$

Proof. We establish this claim in a manner similar to that of Claim 4. We have the following:

$$Pr\left(\mathrm{Coll}^*_{3,i,j,k} \mid \bigwedge_{\ell=0}^{k-1} \neg\mathrm{Coll}^*_{3,i,j,\ell}\right) \leq Pr\left(\mathrm{Coll}_{3,i,j,k} \mid \bigwedge_{\ell=0}^{k-1} \neg\mathrm{Coll}^*_{3,i,j,\ell}\right) + Pr\left(\mathrm{Coll}^*_{3,i,j,k} \mid \bigwedge_{\ell=0}^{k-1} \neg\mathrm{Coll}^*_{3,i,j,\ell} \bigwedge \neg\mathrm{Coll}_{3,i,j,k}\right).$$

Claim 5 gives a bound for the first term on the right hand side. To bound the second term, we observe that if $\mathrm{Coll}_{3,i,j,k}$ does not occur, then all queries $\mathcal{A}$ makes of the form $H(\star\|\mathrm{id}^i\|[j]_2\|[k]_1\|0\mathrm{x}00)$ return a uniformly random response in $\{0,1\}^n$ which is independent of $x^i_{j,k}$. It then follows trivially that the second term is bounded by $q_{3,i,j,k}2^{-n}$. The claim follows. □

Claim 7: For all i, j, $Pr(\bigvee_{k=0}^{w-1} \mathrm{Coll}^*_{3,i,j,k}) \leq 3 \cdot \sum_{k=0}^{w-1} q_{3,i,j,k} \cdot 2^{-n}$.

Proof. Observe that if $\sum_{k=0}^{w-1} q_{3,i,j,k} \geq 2^n/2$ then the value on the right hand side is larger than 1 and the result is trivial. Otherwise we apply Claim 6 and distribute the sum to obtain

$$\begin{aligned} Pr\left(\bigvee_{k=0}^{w-1} \mathrm{Coll}^*_{3,i,j,k}\right) &\leq \sum_{k=0}^{w-1} Pr\left(\mathrm{Coll}^*_{3,i,j,k} \mid \bigwedge_{\ell=0}^{k-1} \neg\mathrm{Coll}^*_{3,i,j,\ell}\right) \\ &\leq \sum_{k=0}^{w-1} \frac{q_{3,i,j,k}}{2^n - q_{3,i,j,k-1}} + \sum_{k=0}^{w-1} \frac{q_{3,i,j,k}}{2^n}. \end{aligned} \tag{3.1}$$

Additionally, we have that $q_{3,i,j,k-1} \leq \sum_{k=0}^{w-1} q_{3,i,j,k} < 2^n/2$. Hence, $2^n/2 - q_{3,i,j,k-1} \geq 0$. Consequently, $2^n - q_{3,i,j,k-1} \geq 2^n/2$. Taking reciprocals and multiplying through by the nonnegative value $q_{3,i,j,k}$ gives

$$\frac{q_{3,i,j,k}}{2^n - q_{3,i,j,k-1}} \leq \frac{2q_{3,i,j,k}}{2^n}.$$

Substituting this into (3.1) gives the claim. □

To finally complete the proof of Theorem 3.2.1, we need to put all of these claims together. Recall that t denotes an upper bound on the number of instances of WOTS-LM. As previously mentioned, Claim 1 with a union bound gives an upper bound on the

adversary's success probability. Explicitly this upper bound is

$$\sum_{i=1}^{t} Pr(\text{Coll}_{1,i}) + \sum_{i=1}^{t} Pr(\text{Coll}^*_{2,i}) + \sum_{i=1}^{t}\sum_{j=0}^{\ell-1} Pr\left(\bigvee_{k=0}^{w-1} \text{Coll}^*_{3,i,j,k}\right).$$

By applying Claims 2, 4 and 7, we can upper bound the above expression by

$$\sum_{i=1}^{t} q_{1,i}2^{-n} + 2 \cdot \sum_{i=1}^{t} q_{2,i}2^{-n} + 3 \cdot \sum_{i=1}^{t}\sum_{j=0}^{\ell-1}\sum_{k=0}^{w-1} q_{3,i,j,k}2^{-n}.$$

Clearly we can upper bound the above by

$$3 \cdot \left(\sum_{i=1}^{t} q_{1,i} + \sum_{i=1}^{t} q_{2,i} + \sum_{i=1}^{t}\sum_{j=0}^{\ell-1}\sum_{k=0}^{w-1} q_{3,i,j,k}\right) \cdot 2^{-n}.$$

Finally, we observe that each H-query that the adversary makes of the appropriate form $H(\star\|0\text{x}00), H(\star\|0\text{x}01), H(\star\|0\text{x}02)$ increases at most one of $q_{1,i}, q_{2,i}$, or $q_{3,i,j,k}$, and thus the value in the parentheses above is bounded above by the total number q of hash queries. This completes the proof. □

In Chapter 4, we revisit the WOTS-LM signature scheme.

As with any security proof, it is important that we ask what the practical implications of Theorem 3.2.1 are. Unfortunately, Katz's proof is not a reductionist proof, and so we cannot estimate parameters from it as we would in other schemes. Moreover, the proof does not seem to inform us whether the hash function employed should be second-preimage resistant or if it should be collision resistant. However, the proof is essentially tight, and so as long as H behaves like a random function and we select a value for n such that the adversary's success probability $3q/2^n$ is acceptably small for a feasible number of hash queries q, we can be confident that no forgeries will occur.

Interestingly enough, the Internet draft by McGrew et al. [29] does not give recommendations for n or q to ensure any particular security level. All of the parameter sets they explicitly give use $n = 256$, but they do not justify this choice in terms of the security level. That said, the cost of an experiment (i.e., a game) that runs in time t with success probability at most ϵ, is given by t/ϵ. In our second game above, we can take t to be approximately the number of hash queries q, and ignoring the factor of 3, we have $\epsilon = q/2^n$. Hence, by taking the ratio of these two values, we can conclude that the cost of this game is about 2^n. It then seems appropriate to select for example $n = 128$ to achieve the 128-bit classical security level. Nonetheless, if resistance to quantum adversaries is desired, then $n = 256$ is recommended.

3.3 WOTS-PRF

This section describes a variant of the Winternitz scheme whose security is in the standard model and is based on pseudorandom functions. We call this signature scheme WOTS-PRF [5]. In Chapter 4 we describe several schemes based on the Merkle signature scheme, but with underlying one-time signature schemes being some variants of WOTS. In particular, we describe the scheme XMSS [7], a full signature scheme based on binary hash trees with underlying one-time scheme WOTS-PRF. For several years XMSS was a scheme of central focus because, its security reduction is tight, exact, in the standard model, and requires minimal security assumptions. Moreover, the techniques that the authors of [5] employed further reduced signature sizes.

Unfortunately, as we detail in this section, the security proof in [5] is flawed. We restrict ourselves to only discussing the most important and pernicious of the errors, but it is important to note (as we detail in later sections) that the erroneous security claims from [5] are used in other important works. Furthermore, slightly different versions of the results from [5] have appeared, [6, 17]. These other works attempt to correct some of the errors, but the most devastating flaws remain.

We first describe the scheme, then give the full security proof from [5], and finally discuss the errors within.

The scheme uses the following parameterized pseudorandom function family

$$\mathcal{F}_n = \{f_k : \{0,1\}^n \to \{0,1\}^n \mid k \in \{0,1\}^n\}$$

for security parameter n. Given a key k and a nonnegative integer e, we define the e-fold iteration of f_k as follows. If $e = 0$, then $f_k^0(x) = k$[2]. If $e > 0$, then

$$f_k^e(x) = f_{f_k^{e-1}(x)}(x).$$

Thus, $f_k^1(x) = f_k(x), f_k^2(x) = f_{f_k(x)}(x)$, and so on.

The scheme also uses a Winternitz parameter w, values ℓ_1, ℓ_2, ℓ, checksum C, and signs m-bit message digests[3]. Let $H : \{0,1\}^* \to \{0,1\}^m$ be a cryptographic hash function. We now describe the algorithms of WOTS-PRF.

Key generation: On input a security parameter n the algorithm proceeds as follows.

1. Select a value $x \xleftarrow{\$} \{0,1\}^n$.

[2] [5] incorrectly defines $f_k^0(x) = x$. This was later fixed in [17], but not in [6].

[3] We replace n with m in the formulation of ℓ_1 and ℓ_2.

2. Select strings $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_\ell \xleftarrow{\$} \{0,1\}^n$.

3. For $i = 1, 2, \ldots, \ell$, compute $\mathsf{pk}_i = f_{\mathsf{sk}_i}^{w-1}(x)$.

The public key is $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, where $\mathsf{pk}_0 = x$. The secret key is $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$.

Signature generation: On input a message $M \in \{0,1\}^*$ and secret key sk, the algorithm proceeds as follows.

1. Compute the m-bit message digest $d = H(M)$.

2. Express d in its base-w representation as $d = b_1 \| b_2 \| \ldots \| b_{\ell_1}$, so that each $b_i \in \{0, 1, \ldots, w-1\}$.

3. Compute the checksum $C = C(d)$, and let $B = d \| C = b_1 \| b_2 \| \ldots \| b_\ell$ be the concatenation of the base-w representations of d and C.

4. For $i = 1, 2, \ldots, \ell$, compute $\sigma_i = f_{\mathsf{sk}_i}^{b_i}(x)$.

The signature returned is $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell)$.

Signature verification: On input a message $M \in \{0,1\}^*$, signature σ and public key pk, the algorithm proceeds as follows.

1. Compute the m-bit message digest $d = H(M)$.

2. Express d in its base-w representation as $d = b_1 \| b_2 \| \ldots \| b_{\ell_1}$, so that each $b_i \in \{0, 1, \ldots, w-1\}$.

3. Compute the checksum $C = C(d)$, and let $B = d \| C = b_1 \| b_2 \| \ldots \| b_\ell$ be the concatenation of the base-w representations of d and C.

4. For $i = 1, 2, \ldots, \ell$, compute $\mathsf{pk}'_i = f_{\sigma_i}^{w-1-b_i}(\mathsf{pk}_0)$.

The signature is accepted if and only if $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$.

We give the following definitions directly from [5].

Definition 3.3.1. *A family of functions $\mathcal{F}_n$ is pseudorandom if no efficient algorithm* Dis *is able to distinguish a randomly chosen function $f_k \in \mathcal{F}_n$ from a randomly chosen function g from the set $\mathcal{G}_n$ of all functions with the same domain and range as $\mathcal{F}_n$.* Dis *gets access to an oracle* $\mathsf{Box}(\cdot)$ *implementing either f_k or g in a black box manner. The distinguisher may adaptively query* $\mathsf{Box}(\cdot)$ *as often as he likes. Finally, the distinguisher outputs* 1 *if he thinks the* Box *models f_k and* 0 *otherwise.*

*Let $\mathcal{F}_n$ be a family of pseudorandom functions and $\mathcal{G}_n = \{g : \{0,1\}^n \to \{0,1\}^n\}$ the family of all functions with domain and range $\{0,1\}^n$. We call $\mathcal{F}_n$ a (t,ϵ)-*PRF *if the advantage*

$$\mathrm{Adv}^{\mathrm{PRF}}_{\mathcal{F}_n}(\mathsf{Dis}) = \left| Pr(\mathsf{Box} \xleftarrow{\$} \mathcal{F}_n : \mathsf{Dis}^{\mathsf{Box}}(\cdot) = 1) - Pr(\mathsf{Box} \xleftarrow{\$} \mathcal{G}_n : \mathsf{Dis}^{\mathsf{Box}}(\cdot) = 1) \right|$$

of any distinguisher Dis *that runs in time t is at most ϵ.*

Definition 3.3.2. *Let $\mathcal{F}_n$ be a family of pseudorandom functions. We call $\mathcal{F}_n$ (t,ϵ)-*KOW *if the success probability*

$$\mathrm{Adv}^{\mathrm{KOW}}_{\mathcal{A}} = Pr((x,k) \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^n, y \leftarrow f_k(x), k' \leftarrow \mathcal{A}(x,y) : y = f_{k'}(x))$$

of any adversary $\mathcal{A}$ that runs in time t is at most ϵ; we also say that $\mathcal{F}_n$ is key one-way.

The key one-way (KOW) property is similar to the one-wayness property that is typically required of hash functions.

Definition 3.3.3. *We define a* key collision *to be a pair of distinct keys $(k,k') \in \{0,1\}^n \times \{0,1\}^n$ such that $f_k(x) = f_{k'}(x)$ for some $x \in \{0,1\}^n$.*

This leads to one more definition[4].

Definition 3.3.4 ([5])**.** *The upper bound κ on the number of key collisions is defined as follows: For each pair (x,k), there exist at most $\kappa - 1$ different keys $k_1, \ldots, k_{\kappa-1}$ which are also different from k, such that $f_k(x) = f_{k_i}(x)$ for $i = 1, \ldots, \kappa - 1$. Similarly we define the lower bound on the number of key collisions κ': For each pair (x,k), there exist at least $\kappa' - 1$ different keys $k_1, \ldots, k_{\kappa'-1}$ which are also different from k, such that $f_k(x) = f_{k_i}(x)$ for $i = 1, \ldots, \kappa' - 1$.*

Roughly speaking, these values of κ and κ' partition $\{0,1\}^n$ for each x into classes of functions that map x to the same image. Hence,

$$\frac{2^n}{\kappa} \leq |\{f_k(x) : k \in \{0,1\}^n\}| \leq \frac{2^n}{\kappa'}$$

for all $x \in \{0,1\}^n$. The authors of [5] write the following: "Also, given $y \xleftarrow{\$} \{0,1\}^n$ the probability that there exists a key k and a preimage x such that $f_k(x) = y$ holds is at least

[4]This definition does not quite capture what the authors intended as it was subsequently changed in [6] and again in [17]. The definition in [17] is the correct one, namely κ should be the *least* upper bound on the number of key collisions and κ' should be the *greatest* lower bound. The definition given in [6] is flawed.

$1/\kappa$". While this is certainly true, the lower bound is very weak. Observe that since x is not fixed, it is quite likely that the probability is *very* high.

We present the first erroneous claim of [5]. The following lemma describes a relation between the security level of pseudorandom functions and the upper bound κ on key collisions.

Lemma 3.3.5 ([5])**.** *Let $\mathcal{F}_n$ be a (t, ϵ)-PRF with security level $b = \log(t/\epsilon)$ and κ as in Definition 3.3.4. Then $\kappa \leq 2^{n-b} + 1$.*

We first present the proof exactly as in [5] and then discuss its flaw.

Proof. Assume $\kappa > 2^{n-b} + 1$ and let (x, y) be a pair where there exist κ keys mapping x to y. Then distinguisher Dis queries Box with x. If $\mathsf{Box}(x) = y$ then Dis returns 1 and 0 otherwise. Clearly Dis runs in time $t' = 1$. Further we have $Pr(\mathsf{Box} \xleftarrow{\$} \mathcal{F}_n : \mathsf{Dis}^{\mathsf{Box}(\cdot)} = 1) = \kappa/2^n > 2^{-b} + 2^{-n}$ and $Pr(\mathsf{Box} \xleftarrow{\$} G : \mathsf{Dis}^{\mathsf{Box}(\cdot)} = 1) = 2^{-n}$ and therefore $\epsilon' = \mathrm{Adv}^{\mathrm{PRF}}_{\mathcal{F}_n}(\mathsf{Dis}) > 2^{-b}$ which is a contradiction. □

The problem with the above proof is the use of the word "let". The distinguishing algorithm Dis takes as input a pair (x, y) for which the value κ is attained. Then, Dis constructively uses that pair to contradict the assumed security level of the PRF. The authors of [5] give no method for obtaining such a pair (x, y), but then uses such a pair as input into their algorithm Dis. A constructive algorithm with non-constructive input is not useful. To really highlight why this kind of argument is flawed, consider a cryptographic hash function H. We construct an efficient adversary $\mathcal{A}$ for finding collisions in H as follows. Let (x, x') be a pair of distinct values in the domain of H such that $H(x) = H(x')$. Give the pair (x, x') to $\mathcal{A}$. The adversary then outputs the pair (x, x'), and has hence successfully produced a collision in H. Thus we have a constructive algorithm with non-constructive input for quickly finding collisions. As we do not provide a constructive method for obtaining the input it follows that this algorithm is rather useless. We point out further consequences of this lemma later in this section.

We have the following proposition relating the KOW property to the PRF property.

Proposition 3.3.6 ([5])**.** *Let $\mathcal{F}_n$ be a (t, ϵ)-PRF. Then $\mathcal{F}_n$ is $(t - 2, \epsilon/(\kappa^{-1} - 2^{-n}))$-KOW.*

Next, we present the security reduction for WOTS-PRF from [5] and analyze its flaws.

Theorem 3.3.7 ([5]). *Let $\mathcal{F}_n$ be a family of pseudorandom functions and κ as in Definition 3.3.4. If $\mathcal{F}_n$ is a $(t_{\mathrm{PRF}}, \epsilon_{\mathrm{PRF}})$-*PRF *then WOTS-PRF is $(t, \epsilon, 1)$ EU-CMA with*

$$t = t_{\mathrm{PRF}} - t_{\mathsf{Kg}} - t_{\mathsf{Vf}} - 2$$
$$\epsilon \leq \epsilon_{\mathrm{PRF}} \ell^2 w^2 \kappa^{w-1} \frac{1}{\kappa^{-1} - 2^{-n}},$$

where t_{Kg} and t_{Vf} are the runtimes of the key generation and verification algorithms respectively.

Proof. The proof works as follows: First we use a forger for WOTS-PRF to construct an adversary on the key one-wayness of $\mathcal{F}_n$. This adversary is then used to construct a distinguisher using Proposition 3.3.6. The signing oracle Sign is simulated by the adversary.

The goal of the adversary $\mathsf{A}_{\mathrm{KOW}}$ is to produce a key k' such that $f_{k'}(x) = y$ for x, y provided as input. $\mathsf{A}_{\mathrm{KOW}}$ begins by generating a regular WOTS-PRF signature key pair and choosing random positions α (a Winternitz chain) and β (a position within that chain). Then he computes the WOTS-PRF verification key using value x. The bitstring at position α in the verification key (pk_α) is computed by inserting y at position β in the hash chain used to compute pk_α. Next, $\mathsf{A}_{\mathrm{KOW}}$ calls the forger and waits for it to ask an oracle query. The forger's query can only be answered if $b_\alpha \geq \beta$ holds because $\mathsf{A}_{\mathrm{KOW}}$ doesn't know the first β entries in the corresponding hash chain. The forgery produced by the forger is only useful to $\mathsf{A}_{\mathrm{KOW}}$ if $b'_\alpha < \beta$ holds. Only then might the bitstring σ'_α in the forged signature yield a key k' such that $y = f_{k'}(x)$ holds.

We now compute the success probability of $\mathsf{A}_{\mathrm{KOW}}$. Without loss of generality we assume that the forger queries the signing oracle. The probability of $b_\alpha \geq \beta$ is at least $(\ell w)^{-1}$. This is because of the checksum which guarantees that not all the b_i are zero simultaneously. The probability that the forger succeeds is at least ϵ by definition. This probability holds under the condition that the verification key pk computed resembles a regular verification key which is the case if there exists a key k such that $f_k^\beta(x) = y$. This happens with probability at least $1/\kappa^\beta$ according to Definition 3.3.4. The probability that $b'_\alpha < \beta$ is at least $(\ell w)^{-1}$. This is because of $M \neq M'$ and the checksum which guarantees that $b_i > b'_i$ for some $i \in \{1, \ldots, \ell\}$. The probability that $y = f_{k'}(x)$ holds is at least $1/\kappa^{w-1-\beta}$. This is because there exist κ^{w-1} keys mapping x to pk_α after $w-1$ iterations and only κ^β of these keys map x to y after β iterations[5]

In summary we have $\epsilon_{\mathrm{KOW}} \geq \epsilon/(\ell^2 w^2 \kappa^\beta \kappa^{w-1-\beta})$ and $t_{\mathrm{KOW}} = t + t_{\mathsf{Kg}} + t_{\mathsf{Vf}}$ as the time for the signature query is already taken into account at the runtime of the forger. Combining

[5] This should say *at most* κ^{w-1} and *at most* κ^β. This was corrected in [17].

this with Proposition 3.3.6 yields $\epsilon_{\text{PRF}} \geq \epsilon(\kappa^{-1} - 2^{-n})/(\ell^2 w^2 \kappa^{w-1})$ and $t_{\text{PRF}} = t + t_{\mathsf{Kg}} + t_{\mathsf{Vf}} + 2$ which concludes the proof. □

To aid in our explanation of the flaws in the proof of Theorem 3.3.7 we introduce the following definition.

Definition 3.3.8. *Let $\mathcal{F}_n = \{f_k : \{0,1\}^n \to \{0,1\}^n \mid k \in \{0,1\}^n\}$ be a pseudorandom function family. Let $x \in \{0,1\}^n$ and $\gamma \in \mathbb{N}$ be arbitrary. Then a* γ-keychain *of x is an ordered γ-tuple $(k_1, k_2, \ldots, k_\gamma)$ of n-bit keys such that $k_{i+1} = f_{k_i}(x)$ for $i = 1, 2, \ldots, \gamma - 1$.*

In other words, a γ-keychain of x can be thought of as the ordered set of keys one would use to evaluate $f^{\gamma}_{k_1}(x)$. We will make use of this definition by thinking of a γ-keychain as a length-γ vector of keys that yield a contiguous length-γ subchain of a Winternitz chain. We are now ready to discuss the flaws in the proof of Theorem 3.3.7.

Both of the flaws we discuss here pertain to the probability analysis of the reduction.

Consider first the statement: "This probability holds under the condition that the verification key pk computed resembles a regular verification key which is the case if there exists a key k such that $f^{\beta}_{k}(x) = y$.". The authors claim that "This happens with probability at least $1/\kappa^{\beta}$ according to Definition 3.3.4." This second statement is false. In the language of keychains, the first statement is equivalent to saying that the referred to probability holds if there exists at least one β-keychain of x that yields y. The number of possible keys k such that $f_k(x) = y$ is at most κ. In general, at any link in a Winternitz chain there exist at most κ keys which yield the next link. The value y is placed at the β^{th} position of the α^{th} Winternitz chain and so, as x is always the input into the PRF in these Winternitz chains, there exist at most κ^{β} distinct keys mapping x to y after β iterations, i.e., there exist at most κ^{β} β-keychains of x that yield y. Observe however, that this is merely an upper bound on the number of possible β-keychains of x that yield y. To prove that one actually exists we would need a (positive) lower bound. Unfortunately, the following argument illustrates that we cannot establish such a lower bound.

The pair (x, y) in the above reduction corresponds to a KOW challenge as per Definition 3.3.2, and thus, we are in fact guaranteed that there exists at least one key k_β such that $f_{k_\beta}(x) = y$. However, we are given no guarantee that there exists a key $k_{\beta-1}$ such that $f_{k_{\beta-1}}(x) = k_\beta$, and similarly for key $k_{\beta-\delta}$ for $\delta = 2, 3, \ldots, \beta - 1$. In other words, we can get to y from κ_β after a single iteration, but we have no promise that we can get to y from any key after exactly β iterations. Thus, we cannot conclude that with positive probability there exist any β-keychains of x that yield y.

Thus, we can conclude that the statement "This happens with probability at least $1/\kappa^{\beta}$ according to Definition 3.3.4." is indeed false and therefore the proof of Theorem 3.3.7 fails.

Unfortunately, there is still another flaw in the proof of Theorem 3.3.7 to be discussed. Consider the following statement: "The probability that $y = f_{k'}(x)$ holds is at least $1/\kappa^{w-1-\beta}$". This statement is false. Consider the tree of all $(w-1)$-keychains of x that yield pk_{α}. By the same argument as above, there exist at most $\kappa^{w-1-\beta}$ distinct keys at the β^{th} level of the tree and hence at most $\kappa^{w-1-\beta}$ distinct $(w-1-\beta)$-keychains of x that yield pk_{α} (keychains are uniquely defined by their first coordinate). Moreover, one of these keys is y. Hence, if the adversary selects from among these possible $(w-1-\beta)$-keychains *uniformly at random* then they have a probability of at least $1/\kappa^{w-1-\beta}$ of selecting the one beginning with y. However, this assumption is not valid. The adversary operates in a black box manner and we cannot assume a probability distribution for it. In particular, it is conceivable that the adversary always selects σ'_{α} so that the unique keychain from σ'_{α} never yields σ_{α}, and thus never uses y. In other words, the probability that $y = f_{k'}(x)$ holds would be zero. Hence, this argument in the proof is also incorrect. In practice people tend to use either WOTS-LM or WOTS^{+} (cf. Section 3.4) over WOTS-PRF. As such, we make no attempt to correct this proof in this work. However, we do remark that there does not seem to be a trivial or obvious fix to these problems.

Figure 3.2 gives an example tree of all $(w-1)$-keychains of x that yield pk_{α}. Nodes are labeled as tuples where the first coordinate is the index in the keychain, and the second coordinate is the key at that index. We see that out of the possible $\kappa^{w-1-\beta}$ keychains beginning at the β^{th} position, there is exactly one whose first key is y.

We have the following corollary to Theorem 3.3.7 that gives a concrete security level for WOTS-PRF.

Corollary 3.3.9 ([5])**.** *Let $b = \log(t/\epsilon)$ denote the security level and use ℓw as an upper bound for t_{Kg} and t_{Vf}, respectively. Let $\mathcal{F}_n$ be a $(2^{n-1-\log(\kappa)}, 1/(2(\kappa^{-1} - 2^{-n})))$-PRF with $\kappa = 2$. Then the security level of WOTS-PRF is*

$$b \geq n - w - 1 - 2\log(\ell w).$$

The ideal PRF family has security level $b \approx n$. For such a family, Lemma 3.3.5 concludes that $\kappa \leq 2$. As we have shown Lemma 3.3.5 to be false, then this proof is also flawed. However, we scrutinize this claim further. What does it actually *mean* that $\kappa \leq 2$? Intuition tells us that this is not a reasonable value. Consider the article [27], wherein the authors give a tight analysis for the following problem. If we sequentially throw M

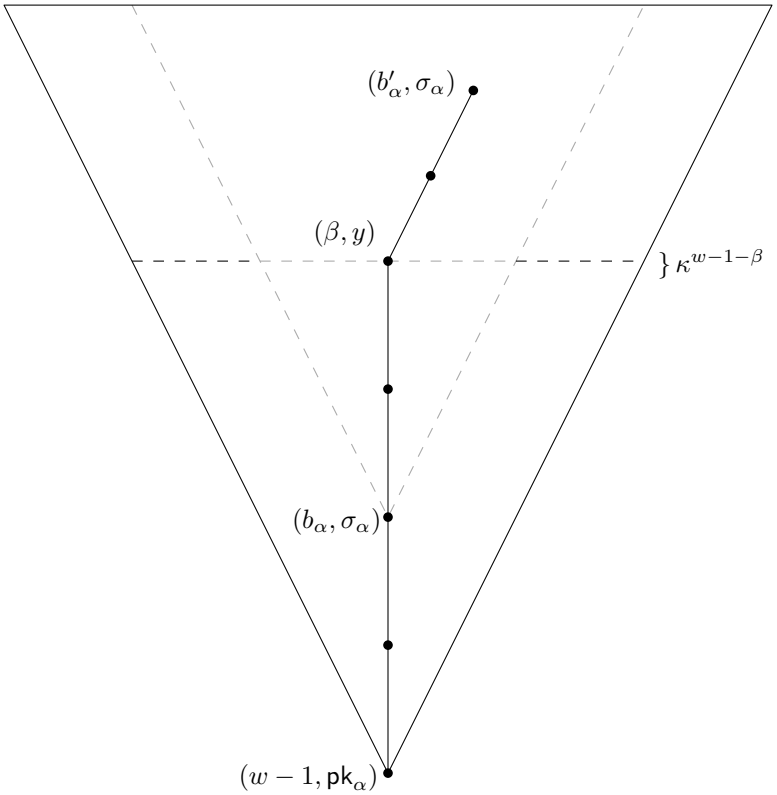

Figure 3.2: A tree of keychains.

balls into N bins independently and uniformly at random, what is the expected maximum number of balls in any particular bin? Of particular interest is [27, Section 4] where the authors study the case $M = N$. This study is analogous to the pseudorandom function family $\mathcal{F}_n$ where, for a fixed input x, the balls are the keys $k \in \{0,1\}^n$, the bins are the elements of the codomain $\{0,1\}^n$, and ball k is placed in bin $f_k(x)$. Then the expected maximum number of balls in a bin is at most κ.

Theorem 3.3.10 ([27])**.** *If N balls are sequentially placed independently and uniformly at random in N bins, then the expected maximum number of balls in a bin is $\frac{\log(N)}{\log\log(N)}(1+o(1))$. Moreover,*

$$Pr\left(\exists \text{ at least one bin with } \geq \alpha\frac{\log(N)}{\log\log(N)} \text{ balls}\right) = \begin{cases} 1-o(1) & \text{if } 0<\alpha<1, \\ o(1) & \text{if } \alpha > 1. \end{cases}$$

Clearly the value $\log(N)/\log\log(N)$ can be made arbitrarily large. Hence, for any $t \in \mathbb{N}$ we can produce values $0 < \alpha < 1$ and $N \in \mathbb{N}$ such that $\alpha\log(N)/\log\log(N) \geq t$. Thus, even though the PRF family $\mathcal{F}_n$ is not uniformly random, this gives strong evidence that $\kappa \leq 2$ is in general untrue. We would expect the maximum number of balls in any particular bin to depend in some way on the value N.

To further illustrate this point consider the following. Fix a positive integer t. Define

$$X_i = \begin{cases} 1 & \text{if bin } i \text{ has exactly } t \text{ balls}, \\ 0 & \text{otherwise.} \end{cases}$$

Then by the binomial distribution, for any i we can calculate the expectation of X_i as follows:

$$\begin{aligned} E[X_i] &= 1 \cdot \Pr(\text{bin } i \text{ has exactly } t \text{ balls}) + 0 \cdot \Pr(\text{bin } i \text{ does not have exactly } t \text{ balls}) \\ &= \Pr(\text{bin } i \text{ has exactly } t \text{ balls}) \\ &= \binom{N}{t}\left(\frac{1}{N}\right)^t\left(1-\frac{1}{N}\right)^{N-t} \\ &\approx \left(\frac{1}{t!}\right)\left(\frac{1}{e}\right). \end{aligned}$$

Now let Y be the random variable counting the number of bins with exactly t balls in them. That is to say $Y = \sum X_i$. Using linearity of expectation and the fact that the X_i are identically distributed, we can conclude that

$$E[Y] = E\left[\sum X_i\right] = \sum E[X_i] = N \cdot E[X_1] \approx \frac{N}{t!e}.$$

Clearly this value can be made arbitrarily large. In words, this means that we can expect arbitrarily many bins to contain arbitrarily many balls if we allow the number N of bins/balls to become arbitrarily large. This further enforces that $\kappa \not\leq 2$.

Consider again Corollary 3.3.9. The lower bound $n - w - 1 - 2\log(\ell w)$ on the security level of WOTS-PRF is obtained in [5, Section 2.4] by using the assumption that $\kappa \leq 2$. If one acknowledges that $\kappa \not\leq 2$, then (without giving any concrete values) the security guarantee one obtains is lower — potentially much lower — than Corollary 3.3.9 claims. This observation shows that any implementation of WOTS-PRF using parameters based on Corollary 3.3.9 may have security levels significantly lower than those thought to be guaranteed by Corollary 3.3.9, and hence may have potential security vulnerabilities.

We conclude then that WOTS-PRF should not be used in practice. Both the theoretical and concrete security are compromised.

The remainder of [5] discusses, among other things, the strong unforgeability of WOTS-PRF. We found that similar problems as the above pervade this section as well, but we do not go into any more detail.

3.4 WOTS$^+$

Perhaps the biggest critique of hash-based signature schemes is that they tend to have very large signature sizes, and indeed this is a major roadblock to making such schemes practical. The Winternitz scheme has the advantage that the public key can be computed from any valid signature, and thus, the public key need not be included in a signature. Hence, signature sizes are naturally smaller in Winternitz-style schemes. However, these sizes may still be too large to be completely practical. Here, we introduce another variant of the Winternitz scheme due to Hülsing [19] which reduces signature sizes even further; the scheme is known as WOTS$^+$.

The main difference between WOTS$^+$ and previous incarnations of the Winternitz OTS is the introduction of so-called "chaining functions". At a high level, WOTS works by applying some function iteratively to a random input, and each iteration of this function gives another link in a chain. The ends of these chains comprise the public key, the beginnings of these chains comprise the private key, and signatures are formed by mapping pieces of a message digest to pseudorandom links in these chains. In WOTS$^+$, Hülsing introduces a more sophisticated chaining function which allows for a tight security proof and a tight analysis of the security level of the scheme. Moreover, the iteration method which the author employs is one which does not require the underlying hash function to

be collision resistant. Instead, this scheme requires that the hash function used be second-preimage resistant, one way and undetectable. Undetectability is a notion we have not yet defined, but is a rather natural notion; we define it at the end of this section.

WOTS$^+$ uses a publicly known keyed function family $\mathcal{F}_n = \{f_k : \{0,1\}^n \to \{0,1\}^n \mid k \in \mathcal{K}_n\}$, where n is the security parameter and $\mathcal{K}_n$ is a key-space dependent on n. We require this family to be second-preimage resistant, one-way and undetectable. This function family is used to describe the chaining function $c_k^i(x, \mathbf{r})$ as follows. The function takes as input a bitstring $x \in \{0,1\}^n$, an iteration counter $i \in \mathbb{N}$, a key $k \in \mathcal{K}_n$, and randomization elements $\mathbf{r} = (r_1, r_2, \ldots, r_j)$ where each $r_a \in \{0,1\}^n$ and $j \geq i$. In the case where $i = 0$, i.e., where we require zero iterations, the function simply acts as the identity function: $c_k^0(x, \mathbf{r}) = x$, and in the case where $i > 0$, we recursively define the functions as

$$c_k^i(x, \mathbf{r}) = f_k(c_k^{i-1}(x, \mathbf{r}) \oplus r_i).$$

In words, in every iteration the function returns f_k evaluated on an intermediate link in the chain which has been **xor**'ed with the bitmask r_i.

To compress notation, we denote by $\mathbf{r}_{a,b}$ the substring $(r_a, \ldots, r_b)$ of $\mathbf{r}$. If $a > b$, then define this to be the empty string. We can now describe WOTS$^+$.

First, select the Winternitz parameter w, the bitlength m of digests to be signed, and compute

$$\ell_1 = \left\lceil \frac{m}{\log(w)} \right\rceil, \quad \ell_2 = \left\lfloor \frac{\log(\ell_1(w-1))}{\log(w)} \right\rfloor + 1, \quad \ell = \ell_1 + \ell_2.$$

The checksum is the same as before. Again, we assume that $H : \{0,1\}^* \to \{0,1\}^m$ is a cryptographic hash function.

Key generation: The key generation algorithm takes as input the security parameter n.

1. Randomly select $\ell + w - 1$ strings from $\{0,1\}^n$. The secret key $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_\ell)$ will consist of the first ℓ of these strings, and the remaining strings become the randomization elements $\mathbf{r} = (r_1, r_2, \ldots, r_{w-1})$.

2. Randomly select a key $k \in \mathcal{K}_n$.

3. The public key is computed as $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, where $\mathsf{pk}_0 = (\mathbf{r}, k)$, and $\mathsf{pk}_i = c_k^{w-1}(\mathsf{sk}_i, \mathbf{r})$ for $i > 0$.

The reader may notice that the public keys in this version of WOTS are larger than in previous incarnations because of the inclusion of the randomization elements. This is unfortunate, but allows for a tight, standard model reduction.

Signature generation: To compute a signature on a message $M \in \{0,1\}^*$ using secret key sk and with randomization elements $\mathbf{r}$, the signer does the following:

1. Compute the m-bit digest $d = H(M)$.
2. Express d in its base-w representation as $d = b_1 \| b_2 \| \ldots \| b_{\ell_1}$, so that each $b_i \in \{0, 1, \ldots, w-1\}$.
3. Compute the checksum $C = C(d)$, and let $B = d \| C = b_1 \| b_2 \| \ldots \| b_\ell$ be the concatenation of the base-w representations of d and C.
4. For $i = 1, 2, \ldots, \ell$, compute $\sigma_i = c_k^{b_i}(\mathsf{sk}_i, \mathbf{r})$.

The signature returned is $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell)$.

Signature verification: To verify a signature σ as above on a message $M \in \{0,1\}^*$ using public key pk do the following:

1. Compute the m-bit digest $d = H(M)$.
2. Compute $B = d \| C$ as in signature generation.
3. Check if

$$\mathsf{pk} = ((\mathbf{r}, k), c_k^{w-1-b_1}(\sigma_1, \mathbf{r}_{b_1+1,w-1}), \ldots, c_k^{w-1-b_\ell}(\sigma_\ell, \mathbf{r}_{b_\ell+1,w-1})).$$

The signature is accepted if and only if the above comparison holds.

Sizes: The size of the public key is $(\ell + w - 1)n + |k|$ bits, the size of the private key is ℓn bits, and a signature has size ℓn bits. Here, we use $|k|$ to denote the number of bits to represent an arbitrary member of $\mathcal{K}_n$.

Even though we have not given the security reduction for this scheme, for completeness we define the security notion of undetectability. Intuitively, one can think of undetectability as "you cannot detect if a hash function has been applied". Suppose we have two distributions over $\{0,1\}^n \times \mathcal{K}$ as follows. The first, $\mathcal{D}_{\mathrm{UD},\mathcal{U}}$ is obtained by randomly selecting a bitstring $u \xleftarrow{\$} \{0,1\}^n$ ($\mathcal{U}$ represents the uniform distribution) and a key $k \xleftarrow{\$} \mathcal{K}$; the sample returned is (u, k). The second distribution $\mathcal{D}_{\mathrm{UD},\mathcal{F}_n}$ is obtained by randomly selecting a key $k \xleftarrow{\$} \mathcal{K}$ and a random bitstring $u \xleftarrow{\$} \{0,1\}^n$, and returning the sample $(k, f_k(u))$. The adversary is challenged with distinguishing between these two samples. A function family $\mathcal{F}_n$ is said to be *undetectable* if the advantage of any possible probabilistic

adversary in distinguishing between such samples in time at most t is negligible in n. In other words, if

$$\mathrm{InSec}^{\mathrm{UD}}(\mathcal{F}_n; t) := \max_{\mathcal{A}} \left\{ \mathrm{Adv}^{\mathrm{UD}}_{\mathcal{F}_n}(\mathcal{A}) \right\} = \mathsf{negl}(n).$$

We have the following theorem from [19].

Theorem 3.4.1. *Let $n, w, m \in \mathbb{N}$ with $w, m = \mathrm{poly}(n)$, and let $\mathcal{F}_n = \{f_k : \{0,1\}^n \to \{0,1\}^n \mid k \in \mathcal{K}_n\}$ be a second preimage resistant, undetectable, one-way function family. Then, $\mathrm{InSec}^{\mathrm{EU-CMA}}(\mathrm{WOTS}^+(1^n, w, m); t, 1)$, the insecurity of WOTS$^+$ against an EU-CMA attack, is bounded above by*

$$w \cdot \mathrm{InSec}^{\mathrm{UD}}(\mathcal{F}_n; t'') + \max\{\mathrm{InSec}^{\mathrm{OW}}(\mathcal{F}_n; t'), w \cdot \mathrm{InSec}^{\mathrm{SPR}}(\mathcal{F}_n; t')\},$$

where $t' = t + 3\ell w$ and $t'' = t + 3\ell w + w - 1$.

It is interesting to note (as the authors of [19] indeed do) that if it can be shown that there exists a function family which combines all three of the above desired properties, then it would be the case that WOTS$^+$ has minimal security requirements.

3.5 WOTS-T

WOTS$^+$, WOTS-PRF, and the schemes (that we introduce in the next chapter) that use these OTS schemes as building blocks suffer from the drawbacks mentioned in Section 2.3.6: vulnerability to multi-target attacks. In that section, we mentioned that if an adversary has more targets to attack then the probability of them launching a successful attack on at least one of these targets is higher than the probability of succeeding against exactly one. When one considers these sorts of attacks they see a linear drop in security (in some parameters of the scheme).

There is another variation of WOTS by Hülsing, Rijneveld and Song [21] which is essentially an improved version of WOTS$^+$ that has *tight* security against multi-target attacks; this scheme is refered to as WOTS-T. Essentially, the major change from WOTS$^+$ to WOTS-T is that a new, pseudorandom function key or bitmask is generated every time a function call needs such an input. This change is at the core of what prevents multi-target attacks.

WOTS-T uses an addressing scheme which assigns to each part of a data structure a unique address. We assume the existence of a function $\textsc{GenAddr}(\mathbf{a}_s, \mathit{index})$ that takes

as input the address of a data structure (for example a Winternitz chain) and an index of a substructure (such as an index within a chain) and outputs a unique address for this substructure. The values w, ℓ_1, ℓ_2 and ℓ are generated as before, and the checksum also remains the same as in previous versions of WOTS. Let $H : \{0,1\}^* \to \{0,1\}^m$ be a cryptographic hash function. Finally we also utilize the following functions: a cryptographic hash function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, and two pseudorandom function families $\mathcal{F}_m : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^m$ and $\mathcal{F}_n : \{0,1\}^n \times \{0,1\}^a \to \{0,1\}^n$ where m is the length of message digests to be signed and where a is the length of addresses.

Next, we describe the chaining function used for this incarnation of the signature scheme. This function takes as input a value $x \in \{0,1\}^n$, an iteration counter $i \in \mathbb{N}$, a start index $j \in \mathbb{N}$, a Winternitz chain address $\mathbf{a}_\mathcal{C}$, and a public seed $\textsc{Seed}$. The function is defined recursively as follows:

$$c^{i,j}(x, \mathbf{a}_\mathcal{C}, \textsc{Seed}) = F(k_{i,j}, c^{i-1,j}(x, \mathbf{a}_\mathcal{C}, \textsc{Seed}) \oplus r_{i,j})$$

if $i > 0$ and returns x otherwise. Here, key $k_{i,j} = \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_\mathcal{C}, 2 \cdot (j+i)))$ and bitmask $r_{i,j} = \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_\mathcal{C}, 2 \cdot (j+i)+1))$. In words, this function takes the **xor** of the previous output with a new bitmask $r_{i,j}$ and then evaluates the function F using key $k_{i,j}$ on this result.

We now have all we need to describe the algorithms of WOTS-T. We remark that this scheme is designed to be implemented within a larger structure similar to the Merkle tree construction described in Section 2.3.3. As such, each key generation for WOTS-T also takes as input an address for itself as an OTS key pair within a binary hash-tree; we denote this as $\mathbf{a}_{\text{OTS}}$.

Key generation: On input of secret seed $\mathcal{S} \in \{0,1\}^n$, address $\mathbf{a}_{\text{OTS}}$ and public seed $\textsc{Seed}$, the key generation algorithm proceeds as follows.

1. For $i = 1, 2, \ldots, \ell$, compute $\mathsf{sk}_i = \mathcal{F}_n(\mathcal{S}, \textsc{GenAddr}(\mathbf{a}_{\text{OTS}}, i))$.
2. For $i = 1, 2, \ldots, \ell$, compute $\mathsf{pk}_i = c^{w-1,0}(\mathsf{sk}_i, \mathbf{a}_{\mathcal{C}_i}, \textsc{Seed})$ where $\mathbf{a}_{\mathcal{C}_i} = \textsc{GenAddr}(\mathbf{a}_{\text{OTS}}, i)$.

The secret key is $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_\ell)$ and the public key is $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_\ell)$.

Signature generation: On input of message $M \in \{0,1\}^*$, secret seed $\mathcal{S}$, address $\mathbf{a}_{\text{OTS}}$ and public seed $\textsc{Seed}$ the signature algorithm proceeds as follows.

1. Compute the message digest $d = H(M)$.
2. Compute the checksum $C = C(d)$, and set $B = d \| C = b_1 \| b_2 \| \ldots \| b_\ell$.
3. Generate the secret key sk as during key generation.

4. For $i = 1, 2, \ldots, \ell$, compute $\sigma_i = c^{b_i,0}(\mathsf{sk}_i, \mathbf{a}_{\mathcal{C}_\mathbf{i}}, \textsc{Seed})$.

The signature produced is $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell)$.

Signature verification: On input of message $M \in \{0,1\}^*$, signature σ, address $\mathbf{a}_{\text{OTS}}$ and public seed $\textsc{Seed}$, the verification algorithm proceeds as follows.

1. Compute the b_i as in signature generation.
2. For $i = 1, 2, \ldots, \ell$, compute $\mathsf{pk}'_i = c^{w-1-b_i,0}(\sigma_i, \mathbf{a}_{\mathcal{C}_\mathbf{i}}, \textsc{Seed})$.

Technically this algorithm should accept the signature if and only if

$$\mathsf{pk}' = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) = \mathsf{pk}.$$

However, WOTS-T is designed to be implemented within the full signature scheme of Section 4.5. Hence, this comparison occurs within that scheme's verification algorithm.

Sizes: Public and private keys are $n\ell$ bits, but we also need a global n-bit secret key, and an n-bit public seed. Signatures are also $n\ell$ bits.

Chapter 4

Merkle Signature Schemes

Recall the Merkle signature scheme from Section 2.3.3. This scheme will serve as a template for the schemes discussed throughout this chapter. In this chapter we explore *full Merkle-signatures*— signature schemes based on the Merkle signature scheme that employ the one-time schemes discussed in Chapter 3. These schemes are able to sign many messages, and as we will show in some of these schemes a single key pair can be used to sign a "virtually unlimited number of messages". We begin by discussing one of the most popular such schemes in Section 4.1 and giving a full security analysis due to Katz [22]. This scheme is particularly appealing due to its relative simplicity when compared to the schemes that follow.

In Section 4.2 we introduce XMSS, the first in a family of signature schemes and discuss its historical significance and security analysis. In Sections 4.3, 4.4 and 4.5 we introduce and discuss the XMSS$^+$, XMSS-MT and XMSS-T members of the XMSS family and the improvements each of them offer over the original. Finally in Section 4.6 we discuss a stateless hash-based signature scheme due to Bernstein et al. [2].

At this point we make an important note. Each of the one-time schemes from Chapter 3 are complete signature schemes by themselves, i.e., they include verification algorithms. In this chapter, when we implement these schemes into larger structures, it is often the case that we do not need to run the OTS verification algorithms as subroutines in the full verification algorithms. This is because we compute Winternitz OTS public keys from their signatures and then use these keys to construct binary hash trees. Often one can prove that assuming an OTS scheme is secure, then simply computing a public key candidate from a signature and subsequently verifying the authenticity of that candidate is sufficient to ensure the security of the full scheme. This saves some computational resources. Hence,

in this chapter when we say that we run an OTS verification algorithm, it may be the case that we simply mean that we calculate the public key from the signature. This distinction should be clear from context.

4.1 The Leighton-Micali Signature Scheme

We alluded to this scheme in Section 3.2. We present this scheme first because it is perhaps the easiest to understand among the schemes discussed in this chapter, and thus provides a good entry point into the world of full Merkle Signatures. At a high level, the LMS scheme is just the Merkle scheme where the underlying OTS is LM-OTS (which we called WOTS-LM) from Section 3.2. We describe the scheme below, and then give a security proof due to Katz [22].

We deviate slightly from the method of ordering tree nodes described in Section 2.3.3. In this tree, the root node (at level h) will be denoted $T[1]$, the leftmost node on level $h-1$ is denoted $T[2]$, and the next is $T[3]$, and so on in the natural order. As in Section 3.2, if i and b are nonnegative integers with $0 \leq i < 2^{8b}$, we denote by $[i]_b$ the b-byte representation of i.

Key generation: The key generation algorithm takes as input a 64-byte identifier I and the tree height h. For convenience, set $N = 2^h - 1$.

1. For $i = 0, 1, \ldots, N$, run the WOTS-LM key generation algorithm to produce key-pairs $(\mathsf{pk}^i, \mathsf{sk}^i)$.

2. For $j = 2^h, 2^h + 1, \ldots, 2^{h+1} - 1$, set $T[j] = H(\mathsf{pk}^{j-2^h} \| I \| [j]_4 \| 0\text{x}03)$. These are the leaf nodes.

3. For $j = 2^h - 1, 2^h - 2, \ldots, 1$, set $T[j] = H(T[2j] \| T[2j+1] \| I \| [j]_4 \| 0\text{x}04)$. These are the internal nodes.

The public key contains the tree height h, the identifier I and the root node $T = T[1]$: $\mathsf{pk} = (h, I, T)$. The secret key is the collection of WOTS-LM secret keys: $\mathsf{sk} = (0, \mathsf{sk}^0, \mathsf{sk}^1, \ldots, \mathsf{sk}^N)$. The first coordinate of sk is the current state of the system. After the first signature is generated, the state is updated to 1, then 2, etc, in the natural order. If $s = N + 1$, then all leaf nodes have been used to sign, and the key is discarded and no more signatures can be generated; we refer to the key pair as *exhausted* at this point.

Signature generation: To sign a message $M \in \{0,1\}^*$ using secret key sk, where the current state is s (using the s^{th} leaf node), proceed as follows. Set $\mathrm{id} = (I, s)$.

1. Compute the WOTS-LM signature $\sigma = \mathsf{Sign}(\mathsf{sk}^s, M, \mathrm{id})$.
2. Compute the authentication path $\mathsf{Auth}_s = (a_0, a_1, \ldots, a_{h-1})$ as described in Section 2.3.3.
3. Increment s by 1.

The signature returned is $\Sigma = (\sigma, \mathsf{Auth}_s)$.

Signature verification: To verify a signature $\Sigma = (\sigma, \mathsf{Auth}_s)$ on a message $M \in \{0,1\}^*$ using public key (h, I, T) proceed as follows.

1. Run the WOTS-LM verification algorithm on σ and M to produce pk'.
2. Extract the state s from σ, and compute the leaf node $T[s + 2^h] = H(\mathsf{pk}' \| I \| s + 2^h \| 0\text{x}03)$.
3. Using the above and the authentication path, compute the root node $T[1]$.

The signature is accepted if and only if $T[1] = T$.

We now discuss the security of this scheme. It was noted erroneously by Katz [22, Section 3.4] that the security of this scheme can be proven "generically based on any one-time signature scheme and any second-preimage resistant hash function". Intuitively his statement makes sense, but when one considers the details it falls apart. A proper second-preimage challenge must be uniformly random, but since the hash function used in the scheme is not a random oracle, the subsequent inputs into the hash function (when constructing the tree) are not uniformly random; i.e., the hash values associated with the nodes of the Merkle tree are not uniform random values since the hash function itself is not a random function. Hence, we do not have valid second-preimage challenges, and second-preimage resistance is not strong enough to prove the security of LMS. Instead, we establish the security of the scheme in the Random Oracle Model.

The following security proof is done in the single-instance setting (for simplicity), but it is not too difficult to show that provided each instance uses a unique identifier I, then in the multi-instance setting security does not degrade. We present a security proof for this scheme that very closely follows Katz's proof from [22].

Theorem 4.1.1 ([22])**.** *For an adversary attacking the LMS scheme and making at most q hash queries and with H a random function, the probability that they successfully forge a signature is at most $\frac{3q}{2^n}$.*

Proof. We bound the adversary $\mathcal{A}$'s success probability in the following game:

1. Let $H : \{0,1\}^* \to \{0,1\}^n$ be a random function.

2. Run the key generation algorithm for LMS using identifier I and tree height h to obtain key pair (PK, SK). The adversary is given PK.

3. The adversary is given oracle access to H and a stateful signing oracle $\mathcal{O}^{\text{LMS}}_{\mathsf{Sign}}(\cdot)$ which on input $M \in \{0,1\}^*$ returns a valid LMS signature on M using SK and I, and which updates the private key.

   We assume without loss of generality that when the adversary is given a signature, they are also supplied with all the H-queries needed to verify that signature.

4. The attacker outputs (M, Σ), where M was not previously queried to the signing oracle. The adversary wins the game if Σ is a valid LMS signature on M under pk. Similarly to the previous step, we assume that the adversary has all the H-queries needed to run the verification algorithm on these inputs.

We can assume the signature output by the adversary is of the form $\Sigma = (\sigma, \mathsf{Auth}_s)$, where Auth_s is an authentication path. Moreover, we may assume that σ is in the format of a WOTS-LM signature; for if either of these is false then the signature output by the adversary is trivially invalid. Let s be the state stored in σ and let pk be the WOTS-LM key produced by running the WOTS-LM verification algorithm on σ. We define two *forge events*.

- Forge_1 is the event that the adversary succeeds and $\mathsf{pk} = \mathsf{pk}^s$.
- Forge_2 is the event that the adversary succeeds and $\mathsf{pk} \neq \mathsf{pk}^s$.

If Forge_1 occurs, then $\mathcal{A}$ must have forged a signature for WOTS-LM, and if Forge_2 occurs, then $\mathcal{A}$ must have found a collision in H for some internal node of the Merkle tree. The probability then that the adversary is successful in producing a forgery is clearly exactly equal to $Pr(\text{Forge}_1) + Pr(\text{Forge}_2)$. Hence, to establish the claimed upper bound on $\mathcal{A}$'s success probability, we upper bound the probabilities of these forge events occurring. We establish these upper bounds via the following two claims.

Claim 1: Let q_1 be the number of H-queries of the form $H(\star, m)$ with $m \in \{0\text{x}00, 0\text{x}01, 0\text{x}02\}$. Then, $Pr(\text{Forge}_1) \leq 3q_1 \cdot 2^{-n}$.

Proof. From $\mathcal{A}$ we construct a second adversary $\mathcal{A}'$ who attacks the underlying WOTS-LM instances. Let $\text{id}^s = (I, s)$ for $s = 0, 1, \ldots, N = 2^h - 1$. The adversary $\mathcal{A}'$ is given the public keys pk^i for $i = 0, 1, \ldots, N$ and proceeds as follows.

1. Compute the root node $T[1]$ as in the key generation algorithm using the set of WOTS-LM public keys. Give the master public key $\mathsf{PK} = (h, I, T[1])$ to $\mathcal{A}$.

2. When $\mathcal{A}$ queries for the signature of a message M_i for $i = 0, 1, \ldots, N$, $\mathcal{A}'$ queries its signing oracle $\mathcal{O}_{\mathsf{Sign}}^{\text{WOTS-LM}}(i, M_i)$ to obtain the signature σ_i. It then computes the relevant authentication path $\mathsf{Auth}_i = a_0, a_1, \ldots, a_{h-1}$ and gives the full signature $(\sigma_i, \mathsf{Auth}_i)$ to $\mathcal{A}$.

3. $\mathcal{A}'$ answers the H-queries of $\mathcal{A}$ by forwarding them to $\mathcal{A}'$'s own H-oracle and then giving $\mathcal{A}$ that response.

4. When $\mathcal{A}$ outputs a forgery $(M, (\sigma, \mathsf{Auth}))$, $\mathcal{A}'$ extracts the state s contained in σ and outputs (s, M, σ) as its own forgery.

If Forge_1 has occurred, then $\mathcal{A}'$ is successful in producing a forgery. Now, $\mathcal{A}'$ makes some H-queries in addition to those made by $\mathcal{A}$; these additional H-queries are of the form $H(\star, m)$, with $m \in \{0\text{x}03, 0\text{x}04\}$. However, the number of H-queries of the form $H(\star, m)$ with $m \in \{0\text{x}00, 0\text{x}01, 0\text{x}02\}$ made by both adversaries is exactly the same. Now we can apply Theorem 3.2.1 to establish the claim. □

Claim 2: Let q_2 be the number of H-queries of the form $H(\star, m)$ with $m \in \{0\text{x}03, 0\text{x}04\}$. Then, $Pr(\text{Forge}_2) \leq q_2 \cdot 2^{-n}$.

Proof. In order to establish this claim, we need to introduce two more collision events. Note that these are different events from those in Section 3.2.

- Coll_j for $j = 2^h, 2^h + 1, \ldots, 2^{h+1} - 1$, is the event that $\mathcal{A}$ queries $H(\mathsf{pk} \| I \| [j]_4 \| 0\text{x}03)$ with $\mathsf{pk} \neq \mathsf{pk}^{j-2^h}$ and receives $T[j]$ as a response. In other words, this is the event that $\mathcal{A}$ finds a collision for leaf node $T[j]$.

- Coll_j for $j = 1, 2, \ldots, 2^h - 1$, is the event that $\mathcal{A}$ queries $H(T \| T' \| I \| [j]_4 \| 0\text{x}04)$ with $(T, T') \neq (T[2j], T[2j+1])$ and receives $T[j]$ as a response. In other words, this is the event that $\mathcal{A}$ finds a collision for the parent node of $T[2j]$ and $T[2j+1]$.

Clearly if Forge_2 occurs, then $\mathcal{A}$ found a collision for a leaf node or for an internal node of the Merkle tree. Thus, if Forge_2 occurs, then Coll_j occurs for some j. Thus,

$$Pr(\text{Forge}_2) \leq \sum_j Pr(\text{Coll}_j).$$

Let q'_j denote the number of H-queries of the form $H(\star \| I \| [j]_4 \| \star)$. As H is a random oracle with range $\{0,1\}^n$ it follows that $Pr(\text{Coll}_j) \leq q'_j \cdot 2^{-n}$. Thus,

$$Pr(\text{Forge}_2) \leq \sum_j q'_j 2^{-n}.$$

Now, each of the adversary's queries of the form $H(\star, m)$ with $m \in \{0\text{x}03, 0\text{x}04\}$ increases the value of at most one q'_j. Hence,

$$\sum_j q'_j \leq q_2.$$

The claim follows. □

To conclude the proof of Theorem 4.1.1 we combine the above two claims with our earlier observation about the adversary's success probability as follows.

$$\begin{aligned} Pr(\mathcal{A} \text{ is successful}) &\leq Pr(\text{Forge}_1) + Pr(\text{Forge}_2) \\ &\leq 3q_1 \cdot 2^{-n} + q_2 \cdot 2^{-n} \\ &= (3q_1 + q_2) \cdot 2^{-n}. \end{aligned}$$

Now, each H-query increases at most one of q_1 or q_2, and so $q_1 + q_2 \leq q$. In particular,

$$3q_1 + q_2 \leq 3q_1 + 3q_2 \leq 3q.$$

The result now follows. □

4.1.1 Hierarchical LMS

The Internet draft [29] also specifies a hyper tree construction (cf. Section 2.3.7) implementing the LMS scheme. This construction is fairly natural and straightforward. As such, we omit the details here, but refer the reader instead to [29]. However, we do mention that it is important to ensure that users wishing to use only single-tree LMS still have interoperability and compatibility with systems utilizing the hierarchical (hyper tree) design.

4.2 XMSS

Recall SPR-MSS [12] described in Section 2.3.4. In this section we introduce the eXtended Merkle Signature Scheme (XMSS) [7], an important variant of the Merkle Signature Scheme based also on [8]. At the same security level as SPR-MSS, XMSS signatures are more than 25% smaller. Moreover, XMSS has private keys consisting only of a seed and the index of the last signature. XMSS uses the WOTS-PRF scheme from Section 3.3 as its underlying OTS. Before we describe the algorithms for this signature scheme, we describe all of the required primitives and public parameters.

Select the security parameter n, the Winternitz parameter w, and the bitlength m of message digests to be signed. Compute ℓ_1, ℓ_2 and ℓ as usual. The checksum C and keyed function family $\mathcal{F}_n$ are the same as in Section 3.3. XMSS also uses a keyed hash function family

$$\mathcal{H}_n = \{H_k : \{0,1\}^{2n} \to \{0,1\}^n \mid k \in \{0,1\}^n\}.$$

Finally, we assume that we have a pseudorandom generator GEN such that for $\lambda \in \mathbb{N}$ and $\mu \in \{0,1\}^n$,

$$\mathrm{GEN}_\lambda(\mu) = f_\mu(1) \| f_\mu(2) \| \ldots \| f_\mu(\lambda).$$

The leaves of the tree are on level 0 and the root node is on level h. The i^{th} node on level j (in the natural order) is denoted $v_j[i]$. The tree construction is almost exactly the same as in Section 2.3.4.

Key generation: On input of security parameter n and tree height h, key generation proceeds as follows.

1. Select a seed $S \xleftarrow{\$} \{0,1\}^n$.
2. Select a hash function $H_k \xleftarrow{\$} \mathcal{H}_n$.
3. For $j = 1, 2, \ldots, h + \lceil \log(\ell) \rceil$, uniformly at random select a bitmask $Q_j \in \{0,1\}^{2n}$.
4. For $i = 1, 2, \ldots, 2^h$, the i^{th} WOTS secret key is generated as $\mathsf{sk}^i = \mathrm{GEN}_\ell(f_S(i))$.
5. To construct the leaf nodes, we first construct the L-trees as described in Section 2.3.4 using hash function H_k and bitmask Q_j on level $j = h+1, h+2, \ldots, h+\lceil \log(\ell) \rceil$. The leaf nodes are the root nodes of these L-trees, also known as the *compressed* public keys of the underlying OTS. Note that we do not include the height of the L-trees in the total tree height h.
6. Compute the non-leaf nodes of the tree as $v_j[i] = H_k(Q_j \oplus (v_{j-1}[2i] \| v_{j-1}[2i+1]))$.

The XMSS secret key is $\mathsf{SK} = (S, 0)$, the second coordinate being the state that is updated as signatures are produced. When $i = 2^h$, no more signatures are generated and new XMSS keys are used. The (master) public key of the XMSS scheme is given as $\mathsf{PK} = (v_h[0], Q_1, Q_2, \ldots, Q_{h+\lceil \log(\ell) \rceil})$.

Signature generation: To sign the s^{th} message $M \in \{0,1\}^*$ do as follows.

1. Compute a signature σ on M by running the WOTS-PRF signature algorithm using the s^{th} WOTS key pair.
2. Compute the authentication path $\mathsf{Auth}_s = (a_0, a_1, \ldots, a_h)$ from $v_0[s]$ to $v_h[0]$.

The signature returned is $\Sigma = (s, \sigma, \mathsf{Auth}_s)$.

Signature verification: To verify signature Σ on message M as above, do as follows.

1. Run the WOTS-PRF verification algorithm on (σ, M) to obtain the s^{th} one-time public key $\mathsf{pk}^s = (\mathsf{pk}^s_1, \mathsf{pk}^s_2, \ldots, \mathsf{pk}^s_\ell)$.
2. Construct and compress the L-tree corresponding to pk^s to obtain the leaf node $v'_0[s]$.
3. Using $v'_0[s]$ and Auth_s, compute the path $p_0, p_1, \ldots, p_h$ to the root node of the tree.

The signature is accepted if and only if $p_h = v_h[0]$.

Sizes: The XMSS public key has size $(2(h + \lceil \log(\ell) \rceil) + 1)n$ bits; secret keys are no longer than $2n$ bits, and signatures have length $(\ell + h)n$ bits.

We have the following "result".

Theorem 4.2.1 ([7])**.** *If $\mathcal{H}_n$ is a second-preimage resistant hash function family and $\mathcal{F}_n$ is a pseudorandom function family, then XMSS is existentially unforgeable under chosen message attacks.*

The proof of this theorem in [7] employs the flawed Theorem 3.3.7 from Section 3.3; hence, this result too is false. It is not known to us if there exists an alternative proof technique. Furthermore, in the same paper, the authors proved that XMSS is forward secure provided the key generation algorithm is modified slightly. This result also uses the flawed Theorem 3.3.7.

It is important to mention that the only flaw known to us in the proof of Theorem 4.2.1 is the use of Theorem 3.3.7. We believe that XMSS itself is provably secure if a provably secure underlying OTS is used in place of WOTS-PRF.

4.3 XMSS$^+$

We introduce the signature scheme XMSS$^+$ [18], a scheme based on the XMSS scheme discussed in Section 4.2. In general, implementing cryptographic protocols on constrained devices poses a complicated problem. On such devices it is standard for memory and processing power to be very limited. Hence, signature and key sizes must be kept as small as possible. Moreover, generating keys on these constrained devices efficiently has proven to be rather challenging. To our knowledge, the authors of [18] give the first full implementation of a hash-based signature scheme on smart cards.

The goal of XMSS$^+$ was to present a practical, forward secure signature scheme which solves the problem of on-card key generation and which could practically compete with signature schemes such as RSA and ECDSA at least in terms of runtimes. While the authors of [18] did indeed accomplish most of theses goals, their objective of forward security falls short because the proof they use to show forward security utilizes the erroneous Theorems 3.3.7 and 4.2.1. However, we still describe the scheme and highlight some of the important techniques used in the paper; in particular, the tree chaining technique from [10] and distributed signature generation from [8]. XMSS$^+$ also uses a more efficient algorithm for computing authentication paths due to Buchmann, Dahmen and Schneider (BDS) [9].

Compared to the original XMSS scheme, XMSS$^+$ reduces the key generation time from $\mathcal{O}(N)$ to $\mathcal{O}(\sqrt{N})$, where N is the number signatures which can be produced from a single key pair. To avoid confusion, we remark that XMSS$^+$ is **not** simply XMSS instantiated with WOTS$^+$, but rather the paper introducing XMSS$^+$ was published before that of WOTS$^+$. However, as is noted in [19, Section 4], using WOTS$^+$ as the underyling OTS in XMSS$^+$ does indeed offer improvements.

XMSS$^+$ uses essentially the same primitives and underlying OTS as XMSS does. One can think of XMSS$^+$ as XMSS using a hyper tree with two levels, each of height $h/2$ (cf. Section 2.3.5). However, in this version of XMSS, the pseudorandom generator used is more explicit. WOTS-PRF key pairs are generated in two steps. First, a stateful, forward secure pseudorandom generator $\text{FsGen} : \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$ is used to generate a new state and a pseudorandom value is used to generate the key pair via $\mathcal{F}_n$. This is done by first selecting a uniformly random initial state $S_0 \in \{0,1\}^n$ and then computing

$$\text{FsGen}(S_i) = (f_{S_i}(0) \| f_{S_i}(1)) = (S_{i+1} \| R_i).$$

The output R_i is used to generate the i^{th} WOTS secret key $\mathsf{sk}^i = (\mathsf{sk}^i_1, \mathsf{sk}^i_2, \ldots, \mathsf{sk}^i_\ell)$ as

$$\mathsf{sk}^i_j = f_{R_i}(j-1), \quad 1 \leq j \leq \ell.$$

Before we can describe the scheme, we need first describe the underlying algorithms used.

4.3.1 The BDS Algorithm

The first algorithm we describe is the so-called BDS Algorithm from [9]. The BDS algorithm is a more efficient method of computing authentication paths in Merkle trees; it increases the memory used, but drastically cuts down on generation times. While we do not get into the intricate technical details of the algorithm, we give a high level overview of how it works so as to make our explanation of XMSS$^+$ more clear. For more specific details see [9].

The algorithm uses a parameter $k \in \mathbb{N}$. This parameter represents a memory-time trade-off; the larger k is the more memory is used, but faster authentication paths can be generated and vice versa. To really drive home the usefulness of this algorithm we mention that in the worst case to generate an authentication path requires $2^h - 1$ leaf computations and evaluations of the TreeHash algorithm (which we describe soon). The BDS algorithm reduces worst-case signing time to only $(h - k)/2$ such evaluations. We note that we will always ensure that $h - k$ is even. The algorithm realizes that the major computational expenses during authentication path generation involve computing right hand nodes, whereas left nodes can be computed cheaply with a few tweaks and observations. The algorithm accomplishes the following three things.

- The BDS algorithm stores certain right nodes that occurred in earlier authentication paths so that left child nodes can be computed from these values using only a single evaluation of the hash function. Observe that right nodes do not have their children used in previous authentication paths and so they must be computed from scratch.
- During key generation the algorithm stores the right nodes from the top k levels of the tree. As these nodes are the most expensive to compute they need only be computed once. In particular, they do not need to be recomputed every time a signature is generated.
- Finally, the algorithm distributes the computations required to compute right nodes amongst previous signature generations. The computation of the next right node on a particular level begins when the most recently computed right node becomes part of the authentication path. This is done using one instance of the TreeHash algorithm that we describe next.

The BDS algorithm uses an internal state $\mathsf{State}_{\mathrm{BDS}}$.

4.3.2 The TreeHash Algorithm

The second algorithm we describe is the TreeHash algorithm. This algorithm is designed to more efficiently construct hash trees. As a subroutine, the TreeHash algorithm uses the BDS algorithm. The initial BDS state $\mathsf{State}_{\mathrm{BDS}}$ is initialized during the root computation we describe below. The algorithm initializes an empty stack Stack, and randomly selects an initial state $S_0 \in \{0,1\}^n$. The algorithm systematically uses the current state to generate the leaves and tree as follows.

1. Compute $\textsc{FsGen}(S_i)$ to obtain a new state S_{i+1} and random output R_i.
2. R_i is used to generate the i^{th} WOTS key pair as described above, and subsequently generates the i^{th} leaf of the tree using the corresponding L-tree.
3. The leaf and Stack are used as input into TreeHash to recursively update Stack:
   (a) While the top node on Stack has the same height as the input node N, set $t \leftarrow N.height() + 1$, and set $N \leftarrow H((\mathsf{Stack}.pop() \| N) \oplus Q_t)$.
   (b) Push the new node N onto Stack: $\mathsf{Stack}.push(N)$.
   (c) Return Stack.
4. Delete the used R_i and WOTS-PRF key pair from memory.
5. After all 2^h leaves have been input into TreeHash, Stack only contains the root node $v_h[0]$.

An XMSS public key PK consists of the set of $h + \lceil \log(\ell) \rceil$ bitmasks, the value x used in the underlying OTS, and the root $v_h[0]$. The secret key SK contains the initial state S_0 and the initial BDS state $\mathsf{State}_{\mathrm{BDS}}$, and is updated accordingly.

4.3.3 XMSS$^+$ Algorithms

The XMSS$^+$ tree is a two-level hyper tree. We call these the upper and lower trees respectively. The same value x for WOTS-PRF and the same bitmasks are used for both levels. XMSS$^+$ uses a randomly selected member H of the second-preimage resistant hash function family $\mathcal{H}_n = \{H_k : \{0,1\}^{2n} \rightarrow \{0,1\}^n \mid k \in \{0,1\}^n\}$ to construct the trees.

Key generation: The key generation algorithm takes as input the security parameter n, the length m of message digests to be signed, and the even tree height h.

1. Set the internal tree height $h' = h/2$. Select two Winternitz parameters w_u and w_l, and two BDS parameters k_u and k_l such that $h' - k_u$ and $h' - k_l$ are both even, and such that $(h' - k_u)/2 + 1 \leq 2^{h'-k_l+1}$.

2. Compute two sets of WOTS parameters $\ell_{1,i}, \ell_{2,i}$ and ℓ_i for $i \in \{u, l\}$ as follows.
   (a) If $i = u$, then calculate the parameters using input w_u and n.
   (b) If $i = l$, then calculate the parameters using input w_l and m.

3. Uniformly at random select $x \in \{0,1\}^n$ and $h' + \max\{\log(\ell_u), \log(\ell_l)\}$ bitmasks $Q_i \in \{0,1\}^{2n}$.

4. The upper tree $\mathcal{U}$ uses the parameters w_u and k_u and signs messages of length n. The lower trees $\mathcal{L}$ use parameters w_l and k_l and signs digests of length m. Construct all of these trees as in Section 4.2, but utilizing the TreeHash algorithm.

5. The root node $v_{h',\mathcal{L}}[0]$ of the first $\mathcal{L}$ is signed using the first (leftmost) WOTS-PRF key pair of $\mathcal{U}$.

6. A FsGen state for the next $\mathcal{L}$ on the lower level is selected, and a new TreeHash stack Stack_{next} is initialized.

The XMSS$^+$ public key PK consists of the $h' + \max\{\log(\ell_u), \log(\ell_l)\}$ bitmasks, the value x, and the root node of $\mathcal{U}$. The secret key SK is a bit more complicated; it contains the two FsGen states S_u and S_l, as well as the BDS states $\mathsf{State}_{\mathrm{BDS},u}$, $\mathsf{State}_{\mathrm{BDS},l}$, the signature on $v_{h',\mathcal{L}}[0]$ and finally the TreeHash stack Stack_{next}, its corresponding BDS state $\mathsf{State}_{\mathrm{BDS},n}$, and an FsGen state S_n.

Signature generation: To sign the i^{th} message $M \in \{0,1\}^*$ using secret key SK do as follows.

1. Sign M as follows:
   (a) The secret key SK contains the states S_l and $\mathsf{State}_{\mathrm{BDS},l}$. Let $j = i \bmod 2^{h'}$. Evaluate $\mathrm{FsGen}(S_l)$ to obtain the next state and randomness R_j.
   (b) Update the secret key with the new state.
   (c) Use R_j to generate WOTS-PRF key pair $(\mathsf{pk}^j, \mathsf{sk}^j)$.
   (d) Sign M using sk^j to obtain σ_l, the one-time signature on M.

2. Compute the authentication path Auth_l to the root node of the current lower tree using the BDS tree traversal algorithm with the appropriate parameters.

3. Sign the root node of $\mathcal{L}$ using the appropriate leaf of $\mathcal{U}$ as described in Section 2.3.5 to obtain the signature σ_u and compute the authentication path Auth_u from this leaf to the root node of $\mathcal{U}$.

The signature generated is $\Sigma = (\sigma_u, \mathsf{Auth}_u, \sigma_l, \mathsf{Auth}_l)$. However, there are still a few details about this algorithm which we have yet to describe. During the generation of σ_l, the BDS algorithm receives $(h' - k_l)/2$ updates. If any of these updates are not used to update $\mathsf{State}_{\mathrm{BDS},l}$, then those updates are instead used to update $\mathsf{State}_{\mathrm{BDS},u}$. Then compute one leaf on the next lower tree which is then used as input to TreeHash to update Stack_{next}.

If we have reached the last leaf on the current lower tree (i.e., if $i \bmod 2^{h'} = 2^{h'} - 1$), then Stack_{next} contains the root of the next lower tree. Now, the root of this new lower tree is signed using $\mathcal{U}$ as before. At this point, BDS receives no updates; the updates needed to compute the next authentication path will come from the next $2^{h'}$ signatures. We update the secret key SK by replacing $\mathsf{State}_{\mathrm{BDS},l}$, and S_l and the signature of the root of the lower tree by $\mathsf{State}_{\mathrm{BDS},n}$, S_n, and the new signature. Finally, the data structures for the next lower tree are initialized as detailed above, and these replace the ones in SK.

Signature verification: To verify a signature $\Sigma = (\sigma_u, \mathsf{Auth}_u, \sigma_l, \mathsf{Auth}_l)$ on a message M using public key PK do as follows.

1. Run the WOTS-PRF verification algorithm on M and σ_l to compute corresponding WOTS-PRF public key.
2. Using the WOTS public key, construct the corresponding L-tree and its associated root node (a leaf node of $\mathcal{L}$).
3. Use Auth_l, the above leaf node and the index $j = i \bmod 2^{h'}$ to compute the root node of the lower tree.
4. Treat this new root node as a message and verify the signature σ_u on it to compute the corresponding WOTS-PRF public key of the upper tree.
5. Using this new WOTS-PRF public key, construct the corresponding L-tree and its associated root node (a leaf node of $\mathcal{U}$).
6. Use Auth_u, the above leaf node and the index $j = \lfloor i/2^{h'} \rfloor$ to compute the root node of the upper tree.

The signature is accepted if and only if this final root node equals the root node included in PK.

Sizes and runtimes: The size of a signature is $(h + \ell_u + \ell_l)n$ bits. The size of the public key is $(h + 2\max\{\log(\ell_u), \log(\ell_l)\} + 2)n$ bits. The private key has size at most $(7.5h - 7k_l -$

$5k_u + 2^{k_l} + 2^{k_u} + \ell_u)n$ bits. Let t_H and t_f denote the times required for one evaluation of H and f respectively. The worst-case key generation time is $2^{h/2}(\ell_u + \ell_l + 1)t_H + 2^{h/2}(4 + \ell_u(w_u + 1) + \ell_l(w_l + 1))t_f$. The worst-case signing time is less than $\max_{i \in \{l,u\}} \{(((h' - k_l + 2)/2) \cdot (h' - k_i + \ell_i) + h')t_H + (((h' - k_l + 4)/2 \cdot (\ell_i(w_i + 1)) + h' - k_l)t_f\}$. Finally, the signature verification time is $(\ell_u + \ell_l + h)t_H + (\ell_u w_u + \ell_l w_l)t_f$.

4.4 XMSS$^{\mathrm{MT}}$

The signature scheme we discuss in this section can be viewed as a d-level hyper tree implementing the original XMSS scheme blended with the tree-chaining idea from [10] and the distributed signature generation from [8] (actually it uses an improved version of the distributed signature generation technique discussed in [18]).

Given an exact security reduction for a particular signature scheme, one can calculate secure parameters for the scheme at a given security level. This sort of parameter selection has been done for example in [5, 7, 8, 10, 12, 19] among others. However, these parameters are selected to be *secure* parameters for some security level, not necessarily *optimal* parameters. In fact, the problem of selecting optimal parameters for a given security level can be a highly non-trivial task. In some instances a parameter space consisting only of reasonable parameters can grow to the order of 2^{80} or larger (according to the authors of [20]). In [20] the authors introduce yet another variant on the XMSS scheme called XMSS$^{\mathrm{MT}}$ (Multi-Tree XMSS) and describe a way to select optimal parameters for this scheme for a given security level.

Due to the technical similarities between XMSS$^{\mathrm{MT}}$ when compared to XMSS^{+} and the scheme we detail in Section 4.6, we omit the algorithmic details of XMSS$^{\mathrm{MT}}$. However, we do highlight some of these similarities. One can picture XMSS$^{\mathrm{MT}}$ as XMSS^{+} with a d-level hyper tree (as opposed to two levels). In particular, XMSS^{+} is a special case of XMSS$^{\mathrm{MT}}$.

To clarify, XMSS$^{\mathrm{MT}}$ uses the BDS algorithm and the TreeHash algorithm outlined in Section 4.3. In particular, it uses d instances of these algorithms, one for each layer in the hyper tree, each of which having its own parameter sets. Furthermore, each level of the tree also has its own bitmasks and its own parameters for the underlying OTS (which is some variant of WOTS). The key difficulty in this scheme is the maintenance and the updating of the various states within the keys and algorithms. For explicit details on how this is done we refer the reader to the original paper [20]. We remark that state management is impressively problematic when it comes to implementing these schemes securely— recall Section 2.3.7.

The advantage of XMSS$^{\text{MT}}$ over previous versions of XMSS is that utilizing more levels in the hyper tree allows for the signing of "a virtually unlimited number of signatures". However, the security of XMSS$^{\text{MT}}$ is based on the erroneous Theorem 4.2.1 from Section 4.2, and so the actual calculated parameters are called into question. However, we emphasize that the overarching methodology seems to be sound.

To systematically select a provably optimal set of parameters for a particular instantiation of XMSS$^{\text{MT}}$, the authors of [20] employ the so-called *generalized lambda technique* of the theory of optimization [26] to linearize the problem instance and then efficiently solve for provably optimal parameters using the very well known Simplex algorithm [11, Part 1]. Indeed using linear optimization to solve for optimal parameters has been done before, but Hülsing et al. give a more explicit formulation of this problem than has been done previously. As this thesis does not focus on Linear Optimization, we omit these technical details, but again refer the reader to the source material [20].

4.5 XMSS-T

Recall Sections 2.3.6 and 3.5. In the former we discussed how schemes have a linear drop in security if multi-target attacks are allowed, and in the latter we described a variant WOTS-T on the Winternitz scheme designed to be resilient to these multi-target attacks. In this section we describe XMSS-T [21], a variant on XMSS using WOTS-T as the underlying one-time scheme. This new scheme is important because it is the only member of the XMSS family to explicitly consider the threat of multi-target attacks and to be proven secure against them. Moreover, in [21] the authors introduce concrete definitions of multi-target security notions for hash functions.

This scheme uses all of the primitives for WOTS-T from Section 3.5 as well as a cryptographic hash function $H : \{0,1\}^n \times \{0,1\}^{2n} \to \{0,1\}^n$ and an "arbitrary input randomized hash function" $\mathcal{H} : \{0,1\}^m \times \{0,1\}^* \to \{0,1\}^m$, where m is the length of message digests to be signed.

In order to be secure against multi-target attacks, we need to modify the binary tree construction slightly. These trees will be of height h (with 2^h leaves). As usual we denote nodes by $v_j[i]$, where j denotes the level of the node and i denotes the index from left to right. To construct the tree, first randomly select n-bit leaf nodes as $v_0[i] \in \{0,1\}^n$ for $i = 0, 1, \ldots, 2^h - 1$ and compute the internal nodes as

$$v_j[i] = H_{k_{i,j}}((v_{j-1}[2i] \| v_{j-1}[2i+1]) \oplus r_{i,j}),$$

where key $k_{i,j} = \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\textsc{Tree}}, 4 \cdot (j+i)))$, and bitmask $r_{i,j} = (\mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\mathcal{C}}, 4 \cdot (j+i)+1)) \| \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\mathcal{C}}, 4 \cdot (j+i)+2)))$. Just as in Section 3.5, $\mathbf{a}_{\mathcal{C}}$ is the Winternitz chain address, and now $\mathbf{a}_{\textsc{Tree}}$ is a unique address for the tree we are constructing.

We remark that to increase practicality, one would need to instantiate XMSS-T with the distributed signature generation method as well as the BDS algorithm from Section 4.3. However, for the sake of giving a clear description of the scheme we do not include those details.

The full scheme will consist of a hyper tree of total height h divided into d layers, each of height h/d. The leaves of each of these trees will themselves be compressed WOTS-T public keys (i.e., they are the root nodes of L-trees). In this sense, one can view a tree in this scheme as a key pair able to sign $2^{h/d}$ messages. The root layer, layer $d-1$, consists of a single tree, layer $d-2$ consists of $2^{h/d}$ trees — one for each leaf on layer $d-1$ — and the roots of these trees will in turn be signed by the corresponding leaf nodes above them on layer $d-1$. In general, layer i consists of $2^{(d-1-i)(h/d)}$ trees whose roots are all signed by the corresponding leaf nodes on layer $i+1$. In layer 0, the leaf nodes of trees are used to sign message digests.

The reader will notice a lot of similarities between this scheme and the final scheme we describe in this thesis (Section 4.6). Hence, understanding this scheme will aid in the understanding of the next one. The authors of [21] note that the techniques used in XMSS$^{\text{MT}}$ could in fact be used to improve the scheme in Section 4.6, but we leave this discussion for other work.

We now describe the algorithms of XMSS-T.

Key generation: On input of security parameter n, the key generation algorithm proceeds as follows.

1. Select two secret values $\mathcal{S}, \mathcal{S}' \xleftarrow{\$} \{0,1\}^n$. The first value $\mathcal{S}$ will be the global secret used to generate WOTS-T keys pseudorandomly while $\mathcal{S}'$ is used to generate pseudorandom inputs to randomize the message digest in the signing algorithm.

2. Select the public seed $\textsc{Seed} \xleftarrow{\$} \{0,1\}^n$.

3. Use $\mathcal{S}$ to generate $2^{h/d}$ WOTS-T key pairs. Compress the public keys into the leaf nodes of the tree on layer $d-1$. Then as described above, use these leaves to construct the full binary tree. Denote the root of this tree as PK_1.

The public key is $\mathsf{PK} = (\mathsf{PK}_1, \textsc{Seed})$ and the secret key is $\mathsf{SK} = (0^h, \mathcal{S}, \mathcal{S}', \textsc{Seed})$, where

0^h is the h-bit string of 0's, and denotes the index of the next WOTS-T key pair used for signing; this index is updated accordingly.

Signature generation: On input of message $M \in \{0,1\}^*$ and a secret key SK, the signing algorithm proceeds as follows.

1. Compute a randomization element $R = \mathcal{F}_m(\mathcal{S}', M)$.
2. Compute a randomized message digest $D = \mathcal{H}(R, M)$.
3. Extract index i from the secret key. Let i_0 be given by the last h/d digits of i, and i'_0 be given by the first $(d-1)(h/d)$ bits of i. Then i corresponds to the i_0^{th} key pair in tree i'_0 of level 0.
4. Sign the message digest D with the i_0^{th} WOTS-T secret key to receive signature $\sigma_{w,0}$.
5. Compute the authentication path Auth_{i_0} from the leaf indicated by index i to the root node of that tree.
6. For layer δ, with $1 \leq \delta \leq d-1$, let $i = i'_{\delta-1}$ be the index of the root node on level $\delta - 1$ that we wish to sign next. Compute i_δ as the last h/d digits of $i'_{\delta-1}$, and i'_δ as the first $(d-1)(h/d)$ bits of $i'_{\delta-1}$.
7. Sign the root node on level $\delta - 1$ with the i_δ^{th} WOTS-T key pair in the tree with index i'_δ on level δ to receive signature $\sigma_{w,\delta}$.
8. Compute the authentication path Auth_{i_δ}.

The signature produced is $\Sigma = (i, R, \sigma_{w,0}, \mathsf{Auth}_{i_0}, \ldots, \sigma_{w,d-1}, \mathsf{Auth}_{i_{d-1}})$.

Signature verification: On input of message M, signature Σ and public key PK, the verification algorithm proceeds as follows.

1. Extract randomness R from Σ and use it to compute $D = \mathcal{H}(R, M)$.
2. Run the WOTS-T verification algorithm with input D, public seed SEED, address $\mathbf{a}_{\mathrm{OTS}_0}$ and signature $\sigma_{w,0}$ to compute the first WOTS-T public key as $\mathsf{pk}_{w,0}$.
3. Construct the L-tree corresponding to $\mathsf{pk}_{w,0}$ and compress it to root node Root_0 using index i_0 and Auth_{i_0}.
4. Repeat this procedure for the remaining $d-1$ levels of the hyper-tree. After all layers are used the final root node computed is Root_{d-1}.

The algorithm accepts the signature if and only if $\text{Root}_{d-1} = \mathsf{PK}_1$.

Sizes: An XMSS-T public key is $2n$ bits, a secret key is $h + 3n$ bits, and signatures are $h + m + d(n\ell) + nd(h/d) = h + m + n(d\ell + h)$ bits. So clearly we see that even though the key sizes are relatively small, the signature size is rather unwieldy. Large signature sizes are inherent with hyper tree style signature schemes. This represents a trade-off between signature size and the number of messages one can securely and practically sign.

As a final remark about XMSS-T, it is the only member of the XMSS family to incorporate not only multi-target resilience, but also a rigorous quantum security analysis. In fact, one can find a complete and detailed proof of security in the Quantum Random Oracle Model of XMSS-T in the source material [21]. Often, quantum security is assumed based on generic quantum attacks against hash function, but here one can find a rigorous analysis. We omit the details and statement of this security result.

4.6 SPHINCS

It is in this section that we describe the final signature scheme of this thesis. We present the hash-based signature scheme Stateless Practical Hash-based Incredibly Nice Cryptographic Signatures, or SPHINCS [2]. As mentioned in Section 2.3.7, one of the biggest challenges in implementing hash-based signature schemes is in secure management of states. The schemes we have described thus far in this chapter all have many states to keep track of, especially if one implements them with the BDS algorithm and distributed signature generation techniques.

SPHINCS also provides benefits besides statelessness. For example, instead of using a one-time signature scheme to sign message digests, SPHINCS uses a few-time signature scheme (cf. Section 2.3.1). This allows for a smaller overall tree height (fewer leaves) which in turn reduces the sizes of signatures as fewer authentication paths and one/few-time signatures need be included in a full signature. Reducing the overall height of the hyper-tree does exponentially increase signing time, but these trade-offs are typical for such schemes.

The underlying FTS used in SPHINCS is based on the HORS scheme described in Section 2.3.1. The authors of [2] noticed that HORS has very large public keys, but SPHINCS signatures must include the public keys of the underlying FTS. It is for this reason that they introduced HORST (HORS with Trees). This new FTS construction has substantially shorter public keys than its predecessor which allows for shorter SPHINCS

signatures. However, as mentioned above we suffer a trade-off to get these shorter keys, namely the runtimes of HORST algorithms are longer than those of HORS.

For the schemes discussed in the remainder of this section we will use the following public functions for the security parameter n and message digest length $m = \text{poly}(n)$.

- Two cryptographic hash functions $F : \{0,1\}^n \to \{0,1\}^n$ and $H : \{0,1\}^{2n} \to \{0,1\}^n$.
- An arbitrary input, randomized hash function $\mathcal{H} : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^m$.
- A family of pseudorandom generators $G_\lambda : \{0,1\}^n \to \{0,1\}^{\lambda n}$.
- A family of pseudorandom functions $\mathcal{F}_\lambda : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$.
- A pseudorandom function $\mathcal{F} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^{2n}$.

In practice each of these functions can in fact be built from the same cryptographic hash function, but for purposes of readability we separate them according to their roles.

4.6.1 HORST

This scheme signs m-bit message digests using parameters $k, \tau \in \mathbb{N}$, with $t = 2^\tau$ such that $m = k\tau$. The difference between this scheme and its predecessor is that HORST uses a binary hash tree that reduces the public key to the root node. In what follows we assume that the bitmasks used are publicly known. Let x be a positive integer minimizing $n(k(\tau - x + 1) + 2^x)$; this will be the public key size. It is possible for two consecutive values of x to minimize this value; if this happens then select the larger value of x.

Key generation: On input of secret seed $\mathcal{S}$ and collection of bitmasks $\mathcal{Q} = \{Q_1, \ldots, Q_\tau\}$, with each $Q_i \in \{0,1\}^{2n}$, the key generation algorithm proceeds as follows.

1. Compute $\mathsf{sk} = G_t(\mathcal{S}) = (\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_t)$.
2. For $i = 0, 1, \ldots, t-1$, compute $L_i = F(\mathsf{sk}_i)$. These L_i will be the leaves of the tree.
3. Construct the hash tree using the L_i and bitmasks $\mathcal{Q}$.

The public key pk is the root node of this tree; the secret key is $\mathcal{S}$.

Signature generation: On input of message digest $d \in \{0,1\}^m$, seed $\mathcal{S} \in \{0,1\}^n$ and bitmasks $\mathcal{Q} \in \{0,1\}^{2n \times \tau}$, the signing algorithm proceeds as follows.

1. Compute the secret key sk as in key generation.

2. Divide d into k τ-bit strings as, $d = d_0 \| d_1 \| \ldots \| d_{k-1}$. Interpret each d_i as an unsigned integer.

3. For $i = 0, 1, \ldots, k-1$, compute $\sigma_i = (\mathsf{sk}_{d_i}, \mathsf{Auth}_{d_i})$ where the authentication path Auth_{d_i} contains only the lower $\tau - x$ elements of the full authentication path for the corresponding leaf $L_{d_i} = F(\mathsf{sk}_{d_i})$.

4. Compute $\sigma_k = (v_{\tau-x}[0], v_{\tau-x}[1], \ldots, v_{\tau-x}[2^x - 1])$, the collection of all 2^x nodes on level $\tau - x$.

The signature produced is $\sigma = (\mathsf{pk}, \sigma_0, \sigma_1, \ldots, \sigma_k)$. Observe that the block σ_k can be computed and stored during key generation. The underlying concept is similar to that of the BDS algorithm.

Signature verification: On input of message digest $d \in \{0,1\}^m$, signature σ and bitmasks $\mathcal{Q} \in \{0,1\}^{2n \times \tau}$, the verification algorithm proceeds as follows.

1. Compute the d_i as in signature generation.

2. For $i = 0, 1, \ldots, k-1$ compute $L_{d_i} = F(\mathsf{sk}_{d_i})$.

3. For $i = 0, 1, \ldots, k-1$ set $y_i = \lfloor d_i / (2^\tau - x) \rfloor$.

4. For $i = 0, 1, \ldots, k-1$ compute node $v'_{\tau-x}[y_i]$ using L_{d_i}, $\mathcal{Q}$ and Auth_{d_i} as described in Section 2.3.3.

5. For $i = 0, 1, \ldots, k-1$ check if $v'_{\tau-x}[y_i] = v_{\tau-x}[y_i]$ using σ_k.

6. If each comparison in the previous step holds, then use σ_k to compute the root node Root_0. The signature is accepted if and only if this happens.

Technically the verification algorithm should compare the computed root node to the public key. However, when instantiated in the full SPHINCS scheme this comparison is an unnecessary operation.

Sizes and runtimes: The size of a HORST public key is only n bits, as is the secret key. A signature is $n(k(\tau - x + 1) + 2^x)$ bits. Key generation requires only a single evaluation of G_t and t evaluations of F to obtain the leaves. From there it takes $t-1$ hashes to compute the public key, giving a total of $2t - 1$ hash evaluations. Signing a message digest also takes $2t-1$ hash evaluations because, we need to calculate the root node. The verification algorithm requires k evaluations of F to compute the leaf nodes. In addition, it requires $\tau - x$ hash evaluations to compute the nodes on level $\tau - x$. Furthermore, $2^x - 1$ hashes are needed to compute Root_0 from σ_k. In total then, verification requires $k(\tau - x + 1) + 2^x - 1$ hashes. Recall that this value is minimized by x.

4.6.2 SPHINCS Algorithms

In addition to the few-time signature scheme HORST, SPHINCS also uses the one-time signature scheme WOTS$^+$ from Section 3.4. SPHINCS uses a d-level hyper tree construction of total height h where the root nodes of internal trees are signed using WOTS$^+$ keys and the trees on the bottom level are HORST trees (see Figure 4.1). The leaves of the internal trees are compressed WOTS$^+$ keys. We do not include the height of the HORST trees at the bottom of this construction in the total tree height.

Similarly to XMSS-T, SPHINCS uses an addressing scheme for pseudorandom key generation. Set $a = \lceil \log(d+1) \rceil + h$; an address is a bitstring of length a. The address of a WOTS$^+$ key pair is obtained as follows. Encode the layer of the tree it belongs to as a bitstring of length $\log(d+1)$ unless it is the top layer in which case use $\log(d-1)$ bits. Next, concatenate the index of the tree within the layer as a $(d-1)(h/d)$-bit string with indexing starting from 0 moving left to right. Finally, concatenate the index of the WOTS$^+$ key pair within the tree as an (h/d)-bit string using the same indexing method. Thus, a WOTS$^+$ key pair address looks like $A = (\text{layer} \| \text{index of tree} \| \text{index of key pair})$. A HORST key pair address is obtained from the address of the WOTS$^+$ key pair used to sign its public key by replaying the layer index with d encoded as a bitstring of length $\lceil \log(d+1) \rceil$.

As the SPHINCS algorithms are quite involved, we give a high-level overview of the key generation and signature algorithms before we describe them.

To generate a SPHINCS key pair, one need only generate the topmost tree. To do so, we first randomly select two secret seeds and sufficiently many bitmasks so that each data structure that needs bitmasks in this construction has one for each of its levels. Using distinct addresses we compute $2^{h/d}$ pseudorandom seeds that we then use together with bitmasks as inputs into the WOTS$^+$ key generation algorithm to produce $2^{h/d}$ WOTS$^+$ public keys. Then we compute the root nodes of the L-trees corresponding to these WOTS$^+$ public keys. These root nodes will be the leaves of the topmost tree. Construct the topmost tree according to the SPR-MSS construction. The global public key contains the root node of this tree and the collection of bitmasks, and the secret key contains the two seeds and the collection of bitmasks.

Key generation: On input of security parameter n, key generation proceeds as follows.

1. Select two secret seeds $\mathcal{S}, \mathcal{S}' \xleftarrow{\$} \{0,1\}^n$. The first seed $\mathcal{S}$ will be used for pseudorandom key generation while the other secret $\mathcal{S}'$ is used to generate pseudorandom inputs to randomize the message digest in the signing algorithm, as well as for generating indicies in the signing algorithm.

2. Set $p = \max\{w-1, 2((h/d) + \lceil \log(\ell) \rceil), 2\log(t)\}$[1]. Select bitmasks $Q_1, Q_2, \ldots, Q_p \xleftarrow{\$} \{0,1\}^{2n}$. Let $\mathcal{Q}_{\text{WOTS}^+}$ consist of the first $w-1$ of these bitmasks, $\mathcal{Q}_{\text{HORST}}$ consist of the first $2\log(t)$ bitmasks, $\mathcal{Q}_{\text{L-Tree}}$ consist of the first $2\lceil \log(\ell) \rceil$ bitmasks, and $\mathcal{Q}_{\text{Tree}}$ consist of the $2h$ bitmasks which come after $\mathcal{Q}_{\text{L-Tree}}$. Denote by $\mathcal{Q}$ the entire collection of bitmasks. Note that some bitmasks may belong to more than one of the five subsets of $\mathcal{Q}$ we described.

3. Generate $2^{h/d}$ WOTS$^+$ key pairs for the tree on layer $d-1$. Do this as follows for $i = 0, 1, \ldots, 2^{h/d} - 1$.
   (a) Compute the WOTS$^+$ key address as $A = (d-1 \| 0 \| i)$
   (b) Compute the seed $\mathcal{S}_A = \mathcal{F}_a(A, \mathcal{S})$. We assume without loss of generality that these seeds are known to any algorithm that knows $\mathcal{S}$. In general this is how we compute seeds with address A.
   (c) Use seed $\mathcal{S}_A$ and bitmasks $\mathcal{Q}_{\text{WOTS}^+}$ as input into the WOTS$^+$ key generation algorithm to produce pk_A.
   (d) Compress pk_A into leaf L_i using an L-Tree with bitmasks $\mathcal{Q}_{\text{L-Tree}}$.

4. Use the set of leaves together with bitmasks $\mathcal{Q}_{\text{Tree}}$ to construct the binary hash tree. Denote the root of this tree by PK_1.

The public key is $\mathsf{PK} = (\mathsf{PK}_1, \mathcal{Q})$ and the secret key is $\mathsf{SK} = (\mathcal{S}, \mathcal{S}', \mathcal{Q})$.

To sign a message M, we first compute a randomized message digest D of M. Using a pseudorandom address we compute a pseudorandom seed that is used with D as input into the HORST signing algorithm to produce a HORST signature; we pseudorandomly generate a new HORST key pair each time we sign a message. Deterministically produce a new address from the previous one and use this new address to produce a new pseudorandom seed. This new seed is used to produce a WOTS$^+$ signature on the HORST public key contained within the HORST signature; at the same time we extract the corresponding WOTS$^+$ public key. Next, using our ensemble of pseudorandom functions and addresses, we construct an SPR-MSS tree where one of the leaves is the compressed WOTS$^+$ public key used to sign the HORST public key. We compute the authentication path corresponding to this leaf. The root node of this tree is denoted ROOT_0. Essentially we repeat this process for the remaining $d-1$ layers signing the root of the tree on the previous level using the WOTS$^+$ signing algorithm and computing the relevant authentication path. A SPHINCS signature contains this collection of WOTS$^+$ signature/authentication path pairs, the HORST signauture, and some pseudorandomness.

[1] The original version of [2] erroneously has h instead of h/d in the calculation for p.

In what follows, if x is a bitstring, we denote by $x(i,j)$ the substring of length $j-i$ of x beginning at the i^{th} position of x.

Signature generation: On input of message $M \in \{0,1\}^*$ and secret key SK, the signing algorithm proceeds as follows.

1. Compute $R = \mathcal{F}(M, \mathcal{S}')$ and parse R as $R = R_1 \| R_2$, where each $R_i \in \{0,1\}^n$.
2. Compute a randomized message digest $D = \mathcal{H}(R_1, \mathcal{S}')$.
3. Set index i equal to the first h bits of R_2.
4. Set $A_{\text{HORST}} = (d \| i(0,(d-1)(h/d)) \| i((d-1)(h/d),(h/d))$ and compute $\mathcal{S}_{A_{\text{HORST}}} = \mathcal{F}_a(A_{\text{HORST}}, \mathcal{S})$.
5. Run the HORST signing algorithm on D with seed $\mathcal{S}_{A_{\text{HORST}}}$ and bitmasks $\mathcal{Q}_{\text{HORST}}$ to obtain signature σ_H. Note that the public key pk_H is contained in σ_H.
6. Set A_0 to be A_{HORST} with the first $\lceil \log(d+1) \rceil$ bits set to 0.
7. Run the WOTS$^+$ signature algorithm on message pk_H with seed $\mathcal{S}_{A_0}$ and bitmasks $\mathcal{Q}_{\text{WOTS}^+}$ to obtain $\sigma_{w,0}$. This signature is essentially treated as a leaf of the tree on level 0.
8. Compute the authentication path Auth_{A_0} corresponding to the WOTS$^+$ key pair used to obtain $\sigma_{w,0}$.
9. Run the WOTS$^+$ verification algorithm for message pk_H and signature $\sigma_{w,0}$ using bitmasks $\mathcal{Q}_{\text{WOTS}^+}$ to obtain public key $\mathsf{pk}_{w,0}$. Note that this could also be calculated and stored in step 7.
10. Compute the root of the tree on level 0 ROOT_0 by first compressing $\mathsf{pk}_{w,0}$ via an L-Tree and then using the index of the WOTS$^+$ pair used in step 7, Auth_{A_0} and bitmasks $\mathcal{Q}_{\text{WOTS}^+}$.
11. Repeat for layers 1 through $d-1$ with the following differences.
    (a) On layer $1 \leq \delta \leq d-1$ use WOTS$^+$ to sign root $\text{ROOT}_{\delta-1}$ which is the root computed at the end of the previous iteration.
    (b) The address of the WOTS$^+$ key used on layer δ is computed as $A = (\delta \| i(0,(d-1-\delta)(h/d) \| i((d-1-\delta)(h/d), h/d))$. In words, the last h/d bits of the tree address become the new leaf address and the remaining bits of the previous tree address become the new tree address.

The signature produced is $\Sigma = (i, R_1, \sigma_H, \sigma_{w,0}, \mathsf{Auth}_{A_0}, \ldots, \sigma_{w,d-1}, \mathsf{Auth}_{A_{d-1}})$.

Signature verification: On input of message $M \in \{0,1\}^*$, signature Σ and public key PK, the verification algorithm proceeds as follows.

1. Extract randomness R_1 from Σ and compute $D = \mathcal{H}(M, R_1)$ as in signature generation.

2. Run the HORST verification algorithm on signature σ_H for message digest D and bitmasks $\mathcal{Q}_{\text{HORST}}$ to obtain HORST public key pk_H. Continue if and only if the HORST signature is accepted.

3. Run the WOTS$^+$ verification algorithm on signature $\sigma_{w,0}$ for message pk_H with bitmasks $\mathcal{Q}_{\text{WOTS}^+}$ to obtain WOTS$^+$ public key $\mathsf{pk}_{w,0}$.

4. Compress $\mathsf{pk}_{w,0}$ using an L-Tree to compute $L_{i((d-1)(h/d),h/d)}$, the leaf corresponding to $\mathsf{pk}_{w,0}$.

5. Compute $\textsc{Root}_0$, the root node of the tree on layer 0 using Auth_0, $L_{i((d-1)(h/d),h/d)}$, and index $i((d-1)(h/d), h/d)$.

6. Repeat for layers 1 through $d-1$ with the following differences.
   (a) On layer $1 \leq \delta \leq d-1$, the root node $\textsc{Root}_{\delta-1}$ of the tree from the previous iteration is used as input into the WOTS$^+$ verification algorithm to compute $\mathsf{pk}_{w,\delta}$.
   (b) The leaf computed from $\mathsf{pk}_{w,\delta}$ using an L-Tree is $L_{i((d-1-\delta)(h/d),h/d)}$. In words, the index of the leaf within the tree can be obtained by cutting off the final $\delta(h/d)$ bits of i and then by using the final h/d bits of the resulting bitstring.

After the final iteration the algorithm has $\textsc{Root}_{d-1}$. The algorithm accepts the signature if and only if $\textsc{Root}_{d-1} = \mathsf{PK}_1$, the first coordinate of PK.

Figure 4.1 shows a high-level visual representation of the SPHINCS signature algorithm. The values on the right are the heights of the respective trees. The HORST tree is shown with hatched lines as its height is not included in the total tree height d.

Sizes: A secret key consists of the two n-bit seeds and the $p = \max\{w-1, 2(h/d + \lceil\log(\ell)\rceil, 2\log(t)\}$ bitmasks for a total of $n(2+p)$ bits. Public keys have size $n(p+1)$ bits. Signatures however are a bit larger. A signature contains an h-bit index, n bits of pseudorandomness, one $n(k(\tau - x + 1) + 2^x)$ bit HORST signature, d ℓn-bit WOTS$^+$ signatures each with h/d n-bit authentication path nodes. This gives a total signature size of $n((k(\tau - x + 1) + 2^x) + d\ell + h + 1) + h$ bits.

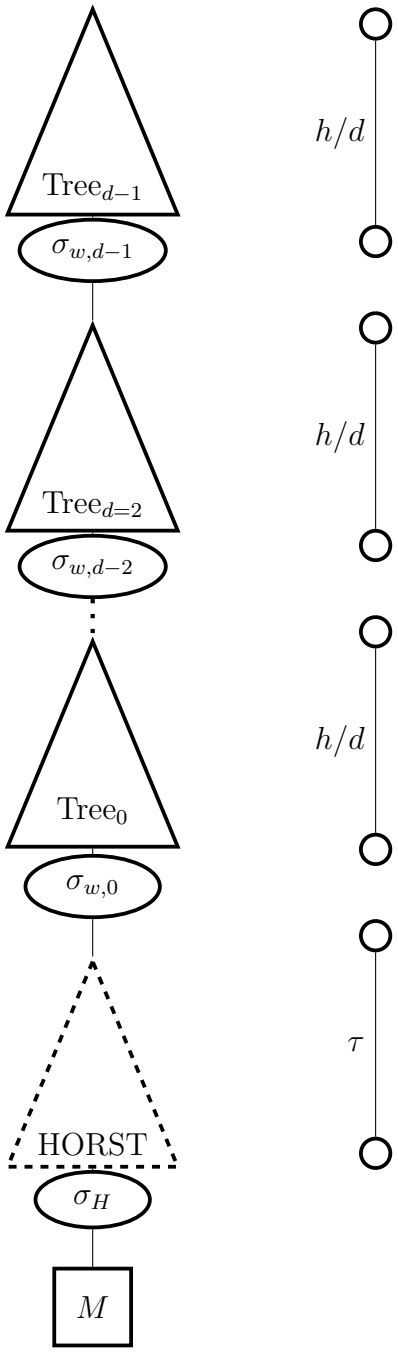

Figure 4.1: A visualization of the SPHINCS signature algorithm.

Runtimes: Key generation requires the construction of the topmost tree. It is not too difficult to see that this requires $2^{h/d}$ PRF calls, $2^{h/d}$ PRG calls, $2^{h/d}$ WOTS$^+$ key generations (each requiring ℓw hashes) and $2^{h/d}$ L-Tree constructions (each requiring $\ell - 1$ hashes). Furthermore, building the tree requires another $2^{h/d} - 1$ hashes. This requires a total of $\ell(w+1)2^{h/d} - 1$ hashes. The signature algorithm requires one PRF call to generate the index and randomness for the message digest as well as for the message digest itself. Then, it requires one PRF call to generate a HORST seed and a HORST signature. Next, d trees need to be built, thus adding d times the time for key generation. WOTS$^+$ signatures can be extracted during WOTS$^+$ key generation and thus incur no extra cost. This totals $d2^{h/d} + 2$ PRF calls, $d2^{h/d} + 1$ PRG calls and $2t + d((\ell(w+1)2^{h/d} - 1)$ hashes. Finally, the verification algorithm needs to compute the message digest, run the HORST verification, and d WOTS$^+$ verifications, L-Tree computations and $h/d-1$ hash evaluations to compute the root nodes. In total this requires $k(\tau - x + 1) + 2^x + d(\ell(w+1) - 2) + h$ hash evaluations.

For concreteness we give the parameters and sizes from [2] for SPHINCS-256, an instantiation of SPHINCS which provides 128 bits of security against a quantum adversary. This instantiation uses $n = 256, m = 512, h = 60, d = 12, w = 16, t = 2^{16}, \tau = 16$ and $k = 32$. These parameter choices yield, $\ell = 67, x = 6$, and $a = 64$. This leads to public keys with size 1056 bytes, secret keys of size 1088 bytes, and signatures of size 41000 bytes.

The number of messages that can be securely signed with SPHINCS-256 depends on what the user deems an acceptable probability for an adversary to succeed in producing a forgery. In [2, Section 3.2], the authors show that the probability that any particular HORST key pair is used to sign exactly γ messages after q queries is bounded above by

$$\left(\frac{q}{2^h}\right)^\gamma \exp\left(\frac{-q}{2^h}\right)\left(\frac{1}{\gamma!}\right).$$

Furthermore, it is shown that if a key pair is used to sign exactly γ messages, then an adversary can "mix and match" components of these valid signatures to break γ-subset-resilience with probability approximately $(t/(\gamma k))^\gamma$ with a classical attack, and approximately $(t/(\gamma k))^{\gamma/2}$ with a quantum attack. Attacking subset resilience is the main (generic) attack against SPHINCS. Thus, a SPHINCS user must find an acceptable balance for themselves between the chances of a key being used γ times and the amount of work an adversary must do in order to break the scheme if this event occurs.

Suppose that a SPHINCS-256 user were able to sign 2^{20} messages per second, and they do so continuously. It would then take more than thirty years for them to sign $q = 2^{50}$ messages. The probability of a fixed HORST key being used γ times is at most $\left(\frac{q}{2^h}\right)^\gamma \exp\left(\frac{-q}{2^h}\right)\left(\frac{1}{\gamma!}\right)$. From this, one can show that if $\rho = q/2^h$ is sufficiently smaller than

1 then there will be approximately q keys used once, approximately $\rho q/2!$ keys used twice, approximately $\rho^2 q/3!$ keys used three times, and so on. The probability that some key will be used $\gamma = -(h/\log(\rho)) + \delta$ times is approximately $\rho^\delta/\gamma!$. Now, if a key is used 9 times and $q = 2^{50}$ messages are queried, then a post-quantum attack has cost $(t/\gamma k)^{k/2} = (2^{16}/9 \cdot 32)^{16} \approx 2^{125.28}$. Thus, if a key is used $\gamma = 9$ times we have 125 bits of security at the post-quantum level; less than the desired 128-bit security level. However, the probability of a 9-time key reuse is approximately $\rho^\delta/\gamma! = (2^{-10})^3/9! \approx 2^{-48.5}$. Hence, SPHINCS-256 provides long-term security against this kind of attack. Assuming a careful choice of the other relevant functions, SPHINCS-256 provides long-term security against generic attacks at the 128-post-quantum and 256-classical bit levels of security.

We present the following theorem from [2] and remark that the reduction is tight. For details see the source material [2].

Theorem 4.6.1. *SPHINCS is existentially unforgeable under q_s-adaptive chosen message attacks if*

- *F is a second-preimage resistance, undetectable, one-way function,*
- *H is a second-preimage resistant hash function,*
- *G_λ is a secure pseudorandom generator for $\lambda \in \{\ell, t\}$,*
- *$\mathcal{F}_\lambda$ is a pseudorandom function family for $\lambda = a$,*
- *$\mathcal{F}$ is a pseudorandom function family, and*
- *for the subset-resilience of $\mathcal{H}_{k,t}$ it holds that*

$$\sum_{\gamma=1}^{\infty} \min\left\{2^{\gamma(\log(q_s)-h)+h}, 1\right\} \cdot Succ^{\gamma-sr}_{\mathcal{H}_{k,t}}(\mathcal{A}) = \mathsf{negl}(n)$$

for any probabilistic polynomial-time adversary $\mathcal{A}$, where $Succ^{\gamma-sr}_{\mathcal{H}_{k,t}}(\mathcal{A})$ denotes the success probability of $\mathcal{A}$ against the γ-subset resilience of $\mathcal{H}_{k,t}$.

More specifically, the insecurity function $InSec^{EU\text{-}CMA}(SPHINCS; \xi, q_s)$ describing the maximum success probability of all adversaries against the existential unforgeability under q_s-

adaptive chosen message attacks, running in time at most ξ, is bounded by

$$\begin{aligned}
&\mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F};\xi,q_s) + \mathrm{InSec}^{\mathrm{PRG}}(\mathcal{F}_a;\xi,N_{fts}+N_{ots}) \\
&+ (N_{ots}+N_{fts}) \cdot \mathrm{InSec}^{\mathrm{PRG}}(G_\ell;\xi) \\
&+ N_{tree} \cdot 2^{h/d+\lceil \log(\ell) \rceil} \cdot \mathrm{InSec}^{\mathrm{SPR}}(H;\xi) \\
&+ N_{ots} \cdot (\ell w^2 \cdot \mathrm{InSec}^{\mathrm{UD}}(F;\xi) + \ell w \cdot \mathrm{InSec}^{\mathrm{OW}}(F;\xi) + \ell w^2 \mathrm{InSec}^{\mathrm{SPR}}(F;\xi)) \\
&+ N_{fts} \cdot 2t \cdot \mathrm{InSec}^{\mathrm{SPR}}(H;\xi) + N_{fts} \cdot t \cdot \mathrm{InSec}^{\mathrm{OW}}(F;\xi) \\
&+ \sum_{\gamma=1}^{\infty} \min\left\{2^{\gamma(\log(q_s)-h)+h}, 1\right\} \cdot \mathrm{InSec}^{\gamma-sr}(\mathcal{H}_{k,t};\xi),
\end{aligned}$$

where $N_{ots} = \min\left\{\sum_{i=1}^{d} 2^{ih/d}, dq_s\right\}$ *denotes the maximum number of WOTS*$^{+}$ *key pairs,* $N_{fts} = \min\{2^h, q_d\}$ *denotes the maximum number of* HORST *key pairs, and* $N_{tree} = \min\left\{\sum_{i=1}^{d-1} 2^{ih/d}, (d-1)q_s\right\}$ *denotes the maximum number of subtrees used answering* q_s *signature queries.*

The SPHINCS signature scheme is particularily appealing due to its statelessness and fast runtimes, but the large signature sizes pose a problem for many real world applications. However, for many other applications larger signatures are acceptable.

Chapter 5

Conclusion

In this work we collected and detailed the underlying tools and techniques of hash-based cryptography. We described the one-time signature schemes which underlie the full signature schemes vying for standardization and pointed out serious flaws in the security proof and analysis of the WOTS-PRF signature scheme. We highlighted the major advantages and drawbacks of these various schemes and what the main obstacles are to efficiency and practicality. We juxtaposed stateful and stateless schemes, noting the security advantages and disadvantages of both. Although a work which describes *all* of the up to date intricate details (of theory and implementation) of each of the many schemes we presented here would be useful, such a project would be too unwieldy.

We conclude this thesis by highlighting a few potential projects for future work.

5.1 Future Work

Much work remains to be done on the problems of designing and optimizing practical, efficient and secure hash-based digital signature schemes. Ideally, we would like signatures to be smaller and signing/verification times decreased. Considering that global key pairs have very long life times, it is less important that key generation times be minimal — anything less than a day is likely acceptable.

Highly worthy of consideration is the recent work of McGrew et al. in [25]. Not only do the authors nicely highlight the problems associated with stateful signature schemes, but they present a theoretical construction for a hybrid stateful/stateless scheme. It is claimed that this sort of hybrid construction eliminates many of the difficulties associated

with maintaining a state. It is a worthwhile research project to analyze this construction and formulate a concrete instantiation along with a security proof and parameter recommendations for both classical and post-quantum security.

We mentioned in Section 4.5 that one could improve the SPHINCS scheme by incorporating some of the techniques from XMSS-T [21]. It would be interesting to concretely construct and analyze this scheme with an accompanied, benchmarked software implementation.

Another project for future work would be designing new few-time signature schemes with smaller signature sizes than HORST. It seems to be fundamental to few-time schemes that they have a large amount of secret information and reveal only a small fraction of this information in a signature. However, this area seems to be rather under-explored and we hope that perhaps more practical schemes can be devised.

Alternatively, if it proves too difficult to devise new and improved few-time schemes, then another avenue of research would be to decrease the computational effort required to produce Winternitz-style signatures. Currently, Winternitz-style signatures require many hash applications and often much random number generation (for example). This presents a significant overhead that would preferably be avoided.

A research project of theoretical interest would be to resurrect the WOTS-PRF scheme or design an alternative WOTS variant which employs pseudorandom functions. In addition to designing such a scheme, it would be desirable to include a reductionist security proof using only the pseudorandomness of the functions as a security assumption.

References

[1] D. J. Bernstein, J. Buchmann and E. Damen, (eds.) Post-Quantum Cryptography. Springer-Verlag Berlin Heidelberg (2009). http://www.springer.com/us/book/9783540887010.

[2] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, P. Schwabe and Z. Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In M. Fischlin and E. Oswald, (eds.), Advances in Cryptology — EUROCRYPT 2015. LNCS 9056, pp. 368-397. Springer (2015).

[3] D. J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? In SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems (2009).

[4] G. Brassard, P. Høyer and A. Tapp. Quantum algorithm for the collision problem. ACM SIGACT News (Cryptology Column), Vol 28, pp. 14-19, (1997).

[5] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing and M. Rückert: On the security of the Winternitz one-time signature scheme. In: A. Nitaj, D. Pointcheval, (eds.) AFRICACRYPT 2011. LNCS 6737, pp. 363-378. Springer (2011).

[6] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing and M. Rückert. On the security of the Winternitz one-time signature scheme. Int. J. Applied Cryptography, Vol 3, No. 1, pp. 84-96 (2013).

[7] J. Buchmann, E. Dahmen and A. Hülsing. XMSS – A practical forward secure signature scheme based on minimal security assumptions. In: B. Y. Yang, (ed.) PQCrypto 2011. LNCS 7071, pp. 117-129. Springer (2011).

[8] J. Buchmann, E. Dahmen, E. Klintsevich, K. Okeya and C. Vuillaume. Merkle signatures with virtually unlimited signature capacity. Proceedings of Applied Cryptography and Network Security 2007. LNCS 4521, pp. 31-45. Springer (2007).

[9] J. Buchmann, E. Dahmen and M. Schneider. Merkle tree traversal revisited. In: J. Buchmann and J. Ding, (eds.), Post-Quantum Cryptography, LNCS 5299, pp. 63-78. Springer (2008).

[10] J. Buchmann, L. C. Coronado Garcia, E. Dahmen, M. Dring and E. Klintsevich. CMSS — An improved Merkle signature scheme. R. Barua, T. Lange, (eds.) INDOCRYPT 2006. LNCS 4329, pp. 349-363. Springer (2006).

[11] V. Chvatal. Linear Programming. New York, W. H. Freeman and Company, (1983).

[12] E. Dahmen, K. Okeya, T. Takagi and C. Vuillaume. Digital signatures out of second-preimage resistant hash functions. In: J. Buchmann and J. Ding (eds.), Post-Quantum Cryptography, LNCS 5299, pp. 109-123. Springer (2008).

[13] W. Diffie and M. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, 22(6), 644-654, (1976).

[14] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In Andrew M. Odlyzko, (ed.), Advances in Cryptology — CRYPTO 86, LNCS 263, pp. 104-110. Springer (1987).

[15] O. Goldreich. Foundations of cryptography: Volume 2, basic applications. Cambridge University Press, Cambridge, UK, (2004).

[16] L. K. Grover. A fast quantum mechanical algorithm for database search. Proceedings, 28th Annual ACM Symposium on the Theory of Computing, pp. 212-219, (1996).

[17] A. Hülsing. Practical forward secure signatures using minimal security assumptions. PhD thesis, TU Darmstadt (2013).

[18] A. Hülsing, C. Busold and J. Buchmann. Forward secure signatures on smart cards. In: L. R. Knudsen and H. Wu. (eds.), SAC 2012. LNCS 7707, pp. 66-80. Springer (2013).

[19] A. Hülsing. W-OTS+ — Shorter signatures for hash-based signature schemes. In: A. Youssef, A. Nitaj, A.E. Hassanien. (eds.) AFRICACRYPT 2013. LNCS 7918, pp. 173-188. Springer (2013).

[20] A. Hülsing, L. Rausch and J. Buchmann. Optimal parameters for XMSS$^{\mathrm{MT}}$. In: A. Cuzzocrea, C. Kittl, D.E. Simos, E. Weippl and L. Xu. (eds.) Security Engineering and Intelligence Informatics. LNCS 8128, pp. 194-208. Springer (2013).

[21] A. Hülsing, J. Rijneveld and F. Song. Mitigating multi-target attacks in hash-based signatures. In: C. Cheng, Ka. Chung, G. Persiano and B. Yang (eds.) PKC 2016. LNCS 9614, pp. 387-416. Springer (2016).

[22] J. Katz. Analysis of a proposed hash-based signature standard. In: L. Chen, D. McGrew and C. Mitchell. (eds.) SSR 2016, LNCS 10074, pp. 261-273. Springer (2016).

[23] F. T. Leighton and S. Micali. Large provably fast and secure digital signature schemes based on secure hash functions. US Patent 5,432,852, July 11, (1995).

[24] L. Lamport. Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, (1979).

[25] D. McGrew, P. Kampanakis, S. Fluhrer, S. Gazdag, D. Butin and J. Buchmann. State management for hash-based signatures. In: L. Chen, D. McGrew and C. Mitchell. (eds.) SSR 2016. LNCS 10074, pp. 244-260. Springer, (2016).

[26] S. Moritz, A mixed integer approach for the transient case of gas network optimization. PhD thesis, TU Darmstadt (2007).

[27] M. Raab and A. Steger. Ball into bins — a simple and tight analysis. In: M. Luby, J. Rolim and M. Serna (eds.) RANDOM'98. LNCS 1518, pp. 159-170, Springer (1998).

[28] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In: L. Batten and J. Seberry. (eds.) Information Security and Privacy 2002. LNCS 2384, pp. 1-47. Springer (2002).

[29] D. McGrew, M. Curcio and S. Fluhrer. Hash-based signatures. Internet Draft draft-mcgrew-hash-sigs-05, October 31, (2016). "work in progress"

[30] R. C. Merkle. Secrecy, authentication, and public key systems. PhD thesis, Department of Electrical Engineering, Stanford University (1979).

[31] R. C. Merkle. A certified digital signature. Advances in Cryptology - CRYPTO'89. LNCS 435, pp. 218-238, Springer (1989).

[32] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In P. Samarati. (ed), Eighth ACM Conference on Computer and Communication Security, pp. 28-37, (2001).

[33] http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/pqcrypto-2016-presentation.pdf

[34] R. L. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), pp. 120-126, (1978).

[35] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26, pp. 1484-1509 (1997).

[36] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. Journal of Cryptology 12, pp. 1-28 (1999).