Design and implementation of a post-quantum hash-based cryptographic signature scheme

Guillaume Endignoux

School of Computer and Communication Sciences

Master's Thesis

July 2017

Responsible
Prof. Serge Vaudenay
EPFL / LASEC

Supervisor
Dr. Jean-Philippe Aumasson
Kudelski Security

Acknowledgements

I first want to thank my supervisor Jean-Philippe Aumasson for taking me on board for this project, and for his insightful comments throughout the semester.

I also thank my colleagues at Kudelski Security, for welcoming me in their office. I learned a lot from our discussions about cryptography, security and many other topics. And I really enjoyed teaming up with some of you for occasional CTFs.

More personally, I don't forget those who brought me support during my master in Lausanne. My family, for helping me to move to this new city. My friends, who came to visit Switzerland and explore steep mountains, discover new cheese and chocolate with me. And my new friends from all over the world that I met here, for making this a wonderful international experience.

Abstract

Digital signatures are essential asymmetric primitives of modern cryptography. Most security protocols rely on them at some point, to authenticate a peer or ensure authenticity and integrity of a document. Yet, signature schemes widely deployed today would be easy to break with a large-scale quantum computer, due to Shor's algorithms. Hence, a significant line of research is focusing on the design of post-quantum cryptographic schemes, and the National Institute of Standards and Technology (NIST) has opened a call for proposals to standardize post-quantum cryptography.

In this master thesis, we focus on signature schemes based on hash functions. They rely on simple assumptions on hash functions, such as preimage or collision resistance, and their security is generally well-understood, thanks to simple security reductions to hash functions properties. However, their signatures are still somewhat large, and the signature size grows with the planned number of messages issued by a key pair. Also, the simplest and most efficient hash-based schemes are *stateful*, which raises the issue of maintaining the state, something non-trivial in practical scenarios.

Therefore, the goal of this project was to improve the efficiency of stateless hash-based signatures. Towards this end, we propose new constructions and improve existing ones. We also present a cryptanalysis of the *subset-resilience* problem, showing new attacks and proposing fixes. We then review security reductions for hash-based signature schemes. Last, we outline a specification for a new stateless scheme that builds upon our analysis, and will be the basis for a proposal to NIST.

Notations

Acronyms

EU-CMA	existential unforgeability under chosen-message attacks
FTS	few-time signature
HORS	hash to obtain a random subset
KiB	2^{10} bytes
LWOTS	Winternitz one-time signature with L-tree
SHA	secure hash algorithm
SIMD	single instruction multiple data
MiB	2^{20} bytes
NIST	National Institute of Standards and Technology
ORS	obtain a random subset
OTS	one-time signature
PORS	PRNG to obtain a random subset
PRF	pseudo-random function family
PRNG	pseudo-random number generator
SU-CMA	strong unforgeability under chosen-message attacks
WOTS	Winternitz one-time signature

Symbols

$\emptyset$	empty set
$\perp$	invalid element, to signal an error
$\mathbb{N}$	set of non-negative integers
A^*	$\bigcup_{n=0}^{\infty} A^n$

A^+	$\bigcup_{n=1}^{\infty} A^n$
$A^{\leq m}$	$\bigcup_{n=0}^{m} A^n$
$A \setminus B$	set difference of A and B, $\{a \in A \vert a \notin B\}$
$(A \to B)$	set of functions from A to B
$\mathcal{P}_k(A)$	subsets of A containing at most k elements
B_n	set of n-bit strings $\{0,1\}^n$
$\vert x \vert$	bit length of string x
$x \vert\vert y$	concatenation of strings x and y
$\lceil x \rfloor_n$	n-bit encoding of integer $0 \leq x < 2^n$
$\oplus$	bitwise XOR
$\wedge$	logical AND
$\vee$	logical OR
$\lfloor x \rfloor$	floor, equal to $\max\{n \in \mathbb{N} \vert n \leq x\}$
$\lceil x \rceil$	ceil, equal to $\min\{n \in \mathbb{N} \vert n \geq x\}$
π	a well-known mathematical constant $\int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}}$
$\binom{n}{k}$	binomial coefficient $\frac{n!}{k!(n-k)!}$
$\left\{ {n \atop k} \right\}$	Stirling number of the second kind $\frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$
$T_k(x)$	Touchard polynomial $\sum_{i=0}^{k} \left\{ {k \atop i} \right\} x^i$
$f(n) = O(g(n))$	$\exists A \in \mathbb{N}, \forall n \in \mathbb{N}, f(n) \leq Ag(n)$
$f(n) = \Theta(g(n))$	$f(n) = O(g(n)) \wedge g(n) = O(f(n))$
$\mathsf{Pr}[A]$	probability of event A
$\mathsf{Pr}[A \vert B]$	probability of event A conditioned to event B
$\leftarrow$	affectation
$\xleftarrow{\$}$	randomized affectation following uniform distribution
$\mathsf{sorted}(\ldots)$	sequence of integers sorted in increasing order
$\mathsf{unique}(\ldots)$	remove duplicate values from a sequence
$\mathsf{unique}_k(\ldots)$	first k distinct values of a sequence
$\mathsf{substr}(x, i, n)$	substring of x of length n bits starting at bit index $0 \leq i < \vert x \vert$

Contents

Introduction

Digital signatures are an essential primitive of modern cryptography, as a means to ensure authenticity of documents and messages. A signature scheme is an asymmetric primitive that allows a *signer* to generate a pair of public and secret keys, distribute the public key to other users, and later issue signatures of messages. These signatures are generated with the secret key and can be verified with the public key.

Signatures are ubiquitous in security protocols, as a means to authenticate communicating parties (TLS, SSH), deploy a public-key infrastructure (certificates), ensure integrity of packages in software stores, etc. Today, widely used signature schemes rely on the hardness of mathematical problems such as factoring or discrete logarithm in certain groups. Yet, if one could build a large-scale quantum computer, these problems would be easy to solve thanks to Shor's algorithms [Sho94]. Even though such a large-scale quantum computer does not exist yet, this is an active research field in physics. Hence, it seems reasonable for cryptographers to research *post-quantum* cryptographic schemes – which would be secure even against quantum computers. In this context, the National Institute of Standards and Technology (NIST) has recently opened a call for proposals for post-quantum cryptography standardization [NIS16].

Several methods were proposed to design post-quantum signature schemes: based on lattices, hash functions, multivariate equations, error-correcting codes. In this project, we focus on hash-based schemes. Their main advantage is that they rely on simple assumptions on hash functions, such as collision or second-preimage resistance, instead of a specific algebraic structure. In particular, if an attack is discovered in a hash function, one can replace it by another function without modifying the overall structure of the scheme. Most hash-based schemes also come with relatively simple proofs of security reductions to the hash function's properties. Their main drawback is signature size, which typically grows with the number of messages signed by a key pair. For this reason, a significant line of research focused on improving their efficiency.

Besides, the simplest hash-based schemes are *stateful*, which means that a signer must maintain a state that is modified every time a signature is issued. This requirement can be a burden because trivial forgeries become possible if it is violated once, e.g. if two signatures are issued in the same state. Stateful schemes must therefore guarantee that this kind of misuse will not happen, which can be non-trivial for practical systems. On can think of rolling back the state of a machine after a crash, cloning virtual machines, or maintaining a pool of signing machines working in parallel. Hence, an alternative design goal is to construct *stateless* signatures schemes, and the recent proposal SPHINCS [BHH$^+$15] is a practical example. Yet, it is much less efficient than stateful signature schemes of the XMSS family [BDH11, HRB13, HRS16].

In this project, we explore ways to improve the efficiency of stateless hash-based signatures, attempting to bridge the gap with stateful schemes. Towards this end, we propose new constructions and improve existing ones. We present a cryptanalysis of the *subset-resilience* problem and review security reductions for hash-based signature schemes. We then outline a specification for a new stateless scheme, improving over SPHINCS thanks to our analyses. It will be the basis for a proposal to NIST.

Organisation of this report

This master thesis is organized as follows. In Chapter 1, we recall general definitions about signature schemes and hash functions, and review the effect of quantum computers against hash functions. In Chapter 2, we review the history of hash-based signature schemes. In Chapter 3, we summarize our improvements proposed and studied during this project to increase the efficiency of stateless hash-based signatures. In Chapter 4, we propose an analysis of the *subset resilience* problem, showing that the textbook version of HORS [RR02] is broken due to adaptive attacks. We also propose a new PORST construction, improving over HORST to increase its security level. In Chapter 5, we propose *octopus authentication*, an improved method to authenticate several leaves in a Merkle tree, and quantify the associated decrease in signature size. In Chapter 6, we explain why XOR masks are not necessary against post-quantum adversaries, and propose clarified security proofs with reductions to collision resistance. In Chapter 7, we propose a specification for a new hash-based signature scheme that we call Gravity, building upon our previous analyses.

Chapter 1

Preliminaries

1.1 General definitions

In this section we recall common definitions and security notions about signature schemes, hash functions and related primitives.

1.1.1 Notations

Given a positive integer n, we denote by $B_n = \{0,1\}^n$ the set of n-bit strings. We let $B_n^* = \bigcup_{i\in\mathbb{N}} B_n^i$ and $B_n^+ = \bigcup_{i\in\mathbb{N}\setminus\{0\}} B_n^i$. Given parameters k and t, we let $T = \{0,\ldots,t-1\}$ and we denote by $\mathcal{P}_k(T)$ the subsets of T of size at most k. Given two sets A and B, we denote by $(A \to B)$ the set of all functions from A to B.

We define security properties in terms of adversarial advantage, i.e. success probability of an adversary. Given a security property P on a scheme $\mathcal{S}$, and an adversary $\mathcal{A}$ against P, we denote by $\textsc{Succ}^{\mathrm{P}}_{\mathcal{S}}(\mathcal{A})$ the advantage of $\mathcal{A}$ against property P of $\mathcal{S}$. Given a set of resources ξ (time, number of queries, etc.), we define the insecurity of $\mathcal{S}$ for P as:

$$\textsc{InSec}^{\mathrm{P}}(\mathcal{S};\xi) = \max_{|\mathcal{A}|\leq\xi}\{\textsc{Succ}^{\mathrm{P}}_{\mathcal{S}}(\mathcal{A})\}$$

where $|\mathcal{A}| \leq \xi$ means that adversary $\mathcal{A}$ uses at most resources ξ.

We formalize security properties with games, following the framework of Bellare and Rogaway [BR06]. A game contains an **Initialize** procedure, a **Finalize** procedure and game-specific oracle procedures. An adversary $\mathcal{A}$ *playing* a game G corresponds to the sequential execution of the **Initialize** procedure, the algorithm $\mathcal{A}$ (with calls to the oracle procedures) and the **Finalize** procedure with the adversary's output as input. Given some scheme Π (that gives a concrete implementation for G), we denote by $\mathsf{Pr}[\mathcal{A}^G_\Pi \Rightarrow b]$ the probability that b is the output of $\mathcal{A}$ playing G.

In algorithms, we denote by $x \leftarrow y$ the affectation of value y to variable x. We denote by $x \xleftarrow{\$} X$ the affectation to x of an element chosen uniformly at random from the finite set X.

1.1.2 Signatures schemes

Definition 1 (Stateless signature scheme)**.** *Given a message space $\mathcal{M}$ (e.g. B_n or $\{0,1\}^*$), a public key space $\mathcal{PK}$, a secret key space $\mathcal{SK}$, and a signature space $\mathcal{SG}$, a stateless signature scheme is a triplet $(\mathcal{KG},\mathcal{S},\mathcal{V})$ where:*

- $\mathcal{KG} : \{1\}^* \to \mathcal{PK} \times \mathcal{SK}$ *is a randomized key generation function, that takes as input a security parameter* $n \in \mathbb{N}$ *in unary notation* 1^n*;*
- $\mathcal{S} : \mathcal{SK} \times \mathcal{M} \to \mathcal{SG}$ *is a deterministic signing function;*
- $\mathcal{V} : \mathcal{PK} \times \mathcal{M} \times \mathcal{SG} \to \{0, 1\}$ *is a deterministic verification function.*

We require the following property:

- *(correctness) if a key pair* $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$ *is the result of a call to* $\mathcal{KG}(1^n)$*, then for all* $M \in \mathcal{M}$*,* $\mathcal{V}(pk, M, \mathcal{S}(sk, M)) = 1$*.*

For convenience, we also define what we call *extractable* signature schemes, for which the verification function takes a particular form. Extractable schemes will be useful for constructions such as Merkle trees.

Definition 2 (Extractable signature scheme)**.** *A stateless signature scheme is* extractable *if there is an efficient (i.e. polynomial time in* n*) deterministic extraction function* $\mathcal{E} : \mathcal{M} \times \mathcal{SG} \to \mathcal{PK}$ *which extracts the public key from a message and signature, such that for all* $pk \in \mathcal{PK}$*,* $M \in \mathcal{M}$ *and* $\sigma \in \mathcal{SG}$*:*

$$\mathcal{V}(pk, M, \sigma) = \begin{cases} 1 & \text{if } \mathcal{E}(M, \sigma) = pk \\ 0 & \text{otherwise} \end{cases}$$

Remark Not all signature schemes are extractable: there may be several public keys that generate the same message-signature pair, and even if extraction is well-defined it is not always efficient. For example, there is no obvious way to efficiently extract an RSA public key in the general case – although in practice the public exponent is often chosen to be a small value. However, any signature scheme can be turned into an extractable scheme by appending the public key to each signature. Yet, this overhead is unnecessary for many hash-based schemes, hence we make an explicit distinction.

We also define the following variants of signature schemes.

Definition 3 (Indexed signature scheme)**.** *Given an index space* $\mathcal{I}$*, an indexed signature scheme is a signature scheme for which signing and verification take an index as additional argument, i.e.* $\mathcal{S} : \mathcal{SK} \times \mathcal{I} \times \mathcal{M} \to \mathcal{SG}$ *and* $\mathcal{V} : \mathcal{PK} \times \mathcal{I} \times \mathcal{M} \times \mathcal{SG} \to \{0, 1\}$*. The scheme can also be extractable, with an indexed extraction function* $\mathcal{E} : \mathcal{I} \times \mathcal{M} \times \mathcal{SG} \to \mathcal{PK}$*.*

Correctness requires that if a key pair $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$ *is the result of a call to* $\mathcal{KG}(1^n)$*, then for all* $i \in \mathcal{I}$*, and all* $M \in \mathcal{M}$*:*

$$\mathcal{V}(pk, i, M, \mathcal{S}(sk, i, M)) = 1$$

Definition 4 (Batch signature scheme)**.** *Given a batch size* N*, a batch signature scheme is a signature scheme that signs* N *messages at once, i.e. the signing function is* $\mathcal{S} : \mathcal{SK} \times \mathcal{M}^N \to \mathcal{SG}^N$*. Verification is unchanged, i.e. each message is verified individually. The scheme can also be extractable.*

Correctness requires that if a key pair $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$ *is the result of a call to* $\mathcal{KG}(1^n)$*, then for all* $i \in \{1, \ldots, N\}$*, and all* $(M_1, \ldots, M_N) \in \mathcal{M}^N$*:*

$$\mathcal{V}(pk, M_i, \mathcal{S}(sk, (M_1, \ldots, M_N))_i) = 1$$

We now recall security notions for signature schemes. We start with existential unforgeability, against which an adversary tries to forge the signature of a message that was not signed.

Definition 5 (EU-CMA)**.** *The advantage of an adversary $\mathcal{A}$ against existential unforgeability under chosen message attacks of $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ is:*

$$\textsc{Succ}^{\text{EU-CMA}}_{(\mathcal{KG},\mathcal{S},\mathcal{V})}(\mathcal{A}) = \mathsf{Pr}[(pk, sk) \xleftarrow{\$} \mathcal{KG}(1^n); (M, \sigma) \leftarrow \mathcal{A}^{\mathcal{S}(sk,\cdot)}(pk) : \mathcal{V}(pk, M, \sigma) = 1]$$

with the restriction that the message M output by $\mathcal{A}$ was not queried to the signing oracle $\mathcal{S}$.

Against strong unforgeability, an adversary can also propose an alternative signature for a message that was already signed.

Definition 6 (SU-CMA)**.** *The advantage of an adversary $\mathcal{A}$ against strong unforgeability under chosen message attacks of $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ is:*

$$\textsc{Succ}^{\text{SU-CMA}}_{(\mathcal{KG},\mathcal{S},\mathcal{V})}(\mathcal{A}) = \mathsf{Pr}[(pk, sk) \xleftarrow{\$} \mathcal{KG}(1^n); (M, \sigma) \leftarrow \mathcal{A}^{\mathcal{S}(sk,\cdot)}(pk) : \mathcal{V}(pk, M, \sigma) = 1]$$

with the restriction that the pair (M, σ) output by $\mathcal{A}$ is such that the signature σ was not the result of a query of the message M to the signing oracle $\mathcal{S}$.

1.1.3 Hash functions

Definition 7 (Hash function family)**.** *Given an integer security parameter n, an integer key parameter k, an integer input parameter m, a hash function family $\mathcal{F}_n$ is a set of functions from $\{0,1\}^m$ to $\{0,1\}^n$ indexed by the key space $\{0,1\}^k$. In other words, $\mathcal{F}_n = \{F_K : \{0,1\}^m \to \{0,1\}^n | K \in \{0,1\}^k\}$.*

Remark In practice, hash functions such as MD5, SHA-1, SHA-256 have no explicit *key*. However, as mentioned by Bellare and Rogaway [BR97, Section 2], some security notions such as collision resistance are meaningless without a key. Indeed, an adversary could simply hardcode the result (i.e. a collision) in its algorithm, obtaining a large advantage without doing much work. With the key, such trivial adversaries are inefficient because they would need to hardcode 2^k results in their algorithm, so there is hope that a secure hash function family exists. In practical terms [BR97, Remark 5.2], one can think of a keyless hash function like SHA-256 as an instance F_K where the key K was implicitly sampled by its creators. This fundamental problem of "human ignorance" was formalized by Rogaway in [Rog06].

We now give security properties for hash functions. A hash function family is one-way if it is hard to retrieve a preimage.

Definition 8 (OW)**.** *The advantage of an adversary $\mathcal{A}$ against one-wayness of $\mathcal{F}_n$ is:*

$$\textsc{Succ}^{\text{OW}}_{\mathcal{F}_n}(\mathcal{A}) = \mathsf{Pr}[K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m; h \leftarrow F_K(M); M' \leftarrow \mathcal{A}(K, h) : F_K(M') = h]$$

A hash function family is second-preimage-resistant if it is hard to retrieve a distinct preimage, knowing one preimage.

Definition 9 (SPR)**.** *The advantage of an adversary $\mathcal{A}$ against second-preimage resistance of $\mathcal{F}_n$ is:*

$$\textsc{Succ}^{\text{SPR}}_{\mathcal{F}_n}(\mathcal{A}) = \mathsf{Pr}[K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m; M' \leftarrow \mathcal{A}(K,M) : M' \neq M \wedge F_K(M') = F_K(M)]$$

A hash function family is collision-resistant if it is hard to find two messages with the same image.

Definition 10 (CR)**.** *The advantage of an adversary $\mathcal{A}$ against collision resistance of $\mathcal{F}_n$ is:*

$$\textsc{Succ}^{\text{CR}}_{\mathcal{F}_n}(\mathcal{A}) = \mathsf{Pr}[K \xleftarrow{\$} \{0,1\}^k; (M_1,M_2) \leftarrow \mathcal{A}(K) : M_1 \neq M_2 \wedge F_K(M_1) = F_K(M_2)]$$

A hash function family is undetectable if it behaves as if outputting random bits, under a randomly selected key.

Definition 11 (UD)**.** *The advantage of an adversary $\mathcal{A}$ against undetectability of $\mathcal{F}_n$ is:*

$$\textsc{Succ}^{\text{UD}}_{\mathcal{F}_n}(\mathcal{A}) = \left|\mathsf{Pr}[\mathcal{A}^{\mathcal{D}_{\text{UD},\mathcal{F}_n}} = 1] - \mathsf{Pr}[\mathcal{A}^{\mathcal{D}_{\text{UD},\mathcal{U}}} = 1]\right|$$

where $\mathcal{D}_{\text{UD},\mathcal{F}_n}$ and $\mathcal{D}_{\text{UD},\mathcal{U}}$ are distributions over $\{0,1\}^k \times \{0,1\}^n$ defined as follows. To sample from $\mathcal{D}_{\text{UD},\mathcal{F}_n}$, sample a key $K \xleftarrow{\$} \{0,1\}^k$, a message $M \xleftarrow{\$} \{0,1\}^m$ and output $(K, F_K(M))$. To sample from $\mathcal{D}_{\text{UD},\mathcal{U}}$, sample a key $K \xleftarrow{\$} \{0,1\}^k$, a bit string $U \xleftarrow{\$} \{0,1\}^n$ and output (K,U).

1.1.4 Pseudo-random functions

A function family $\mathcal{G}_n = \{G_K : \{0,1\}^m \to \{0,1\}^p | K \in \{0,1\}^n\}$ is pseudo-random if it is hard to distinguish from a function uniformly sampled in the space of functions from $\{0,1\}^m$ to $\{0,1\}^p$.

Definition 12 (PRF)**.** *The advantage of an adversary $\mathcal{A}$ against pseudo-randomness of $\mathcal{G}_n$ is:*

$$\textsc{Succ}^{\text{PRF}}_{\mathcal{G}_n}(\mathcal{A}) = \left|\mathsf{Pr}[K \xleftarrow{\$} \{0,1\}^n : \mathcal{A}^{G_K(\cdot)} = 1] - \mathsf{Pr}[F \xleftarrow{\$} (\{0,1\}^m \to \{0,1\}^p) : \mathcal{A}^{F(\cdot)} = 1]\right|$$

1.2 Impact of quantum computers on hash functions

Given that currently deployed signature schemes (DSA, ECDSA) are much more efficient than hash-based signature schemes, the latter would mostly be useful in a post-quantum setting – unless an efficient classical attack against (EC)DSA is discovered. Hence, it is important to have in mind the complexity of generic quantum attacks against hash functions. We consider a hash function family $\mathcal{F}_n$ that outputs n bits.

In the classical world, generic attacks against (second-)preimage resistance have a complexity of $\Theta(2^n)$ (i.e. brute-force guessing the preimage), and generic attacks against collision resistance have a complexity of $\Theta(2^{n/2})$ [vOW94] (due to the birthday paradox). For this reason, hash-based schemes have often tried to rely only on (second-)preimage resistance of the underlying hash function rather than collision resistance, as n can potentially be chosen twice smaller.

	classical	quantum
Preimage	$\Theta(2^n)$	$\Theta(2^{n/2})$
Second-preimage	$\Theta(2^n)$	$\Theta(2^{n/2})$
Collision	$\Theta(2^{n/2})$	$\Theta(2^{n/2})$

Table 1.1: Complexity of generic attacks against hash functions families with n bits of output.

In the post-quantum world, Grover's algorithm [Gro96] allows to find preimages with a complexity of $\Theta(2^{n/2})$. However, collision resistance is not affected: even though Brassard et al. [BHT98] proposed a quantum algorithm making $\Theta(2^{n/3})$ quantum queries, a finer analysis by Bernstein [Ber09] showed that this method would require so much hardware that the overall complexity wouldn't beat the classical algorithm of complexity $\Theta(2^{n/2})$. To summarize, assuming only generic attacks against $\mathcal{F}_n$, the post-quantum security of $\mathcal{F}_n$ is $n/2$ bits for preimage resistance as well as collision resistance (Table 1.1).

Remark For applications that consider only classical attacks in their threat model, constructions that rely only on (second-)preimage resistance may exist and be more efficient than constructions that require collision resistance. However, in this report we consider quantum attacks in our threat model.

Chapter 2

A history of hash-based signature schemes

In this section, we review the current state of hash-based signature schemes. The first such scheme proposed in 1976 allowed to sign a single bit with an asymmetric key pair [DH76]. Since then, several innovations have allowed to sign larger messages with a small public key, to scale to a practically unlimited number of messages signed by a single key pair, and to construct stateless schemes. We adopt a bottom-up approach, starting with the older building blocks, that are the basis of more recent and complex schemes.

2.1 Foundations of hash-based signatures

The most basic hash-based signature scheme was proposed by Diffie and Hellman upon an idea by Lamport [DH76]. Given a security parameter n and a one-way function $F : \{0,1\}^n \to \{0,1\}^n$, the following one-time scheme allows to sign a single bit. The secret key consists of two random values $x_0, x_1 \in \{0,1\}^n$; the associated public key is $(y_0, y_1) := (F(x_0), F(x_1))$. The signature σ for a bit b is the associated secret value $\sigma = x_b$; to verify the signature, one applies F to check that $y_b = F(\sigma)$.

From this simple scheme, the authors proposed to sign messages of m bits by using m instances of the 1-bit scheme. More precisely, the secret key consists of $2m$ random values x_b^i (for $b \in \{0,1\}$ and $1 \leq i \leq m$); the associated public key is $(y_b^i)_{b,i} := (F(x_b^i))_{b,i}$. The signature σ for an m-bit message $M = M_1 \dots M_m$ is then $\sigma = (x^i_{M_i})_i$, which can be verified by checking that $y^i_{M_i} = F(\sigma_i)$ for all i.

Arbitrary long messages

To sign messages of arbitrary length, Diffie and Hellman further proposed to use the *hash-then-sign* construction: a collision-resistant hash function H is applied to the message M to obtain a hash $h = H(M)$ of m bits, which is itself signed with the previous scheme, i.e. the signature σ consists of $\sigma = \mathsf{sign}(H(M))$.

However, if an attacker manages to find a collision $H(M_0) = H(M_1)$, they can then request a signature σ for M_0 and reuse σ as a forgery for M_1. To relax this requirement of

a collision-resistant hash function, Naor and Yung proposed *universal one-way hash functions* (UOWHF) [NY89], reformulated by Bellare and Rogaway as *target collision resistance* (TCR) [BR97]. In practice, one uses a keyed hash function family with a per-signature key K, and the signature is $\sigma = (\mathsf{sign}(K||H_K(M)), K)$, where the key K is randomly and uniformly chosen by the signer for every signature. This means that if an attacker finds a collision $H_{K_0}(M_0) = H_{K_1}(H_1)$ in an offline manner, they cannot apply it to attack the scheme because the key chosen by the signer is unlikely to be K_0. Instead, the adversary needs to break the TCR property of the hash function family, for which generic attacks are much slower than birthday attacks.

The need for more efficient schemes

This method already allows to sign arbitrary messages with minimal assumptions on the underlying functions, notably without relying on *trapdoors*. Yet it is still unpractical: keys and signatures are quite large and it can only be used one time. Indeed, if the scheme is used twice, an attacker requesting the all-zero and all-one messages (in a chosen message attack) can then forge any message, because they know all the secret values x_b^i. Despite this major drawback, many other signature schemes started from this building block and improved the idea to make it more efficient.

2.2 One-time signature (OTS) schemes

In this section, we review one-time signature schemes that were proposed to sign elements of a large message space, namely 128 or 256 bits (which is sufficient for messages of practically unlimited length thanks to the *hash-then-sign* construction). Although these schemes can only be used once, they are useful building blocks for practical many-time signature schemes.

2.2.1 Winternitz

The Winternitz one-time signature scheme (WOTS) was first proposed by Merkle [Mer89] following an idea of Winternitz. For a target message space of n bits, one chooses parameters ℓ and w such that $\ell \cdot \log_2 w = n$. The secret key consists of ℓ n-bit strings $(s_1, \ldots, s_\ell)$ and the public key is $(F^{w-1}(s_1), \ldots, F^{w-1}(s_\ell))$, that is a one-way function F iterated $w-1$ times on the secret key. We can view this construction as ℓ chains of iterated F, each chain being of length $w-1$. To sign a message x, decomposed into ℓ chunks of $\log_2 w$ bits $(x_1, \ldots, x_\ell)$, the signer issues $(F^{x_i}(s_i))_{1 \leq i \leq \ell}$. Verification is done by computing $F^{w-1-x_i}(y_i)$ for each signature element y_i, and comparing the result to the public key element $F^{w-1}(s_i)$.

However, this scheme is not secure in this form because given the signature of $x = (x_1, \ldots, x_\ell)$ one can easily forge a signature for any $y \geq x$ (by this notation we mean $\forall i, y_i \geq x_i$). The public key is also trivially the signature of the all-ones message. To cope with that, a checksum $C(x)$ is first appended to the message x to form $x' = x||C(x)$, and the signature is $\sigma = \mathsf{sign}(x')$. This checksum is computed as:

$$C(x) = \sum_{i=1}^{\ell} w - 1 - x_i$$

Due to this choice, $y > x$ implies that $C(y) < C(x)$, so that $y' \geq x'$ never holds for distinct x and y, hence avoiding the trivial forgery attack. The checksum consists of ℓ_C bits, resulting in keys and signatures of $\ell' = \ell + \ell_C$ values instead of ℓ, where:

$$\ell_C = \left\lfloor \frac{\log_2 \ell(w-1)}{\log_2 w} \right\rfloor + 1$$

Variants of the chaining function

Several variants were proposed for the chaining function F^i, to reduce the security of the scheme to various properties of F. We reuse the terminology of Hülsing [Hül13] to name these variants, i.e. WOTS$^{\text{CR}}$, WOTS$^{\text{PRF}}$ and WOTS^{+}.

The simplest variant is WOTS$^{\text{CR}}$ – appearing in Merkle's paper [Mer89] – for which the chaining function F^i is the i-th iteration of F: $F^0(x) = x$, $F^i = F(F^{i-1}(\cdot))$. Dods et al. [DSS05] have shown that WOTS$^{\text{CR}}$ is EU-CMA assuming that F is one-way, collision-resistant and undetectable.

In [BDE^{+}11], Duchmann et al. proposed WOTS$^{\text{PRF}}$, where F is replaced by a pseudo-random function family (PRF) $\mathcal{F}_n = \{F_K\}$. They proposed an unusual construction for the chaining function, where the output of the previous iteration is used as the PRF *key*: $F_K^0(\cdot) = k$, $F_K^i = F_{F_K^{i-1}(\cdot)}(\cdot)$. This leads to non-standard security notions, e.g. second key resistance or key collision resistance. They showed that WOTS$^{\text{PRF}}$ is SU-CMA assuming that $\mathcal{F}_n$ is second-key- or key-collision-resistant.

In [Hül13], Hülsing proposed WOTS^{+}, for which each iteration uses a random mask r_i, namely $F_K^0(x) = x$, $F_K^i = F_K(F_K^{i-1}(\cdot) \oplus r_i)$. The PRF key K and the masks $(r_1, \ldots, r_w)$ are part of the public key. Please note that the masks $(r_1, \ldots, r_w)$ are the same for all the chains, i.e. the chosen mask depends only on the iteration counter i. The advantage of WOTS^{+} is that is does not require $\mathcal{F}_n$ to be collision-resistant, but simply second-preimage resistant and pseudo-random (a proof of EU-CMA is given in [Hül13]).

Against classical adversaries, this relaxed requirement avoids birthday attacks, which is beneficial for performance, because smaller parameters offer the same level of security w.r.t. WOTS$^{\text{CR}}$. However, as we mentioned before, avoiding collision resistance is not worth the extra complexity against quantum adversaries. We study this in more details in Chapter 6.

2.2.2 Theoretical results on graph-based one-time signature schemes

Between 1994 and 1996, Bleichenbacher and Maurer published several theoretical results for one-time signature schemes based on directed acyclic graphs [BM94, BM96b, BM96a]. In these schemes, edges are one-way functions and vertices are inputs/outputs to these functions. A subset of these vertices is the secret key, another subset is the public key, and signatures are *cutsets* between the two. Each element in the message space is associated with a distinct cutset. For the signature scheme to be (computationally) unforgeable, the set of signatures must be an *antichain* of a suitable partially-ordered set (*poset*) over all minimal cutsets in the graph.

An interesting property is the size of the message space achievable for a given graph, which corresponds to the number of elements in a maximal antichain. For example, one may be interested in signing messages of 128 bits, in which case an antichain of 2^{128} elements

is necessary. Hence, Bleichenbacher and Maurer studied the maximal efficiency that can be obtained for such schemes, in terms of the message space size w.r.t. the number of nodes in the graph. They also conjectured that some graphs are maximally efficient in terms of the message space size w.r.t. the signature size [BM96a], namely that Winternitz-like one-time signatures [Mer89, DSS05, BDE+11, Hül13] are asymptotically optimal.

In 2002, Hevia and Micciancio [HM02] proposed a new formalism for such graph-based one-time signature (GBOTS) schemes. They considered graphs with two types of nodes: *expansion* and *compression* vertices, respectively corresponding to a length-doubling pseudo-random number generator (PRNG) and a length-halving hash function. They proved that the security of GBOTS schemes (existential forgery under one-chosen-message attack) can be reduced to usual properties of the underlying hash function and PRNG, namely non-invertibility, collision resistance and pseudo-randomness. In [DSS05], Dods et al. did a similar analysis to reduce the success probability of a forger to the undetectability, second-preimage resistance and collision resistance of the hash function.

An interesting perspective is that this formalism encompasses most proposed one-time signature schemes, among which Winternitz [Mer89, DSS05, BDE+11, Hül13], HORS [RR02], HORS++ [PWX03], HORST [BHH+15]. However, some of the latter schemes are also few-time signature schemes, which goes beyond the scope of Bleichenbacher-Maurer and Hevia-Micciancio works.

2.3 Few-time signature (FTS) schemes

Winternitz-like schemes are trivially insecure against two chosen messages attacks: with the signatures of the all-zero and all-one messages, an adversary can forge the signatures of all messages (although a recent work [BH16] has shown that it is still asymptotically secure for two messages in the model of *random message attacks*). On the other hand, so-called *few-time signature* (FTS) schemes were designed to be reusable a few times – typically a dozen – and still remain secure. They are also useful building blocks for many-time signature schemes.

2.3.1 Bins and Balls (BiBa)

In [Per01], Perrig proposed the "bins and balls" (BiBa) scheme. This scheme is parametrized by a number of bins n, a number of balls t, as well as integers m_1 and m_2. The secret key consists of t balls (also called "SEALs"), that is random m_2-bit values $(x_1, \ldots, x_t)$. The public key consists of t commitments to these values $(y_1, \ldots, y_t)$; e.g. the author proposed to use a PRF $\mathcal{F}_n$ as: $y_i = F_{x_i}(0)$.

To sign a message M, one first computes a hash $h = H(M)$. This hash allows to select a function $G_h : \{0,1\}^{m_2} \to \{0, \ldots, n-1\}$ among a hash function family $\mathcal{G}_n$. The signer then computes $g_i = G_h(x_i)$ for all the balls x_i, hoping to find a collision $g_i = g_j$. If this is the case, the signature σ consists of the secret values along with their indices $\sigma = (i, j, x_i, x_j)$, otherwise signature fails. To verify the signature, one computes $h = H(M)$ and checks that $G_h(x_i) = G_h(x_j)$ as well as the commitments to the secret values $y_i = F_{x_i}(0)$ and $y_j = F_{x_j}(0)$.

Thanks to the birthday paradox, the signing failure probability is $\approx \exp(-t(t-1)/2n)$, i.e. failure can be ruled out with high probability if t is large w.r.t. $\sqrt{n}$. On the other hand, an adversary knowing few secret values (from previous signatures) can forge a given message

with probability $1/n$, assuming they cannot invert $\mathcal{F}_n$. To increase the security level, Perrig proposed to use multi-way collisions instead of two-way collisions. If one requires a collision of k balls in the same bin, the forgery probability becomes $(1/n)^{k-1} \approx k!/2t^k$ for a constant failure probability of $1/2$. Perrig also analyzed how the scheme can be used a few times.

2.3.2 Subsets-based schemes

In 1979, Lamport [Lam79] proposed a new scheme, based on a message space $\mathcal{M}$, a security parameter n and an integer parameter k (e.g. $k = 20$). Given a one-way function $F : B_n \to B_n$ and a second-preimage resistant function $G : \mathcal{M} \to \mathcal{P}_k(2k)$, where $\mathcal{P}_k(2k)$ denotes the subsets of $\{1, \ldots, 2k\}$ of size k, the scheme works as follows. A secret consists of $2k$ values $(x_1, \ldots, x_{2k}) \in B_n^{2k}$; as usual the associated public key is the images of these keys by F, i.e. $(F(x_1), \ldots, F(x_{2k}))$. To sign a message $M \in \mathcal{M}$, one selects a subset of k indices as $\{i_1 < \cdots < i_k\} = G(M)$ and reveals the associated secrets $(x_{i_1}, \ldots, x_{i_k})$. The verifier computes the set of indices with G and verifies that the revealed keys are legitimate by applying F.

In this original version, Lamport proposed to construct G by first applying a one-way function $H : \mathcal{M} \to \{0,1\}^{2k}$ to obtain an $2k$-bit digest of the message, and then include the indices i such that $H(M)_i = 1$. However, this works only if $H(M)$ contains exactly k bits equal to 1, which occurs with a probability of $P_{succ} \approx 0.13$ for $k = 20$. Lamport proposed to modify the message (e.g. by means of a separate counter) until the signature succeeds – which takes an average of $1/P_{succ}$ trials – but admits that this construction is not satisfactory.

Bos and Chaum [BC92] improved this scheme by proposing a function G defined over the whole message space $\mathcal{M}$. They assumed a message space of the form $\mathcal{M} = \{0, \ldots, |\mathcal{M}| - 1\}$ with size $|\mathcal{M}| \leq \binom{2k}{k}$. As pointed out by Reyzin and Reyzin [RR02], they use the following equation to determine iteratively if an index i should be included in the subset $G(M)$:

$$\binom{j}{i} = \binom{j-1}{i-1} + \binom{j-1}{i}$$

More details about this subset-selection algorithm can be found in the corresponding papers [BC92, RR02].

These schemes can be viewed as an extension of the Lamport-Diffie-Hellman scheme [DH76] (that we described in Section 2.1), where instead of revealing one value for each of k pairs, the signer reveals k values chosen globally without constraint. For the same parameters, this allows to use a message space of size $\binom{2k}{k}$ instead of 2^k, that is messages of $\approx 2k - \frac{1}{2}\log_2(k\pi)$ bits instead of k bits.

An interesting question is whether we can further increase the size of the message space for a fixed k, by using a different family of subsets. According to Sperner's theorem [Spe28], $\binom{2k}{k}$ is the maximal number of subsets of $\{1, \ldots, k\}$ such that no subset contains another subset of the family, which means that this construction is optimal in this setting.

Hash to Obtain a Random Subset (HORS)

In [RR02], Reyzin and Reyzin proposed a more efficient algorithm to map an integer to a k-subset of $\{1, \ldots, 2k\}$, improving upon Bos and Chaum [BC92]. They also generalized the scheme to k-subsets of $\{1, \ldots, t\}$ where k can be much smaller than $t/2$, in order to reduce the size of signatures (at the expense of larger public keys).

They then proposed a new construction called *hash to obtain a random subset* (HORS). Instead of using an injective mapping $G : \mathcal{M} \to \mathcal{P}_k(t)$, they propose to use a hash function $H : \mathcal{M} \to \{0,1\}^{k\tau}$, where $t = 2^\tau$. To sign a message M, the hash $h = H(M)$ is split into k numbers of τ bits $h_1, \ldots, h_k$. These numbers form a subset $G(M) = \{h_1, \ldots, h_k\}$ of $\{1, \ldots, t\}$ and the signature consists of the secrets $(x_{h_1}, \ldots, x_{h_k})$. The subset $G(M)$ may contain redundant indices, but they argue that this is not a problem, and the mapping from messages to subsets is more straightforward to compute with this method. However, we will see in Chapter 4 that this potential redundancy is problematic.

Reyzin and Reyzin also note that this scheme allows to produce several signatures. In that case, security reduces to the hardness of the *subset-resilience problem*, and they estimate the security level for r signatures as $k(\log_2 t - \log_2 k - \log_2 r)$ bits for non-adaptive attacks. However, they did not study adaptive attacks – where the adversary can compute H in an offline manner – and we will see in Chapter 4 that these attacks are much more effective.

In [PWX03], Pieprzyk et al. proposed HORS++. Given a message space of size n, a secret key consisting of t values and a reuse parameter r, they propose to use an (n, t, r)-cover-free family (CFF) $\mathcal{X}$ to produce up to r signatures without loss of security. More precisely, a cover-free family $\mathcal{X} = \{X_i | 1 \leq i \leq n\}$ with $X_i \subseteq \{1, \ldots, t\}$ is such that for all $\Delta \subseteq \{1, \ldots, n\}$ with $|\Delta| = r$ and for all $i \notin \Delta$:

$$X_i \setminus \bigcup_{j \in \Delta} X_j \neq \emptyset$$

In other words, no set X_i is covered by r other sets of $\mathcal{X}$. To estimate the performance of the scheme, they use well-known results about the existence of CFFs depending on the parameters (n, t, r). They also propose instantiations of CFFs based on polynomials, error-correcting codes and algebraic curves.

It is worth noting that contrary to HORS, for which security degrades gradually with r, the maximal number of messages signable by an instance of HORS++ is fixed in advance. Indeed, HORS++ is secure up to r chosen-message signatures – assuming one-wayness of the function F used to compute public keys – but insecure for $r + 1$ chosen messages.

In [BHH+15], Bernstein et al. proposed HORST, a.k.a. HORS with trees. They noted that a HORS public key can be quite large – typically $t = 2^{16}$ values of $n = 256$ bits each – but only a few secret values are revealed in a signature – typically $k = 32$ values per signature. To reduce the combined size of a public key and a signature, they proposed to use a *Merkle tree* (see Section 2.4.1) to authenticate the public values. Hence, the public key consists of only one n-bit value (the root of the Merkle tree), and each signature includes k authentication paths (each of $\log_2 t$ n-bit hashes) along with the k secret values. They also proposed to cut the authentication paths before the root and instead include all Merkle tree nodes at level $x = 6$, to avoid repetitions between overlapping authentication paths and further reduce the signature size. In Chapter 5, we will present an improvement over this authentication method.

2.4 Many-time signature schemes

In practice, a signer wishes to create many more signatures. This is for example the case for a software store or a certificate authority. A practical target could be 2^{50} messages signed by a given public key: as pointed out by Bernstein et al. [BHH+15], this corresponds to

2^{20} messages per second for over 30 years. In the NIST call for proposals for post-quantum cryptography, the target is 2^{64} chosen messages [NIS16, Section 4.A.4].

2.4.1 Merkle Trees

The first way to create a many-time signature scheme out of a one-time signature scheme is to use the construction proposed by Merkle in 1989 [Mer89]. Given integers n, h and a hash function $H : \{0,1\}^{2n} \to \{0,1\}^n$, a so-called *Merkle tree* is a binary tree of height h whose nodes are each labelled with a value $x \in \{0,1\}^n$, such that the value of each internal node is computed as $x = H(y||z)$ where y and z are the values of the left and right children.

The root value r can first be sent to later authenticate compactly any of the 2^h leaf values $v_1, \ldots, v_{2^h}$. Indeed, to verify that a value v is at leaf index i, one simply needs v, i and the *authentication path* of i. This authentication path contains the siblings of all the nodes on the path between leaf i and the root (h values). This allows to recursively compute the values of internal nodes up to the root, and compare the result to r.

This construction allows to turn a one-time signature scheme into a many-time signature scheme as follows. Given 2^h instances of the OTS, the signer generates the Merkle tree whose leaf values are each the public key of an OTS instance. The overall public key is the root value. The i-th signature contains a signature generated by the i-th OTS instance, along with the authentication path of i.

Hence, the public key only contains n bits, compared to a naive approach with a public key of 2^h OTS public keys, at the expense of slightly larger signatures – h extra n-bit strings. However, key generation time is exponential in h, because the full Merkle tree needs to be computed at this stage. For example, $h = 20$ is possible, but may not be enough for all signers. Also, the signer needs to keep track of the indices i that have already been used, so the scheme is *stateful*.

Relaxing the collision resistance requirement

The original construction of Merkle trees required a collision-resistant hash function family. To relax this requirement, Dahmen et al. proposed the *second-preimage resistant (SPR) Merkle tree* construction (SPR-MSS), by introducing random masks that are XORed after each node of the tree [DOTV08]. These masks are part of the public key and allow to reduce the EU-CMA security of the scheme to the second-preimage resistance of the underlying hash function family.

This introduction of masks is similar to the WOTS$^+$ construction [Hül13], and as such we repeat our reservations. Even though birthday attacks are avoided classically, the extra complexity of masks may not be worth it against quantum adversaries. Also, these more elaborate constructions come at the expense of adding $2h$ masks to the public key for a tree of height h. And looking closely to the proofs of security, the security reduction degrades with a factor of 2^h w.r.t. the second-preimage resistance of the hash function – i.e. larger trees require more resistant hash functions – whereas reduction to collision resistance doesn't introduce this degradation factor.

XMSS [BDH11] is a variant of SPR-MSS, where (1) secret values are generated with a PRNG so that the secret key only consists of a small seed and (2) a so-called *L-tree* allows to use any number of bits per signature (not necessarily a power of two).

In their (now expired) patent [LM95], Leighton et al. further proposed to specialize each invocation of the hash function in the Merkle tree – and in the underlying OTS – by using a so-called "security string" consisting of an identifier I unique to the key pair and a unique address A of this invocation in the tree. In other words, the hash of a value V by a signer with identifier I at position A in the tree is computed as $H(V||I||A)$, instead of $H(V)$. With this construction, they claimed to reduce security to second-preimage resistance and to protect against worldwide attackers that try to forge a signature for any signer in the world, thereby allowing to use shorter hashes and reducing the size of signatures. Similar constructions were analyzed by Katz [Kat16]. Yet, we point out that $I||A$ is predictable and public, so the security reduction would rather be relaxed to "known-suffix collision attacks", which are still vulnerable to the birthday paradox. Besides, our reservations about second-preimage reduction also hold in this case.

Incremental algorithms to compute authentication paths

An important performance issue is the computation of authentication paths by the signer. Indeed, a naive algorithm needs to iterate over all the leaves to generate such a path, hence running in $\Theta(2^h)$ time. However, authentication paths for neighboring leaves share a lot in common, so a natural idea is to use leaves in sequential order with an incremental algorithm to update the authentication path.

In Merkle's original paper [Mer89], such an incremental algorithm was proposed, where h parallel processes compute each a layer of the path, ensuring a worst-case runtime of $\Theta(h)$ per signature. Further improvements were proposed to amortize the costs for stateful signature schemes that use leaves in sequential order [JLMS03, Szy04, BDS08]. For example, FMTseq [NSW05] a.k.a. "Fractal Merkle Tree sequential one-time signature" uses the fractal algorithm by Jakobsson et al. [JLMS03] to produce signatures; from the verifier's point-of-view it is a standard combination of a Merkle tree and WOTS.

2.4.2 Generic compositions

In [MMM02], Malkin et al. present two generic composition methods to create new signature schemes. Given two schemes Σ_0 and Σ_1 that can respectively sign T_0 and T_1 messages, the *sum* composition $\Sigma_0 \oplus \Sigma_1$ allows to sign $T_0 + T_1$ messages and the *product* composition $\Sigma_0 \otimes \Sigma_1$ allows to sign $T_0 \cdot T_1$ messages. They reduce the security of composed schemes to the security of the underlying schemes, in the standard model.

They further present these compositions in the context of *forward-secure* signature schemes (for which the leakage of the secret key at a given time period does not allow to forge messages from previous time periods), and show that they preserve this forward-security property.

They also present a scheme that can sign an "unbounded" number of messages (more precisely 2^n messages where n is the security parameter), and whose performance depends on the current "time period" (i.e. the number of messages signed so far).

2.4.3 Hyper trees

To overcome the limitation on the number of messages that can be signed with a Merkle tree – typically 2^{20} – new constructions were proposed that use several layers of Merkle trees. Essentially, one builds a chain of Merkle trees to increase the capacity of the scheme (i.e.

the number of messages that can be signed by a single key pair), in the spirit of the product composition of Malkin et al. [MMM02].

The first such construction was CMSS [BGD+06], which uses two layers of Merkle trees. A primary tree – whose root is the public key – signs the root of a secondary tree (with Winternitz OTS) and the secondary tree signs the messages. If both trees have a height $h \approx 20$, 2^h messages can be signed with the secondary tree, and 2^h trees can be signed by the primary tree, so 2^{2h} messages can be signed in total. To amortize computation costs, only one tree on each layer is created at key generation; subsequent secondary trees are each generated progressively as the previous one is used for signatures: one leaf of the new tree is constructed every time one leaf of the old tree is used. To reduce the memory requirements for the signer, leaves are generated with a PRNG, so that only a seed is stored in memory.

GMSS [BDK+07] (generalized Merkle signature scheme) extends this construction by using an arbitrary number of layers of trees. Here again, the public key is the root of the tree at the uppermost layer; the root of each other tree is signed by its parent tree by means of a Winternitz OTS; trees at the bottom layer are used to sign messages. The tree height and Winternitz parameter need not be the same at all layers, and the authors propose several instantiations of these parameters, optimizing for size or speed. They also propose various signature capacities, namely 2^{40} or 2^{80} messages per key pair.

Similarly, XMSS$^{\mathrm{MT}}$ [HRB13] is a multi-tree variant of XMSS [BDH11]. An overview of the EU-CMA security proof is given, relying on [MMM02, BDH11, HBB12]. The authors also formalize selection of parameters as a linear optimization problem and give an optimal choice of parameters to instantiate the scheme.

XMSS-T [HRS16] is another variant aimed at mitigating *multi-target* attacks against the underlying hash function. Indeed, the same hash function is used many times in a signature scheme such as XMSS, and an attacker could take advantage of finding a (second-)preimage for any of these instances to break the signature scheme. XMSS-T mitigates such attacks by keying each instance of the hash function: each node in the hyper tree is given a public and unique address, from which a hash key is derived.

2.5 Stateless signatures

Although the previous schemes already allow to sign a practically unlimited number of messages at a reasonable cost, they are stateful, in the sense that the signer must update its internal state for each signature, for example by using Merkle tree leaves sequentially. This can be a limitation because the signer must properly manage the state – having several messages signed in the same state opens the door to trivial forgeries. For example, if one wishes to sign messages concurrently with several signing machines, the state must be synchronized over these machines. To overcome this limitation, several *stateless* signature schemes were proposed.

2.5.1 Goldreich's construction

In [Gol04, Section 6.4.2], Goldreich presented a construction based on a binary tree of one-time signatures. In this construction, each node corresponds to an OTS instance. Each internal node authenticates the public keys of its two children by means of the OTS, i.e. the OTS of

node i signs $(pk_{left(i)}, pk_{right(i)})$ with sk_i. The overall public key is the OTS public key of the root node. Leaf nodes use their OTS to sign messages.

More precisely, each signature consists of a path $\rho = (\rho_1, \ldots, \rho_h)$ (where ρ_i is a bit meaning *left* or *right*), authentications A_i of each node on this path and signature σ of the message by the leaf of the path.

$$A_i = \mathsf{sign}(sk_{\rho_i}, (pk_{left(\rho_i)}, pk_{right(\rho_i)}))$$

$$\sigma = \mathsf{sign}(sk_{\rho_h}, M)$$

For the scheme to be secure, one must make sure that each OTS is effectively used at most once. This constraint is verified for internal nodes (which only sign the public keys of their children), but one needs to make sure that each leaf is used for at most one document. In other words, the path ρ must be different for every message. Goldreich proposed two ways to implement this constraint: either define ρ as a counter (but the associated scheme is stateful), or use a random ρ for every message. In the latter case, one must use a larger tree height h to avoid collisions, due to the birthday paradox.

Besides, Goldreich proposed to generate the tree on demand to avoid a huge computational and memory overhead for the signer. More precisely, the secret key only consists of a seed and each OTS instance is generated by a PRNG taking as argument this seed and the node index i in the tree. That way, only the root node needs to be computed at key generation time, and only the nodes on the path need to be computed for each signature. The PRNG is deterministic (for a given seed), which guarantees the consistency of the scheme (i.e. if a node is used by several signatures, the same OTS instance will be generated for this node every time).

2.5.2 SPHINCS

Although Goldreich's construction already provides a stateless signature scheme, it is highly inefficient: the typical signature size is more than 1 MiB, as pointed out in [BHH+15]. To overcome this limitation, the SPHINCS construction [BHH+15] introduces several improvements.

First, the OTS for leaves is replaced by a FTS, to increase the resilience to path collisions, hence reducing the overall height of the tree.

Second, intermediate nodes are replaced by Merkle trees that each sign 2^h children instead of 2, to form a fully-fledged hyper-tree structure. Although this modification increases signing time (each Merkle tree on the path needs to be fully generated for every signature), this reduces the size of signatures, because less OTS instances are included in a signature. Indeed, Winternitz OTS instances account for the majority of the signature size.

These two optimizations reduce the size of a signature to 41 KiB, with public and secret keys of 1 KiB, for parameters suitable for 128-bit security against quantum computers.

In Chapter 3, we will present new methods to further improve the performance of SPHINCS, leading to a new scheme that we call Gravity and formalize in Section 7.

2.6 Efficient implementations of hash functions

Because hash-based signature schemes often use many evaluations of the underlying hash function to generate each signature, it is desirable to have fast implementations of hash functions. This often goes by leveraging CPU-specific instructions, such as Intel's support for *single-instruction multiple-data* (SIMD) instructions – notably for primitives based on the *add-rotate-xor* (ARX) paradigm (e.g. Salsa, ChaCha, BLAKE, BLAKE2) – and direct support for some cryptographic primitives (AES-NI). Some recent constructions were even proposed specifically to be used in hash-based signature schemes, such as Haraka [KLMR16].

SHA functions

The *secure hash algorithms* (SHA) are general-purpose hash functions standardized by the NIST. SHA-1 [sta02] is notoriously broken [WYY05] and a collision was recently disclosed by Stevens et al. [SBK+17]. SHA-2 and SHA-3 [Dwo15] are general-purpose hash functions that are mostly optimized for large messages. For example, the Keccak permutation in SHA-3 uses 1600 bits, which implies overhead when hashing small messages (e.g. 256 or 512 bits). Hence, SHA functions are not optimized for hash-based constructions such as Winternitz OTS and Merkle trees, that operate on small messages.

BLAKE

In [NA12], Neves and Aumasson presented an optimized implementation of the BLAKE hash function [AHMP08] with the AVX2 (Intel) and XOP (AMD) processor extensions. They explained how the new 256-bit SIMD instructions allow to compute ARX operations on several columns of the function's state in parallel. They considered both the BLAKE-256 and BLAKE-512 versions of the hash function, showing that the latter benefits more of the new extensions because 256-bit SIMD instructions naturally operate on rows of the function's state. They noted that XOP extensions are faster because they feature a native *rotate* instruction, whereas one must use two shifts and a XOR to simulate a rotate with non-XOP instruction sets (including AVX2).

BLAKE2b

BLAKE2 [ANWW13] is an improvement over BLAKE optimized for speed: the number of rounds is smaller thanks to a large security margin in BLAKE, the "big" version BLAKE2b is faster on 64-bit processors, and the choice of rotation constants allows better implementations on Intel processors with SIMD instructions (a *shuffle* can be used to implement rotation by a multiple of 8 bits). These choices make it one of the fastest hash functions for large inputs.

Haraka v2

In [KLMR16], Kölbl et al. presented Haraka v2, a hash function based on the AES permutation that supports only inputs of 256 and 512 bits, for outputs of 256 bits. This restriction of the domain allows a much simpler design than general-purpose hash functions, and optimized implementations on processors supporting AES-NI instructions. This makes it one of the

fastest hash functions on short inputs, and a good candidate to instantiate Merkle trees and Winternitz OTS.

The authors propose a 5-round version offering second-preimage resistance, but suggest that a 6-round version would be collision-resistant. The difference between versions 1 and 2 of Haraka is in the choice of constants, to avoid an attack by Jean that uses symmetries in the AES permutation [Jea16].

Recent benchmarks by Kölbl suggest that Haraka is the fastest function to instantiate SPHINCS, both on Intel and ARM CPUs [Kö17].

Simpira v2

Simpira [GM16] is another proposal based on AES. It is a family of permutations of variable size (a multiple of 128 bits), based on generalized Feistel structures (GFS). The authors of Simpira proposed various applications of these permutations, in particular one can construct hash functions with a Davies-Meyer feed-forward [MvOV96, Section 9.4.1].

Other constructions

In [RED+08], Rohde et al. proposed two hash functions with respectively 128 and 256 bits of output, based on the Matyas-Meyer-Oseas construction [MvOV96, Section 9.4.1] with a block cipher such as AES. They show that on small inputs, these functions are faster than general-purpose hash functions, because the block length is smaller. They also implemented the Merkle signature scheme based on these functions on an embedded platform (8-bit AVR microcontroller) and showed that signing time was comparable to ECDSA-160 and much faster than RSA-2048, and verification was much faster than ECDSA and RSA. They did not implement key generation on the microcontroller, as it was too costly.

However, their parameters are not suited for post-quantum security and they do not seem faster than the more recent Haraka. Indeed, Haraka reduces the number of rounds and applies the AES permutation directly without key schedule, whereas Rohde et al. use the full AES block cipher.

A not-so-efficient construction

In 1990, Rompel [Rom90] proposed a generic method to transform "one-way functions" into "one-way hash functions". In modern terms, this means constructing second-preimage-resistant functions from preimage-resistant functions. This work as often been cited as a proof that one-way functions are sufficient to create signature schemes.

Rompel's construction notably uses k-universal families of hash functions to operate the transformation. It is however not practical, as the complexity to compute such a one-way hash function is $O(n^8)$ where n is the number of bits in the output! For example, with $n = 256$ the complexity would be in the order of 2^{64}.

Chapter 3

Improvements for stateless hash-based signature schemes

SPHINCS [BHH+15] has shown that stateless hash-based signature schemes can be practical. Signing is reasonably fast, verification is even much faster, and signature sizes are reasonable yet still somewhat large compared to current pre-quantum signatures. SPHINCS signatures are 41 KiB large, whereas ECDSA signatures take 70 bytes[1] [Por13].

In this chapter, we propose improvements for stateless hash-based signature schemes. Our goal is mainly to improve signature and public key sizes compared to SPHINCS, for similar signing and verification times. We give further details for some of these improvements in the following chapters. Based on these improvements, we propose a new signature scheme called Gravity, for which we give a specification draft in Chapter 7.

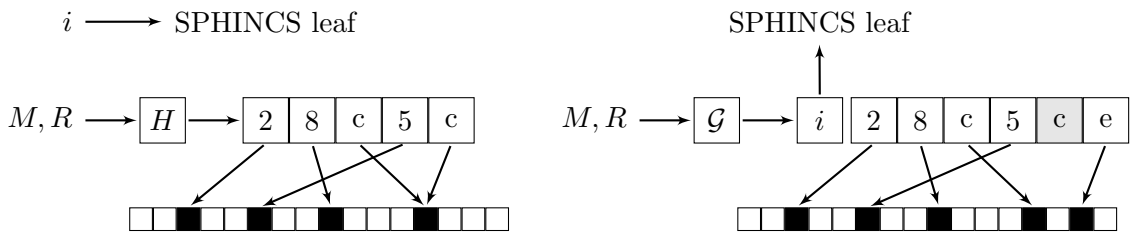
3.1 PORS or PRNG to obtain a random subset

As we have seen, state-of-the-art few-time signature schemes are based on the *hash to obtain a random subset* (HORS) construction. Yet, HORS was only partially studied, as only non-adaptive attacks were considered by Reyzin and Reyzin [RR02]. In particular, we will see in Chapter 4 that this textbook version of HORS is susceptible to adaptive attacks, that are made even worse by the simplicity of HORS: take the output of a hash function and split it into blocks to obtain a set of indices. Indeed, nothing prevents some of these indices to collide, reducing the size of the obtained subset and decreasing the security.

Even though HORS shines by its simplicity and speed compared to more elaborate methods to obtain random subsets of guaranteed size, its speed is not critical in complex schemes such as SPHINCS, for which WOTS and Merkle trees dominate the computational cost. Hence, we propose a new construction using a PRNG to obtain a random subset, that we call PORS. Instead of using a hash function, we seed a PRNG from the message (and salt) and query it until we obtain a subset of k distinct indices (Figure 3.1). The computational overhead is equivalent to a few additional hash evaluations, for a significant security increase.

In the case of SPHINCS, we also noticed that adversaries have full control over the selected leaf in the hyper-tree. Instead, we propose to generate this leaf index with the PRNG, further

[1] over a 256-bit field and written in ASN.1 notation



(a) HORS. (b) PORS.

Figure 3.1: Comparison of the HORS and PORS constructions to obtain a random subset from a message M and a salt R. In HORS (left), the hash function output is split into τ-bit blocks that may collide. If the scheme is used in SPHINCS, the signer (or adversary!) provides the leaf index i. In PORS (right), a PRNG is used to produce as many τ-bit numbers as necessary, as well as the leaf index i if used in SPHINCS.

increasing the security level. This increased security margin allows to reduce the hyper-tree height by 2 layers of WOTS, saving 4616 bytes. More details and a security analysis are given in Chapter 4.

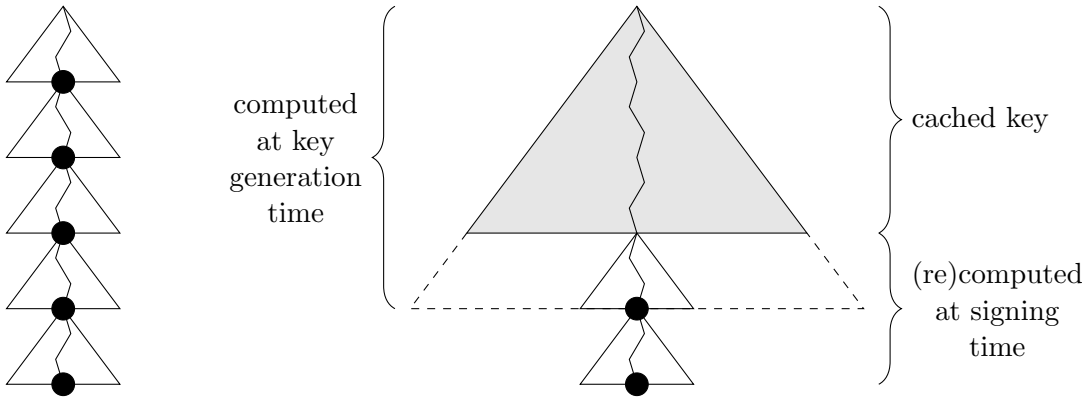
Remark In SPHINCS, the public salt R is computed by the signer as $\mathsf{hash}(\mathsf{salt}||M)$ for a secret salt. This means that if the message M is long, the signer needs to compute two long hashes: $R \leftarrow \mathsf{hash}(\mathsf{salt}||M)$ and the HORST subset as $H(R, M)$. Instead, with PORS we propose to compute a long hash $m \leftarrow \mathsf{hash}(M)$ and then two small hashes on m as $R \leftarrow H(\mathsf{salt}, m)$ and $\mathsf{seed} \leftarrow H(R, m)$ as a seed for the PRNG. This halves the computational overhead of signing long messages.

3.2 Secret key caching

XMSS [BDH11] is a signature scheme similar to SPHINCS, but with smaller signatures at the expense of being stateful. For example, the XMSS-T variant [HRS16] produces signatures of 8.8 KiB for a capacity of 2^{60} messages and 128 bits of quantum security.

The main difference is that the hyper-tree of SPHINCS is divided into many layers because these trees have to be fully recomputed on-the-fly for each signature. On the contrary, XMSS can benefit from efficient incremental algorithms to amortize the computational cost among many signatures [JLMS03, Szy04, BDS08]. Consequently, SPHINCS authors proposed to divide an hyper-tree of height 60 into 12 layers of Merkle trees, each of height 5, meaning that there are 12 WOTS signatures to connect these layers. Hence, most of the size of SPHINCS signatures is used by WOTS signatures, each containing $\ell = 67$ hash values, i.e. 2144 bytes per WOTS signature. In contrast, an authentication path in a Merkle tree of height 5 requires only 5 hash values, i.e. 160 bytes.

However, we point out that the root layer of SPHINCS contains only one tree, which is recomputed for every signature, independently of the selected path in the hyper-tree. Hence, it seems natural for the signer to cache it during key generation, to save computation time later. In that case, we can choose a much larger height for this root tree than the other layers, because the cost of key generation is amortized among many signatures (up to 2^{50}



(a) SPHINCS. (b) Secret key caching.

Figure 3.2: Secret key caching. Triangles represent Merkle trees, black circles represent WOTS signatures. In SPHINCS (left) the hyper-tree is simply made of equal-height trees. With secret key caching (right), a large root Merkle tree is computed at key generation (dashed triangle) and part of it is cached in the secret key (grey triangle). At signature time, the relevant lower part of this root tree is recomputed, along with lower layers in the hyper tree.

for SPHINCS), and in practice key generation does not have the same latency constraints as signing. For the same hyper-tree height, this reduces the number of layers, which means less WOTS per signature and smaller signatures.

For example, for a total hyper-tree height of 60, we can use a top layer of height 20, and 8 other layers of height 5, saving 3 WOTS instances. In the top tree, the signer can cache the first 15 levels (i.e. $2^{16} - 1$ hashes of 32 bytes) with 2 MiB of memory. Then, at signature time, the signer regenerates the 8 lower layers and the bottom 5 levels of the top layer (Figure 3.2). Compared to SPHINCS, this saves 201 hashes per signature, i.e. 6432 bytes. Besides, signature and verification are a little faster, due to less WOTS instances.

We point out that this method does not make the scheme stateful. Contrary to the state of XMSS, our cache is static and not modified by the signing process. Further, the cache can be recomputed on-demand from a small secret seed. This means that one can easily set-up new signing machines by sending them the secret seed; there is no need to send the full cache. Similarly, the cache need not be stored in persistent memory, it can be regenerated after a reboot, a machine reinstallation, etc.

Last, parameters are easy to adapt to the user's configuration. If the signing machine is not powerful enough – e.g. an embedded device with low memory and power – a smaller cache can be used. On the contrary, more powerful machines can use a larger cache to further reduce signature size.

3.3 Removing redundancy in authentication paths

A significant part of the size of a SPHINCS signature is taken by a single HORST instance, and in particular by $k = 32$ authentication paths in a HORST tree of height 16. SPHINCS already shortened these authentication paths from length 16 to length 10 by including all

nodes at level 6, as there is a lot of redundancy next to the root. In total, authentication paths contain 384 values, i.e. 12288 bytes. Yet, on average most of the nodes at level 6 can already be inferred from authentication paths, so there is still some redundancy. Besides, some authentication paths may merge even below this threshold, introducing even more redundancy.

Instead, we propose to use a dynamic strategy to include only necessary values, in what we call an *authentication octopus*. We study this construction in more details in Chapter 5, and show that we can reduce the authentication set to 352 values in the worst case, and 324.3 values on average, i.e. saving at least 1024 bytes and on average 1909 bytes, compared to SPHINCS.

Caveat We point out that an authentication octopus contains a variable number of values, depending on the set of indices selected by HORS/PORS. Consequently, signatures do not have a fixed size, and verifiers must take extra care to validate their inputs, e.g. to avoid buffer overflows. An adversary could indeed corrupt the size of a signature blob so that it differs from the logical size that corresponds to the selected indices.

Besides, a signer may be tempted to brute-force the input space (e.g. by cheating on the salt of HORS/PORS) to obtain shorter signatures. This must not be done because the security of HORS/PORS relies on uniform distribution of the selected indices.

3.4 Mask-less constructions

Recent versions of Merkle tree and Winternitz OTS constructions [DOTV08, Hül13] interleave hash evaluations with masking: the public key contains a list of uniformly generated masks, and each hash evaluation is preceded by XOR-ing a mask. The mask to select depends on the location of this hash evaluation in the overall structure.

Although masking allows to relax security requirements to second-preimage-resistant functions instead of collision-resistant functions, this reduction is less tight and security degrades with the total number of hash evaluations in the construction (e.g. 2^h for a Merkle tree of height h). Besides, against quantum computers, collision resistance and second-preimage resistance have the same generic security of $O(2^{n/2})$ for n bits of output [Ber09].

Hence, we propose to remove masks in these constructions (Figure 3.3). This gives a simpler design and slightly reduces the size of public keys – something to consider in the context of certificate chains. In Chapter 6, we review proofs of security for mask-less constructions relying on collision-resistant functions.

3.4.1 6-round Haraka

In [KLMR16], Kölbl et al. specified a 5-round version of Haraka v2 with second-preimage resistance in mind. As mask-less constructions require collision-resistant hash functions, we propose to use a 6-round version of Haraka. Indeed, no attack better than generic attacks is known against the collision-resistance of 6-round Haraka [KLMR16].

Although Haraka was fully specified and implemented in its 5-round version only, we straightforwardly extend the round constants for a 6-th round, using the same procedure. Namely, if p_i is the least significant bit of the i-th decimal digit of π, then the round j-th constant is defined as:

$$RC_j = p_{128j+128}|| \dots ||p_{128j+1}$$

(a) SPHINCS construction. (b) Mask-less construction.

Figure 3.3: Mask-less hashing in Merkle trees. In the maskful construction (left), the hash function H is assumed to be second-preimage-resistant. In the mask-less construction (right), H is assumed to be collision-resistant.

We explicit here the 8 new constants thereby obtained; for the previous rounds the reader is invited to refer to the specification of Haraka [KLMR16].

$$RC_{40} = \texttt{2ff372380de7d31e367e4778848f2ad2}$$
$$RC_{41} = \texttt{08d95c6acf74be8bee36b135b73bd58f}$$
$$RC_{42} = \texttt{5880f434c9d6ee9866ae1838a3743e4a}$$
$$RC_{43} = \texttt{593023f0aefabd99d0fdf4c79a9369bd}$$
$$RC_{44} = \texttt{329ae3d1eb606e6fa5cc637b6f1ecb2a}$$
$$RC_{45} = \texttt{e00207eb49e01594a4dc93d6cb7594ab}$$
$$RC_{46} = \texttt{1caa0c4ff751c880942366a665208ef8}$$
$$RC_{47} = \texttt{02f7f57fdb2dc1ddbd03239fe3e67e4a}$$

3.5 Batch signing

Signing is a somewhat costly process, that can induce a non-negligible overhead when many messages are signed. To amortize this computational cost, several *batching* methods have been developed to speed it up when several messages are available at the same time (for various recipients). Some methods leverage the algebraic structure of the signature scheme (e.g. RSA), but others are more generic. We now give a brief overview of batch signing methods and then show how they can bring a performance improvement to hash-based signatures, in terms of speed as well as signature size.

3.5.1 Examples of batch signatures

In 1989, Fiat presented a batching method for RSA signatures [Fia89]. For a fixed modulus, each message in a batch is assigned a distinct public exponent, such that these exponents are relatively prime (and relatively prime to the multiplicative group order). This choice allows to amortize the computational cost of private exponentiations throughout the batch. Compared to regular RSA, the public exponent is not included in the public key but in each signature. In short, this batching method heavily relies on the algebraic structure of RSA to trade signature size for signing speed.

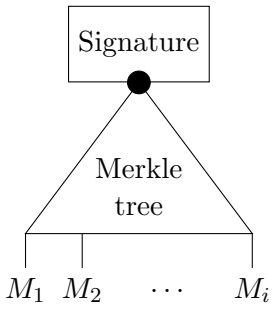


Figure 3.4: Pavlovski's batch signing [PB99]. A batch of messages $M_1, \ldots, M_i$ are hashed together in a Merkle tree. The root of this tree is authenticated with an expensive signature scheme.

In 1998, Bellare et al. presented a method for batch *verification* of modular exponentiation [BGR98], which can be useful to verify certificate chains with modular-exponentiation-based signatures (e.g. DSS), but also zero-knowledge proofs. Their batch verifier is probabilistic, i.e. sometimes accepts batches containing an illegitimate signature. This method does not claim strong unforgeability, but only existential unforgeability.

In 1999, Pavlovski et al. proposed a *generic* batch signing method [PB99]. The principle is to gather all the messages that are signed at the same time, compute a Merkle tree from their respective hashes and sign only the Merkle tree root with a traditional signature scheme. Then, the signature of each message contains the signature of the Merkle tree root and the authentication path for the corresponding message (Figure 3.4).

3.5.2 Application to hash-based signatures

In the context of hash-based signatures, batch signing has additional advantages. Indeed, we recall that a limiting factor to the development of hash-based signatures is the total number of messages that can be signed, as WOTS can only be used once and HORS/PORS can only be used a few times. A way to increase the targeted number of signatures is to use a large hyper-tree structure, as in XMSS and SPHINCS, but comes at the price of large signature sizes.

With batch signing, the total number of messages signed can be reduced, and consequently each signature can be made smaller. For example, one can define a signing period T (e.g. a millisecond for TLS connections, a day for software updates), group all messages within each period, and release a single signature at the end of the period. For a given use case, the frequency of signatures T^{-1} is predictable and allows to adapt the signature parameters to the life duration of a key pair. Further, such a scheme is still flexible and allows to occasionally shorten a period (e.g. if an emergency security update must be issued before the end of the day). We note that the precise signing period T is private to the signer, so there is no need to synchronize clocks with recipients.

Batch signing also allows to allocate more computing power to each signature, because this cost is amortized among many messages. Indeed, if N messages are signed in a period, computing N signatures each in time t (without batching) takes the same resources as computing 1 signature in time Nt (with batch signing). In the case of SPHINCS, this allows to

increase the height of each layer in the hyper tree, hence reducing the total number of WOTS signatures and the signature size.

Practical parameters

With a hyper-tree of height 60, SPHINCS authors targeted at most 2^{50} messages per key pair, arguing that it would take more than 30 years to exhaust a key at a rate of 2^{20} messages per second. Even for highly interactive environments, a period T of one millisecond reduces the target to 2^{40} signatures for more than 30 years per key pair, with 2^{10} messages per batch. The latency overhead of one millisecond seems acceptable, given that signing time is an order of magnitude larger on a single CPU[2].

With that in mind, the hyper-tree height of SPHINCS can be reduced by 10, hence removing 2 layers of WOTS signatures, saving 144 hashes, or 4608 bytes. On the other hand, the batching Merkle tree adds 10 authentication nodes per signature, i.e. 320 bytes. A batch index must also be sent, for example on 2 bytes. Overall, batch signing saves 4286 bytes. The height of internal Merkle trees can also be increased to save additional WOTS signatures.

Real-time deployment

Batch signing also offers advantages for highly interactive environments (e.g. TLS servers). First, the marginal cost of signing an additional message in a given period is in the order of a few hash evaluations, much faster than computing a full signature. Consequently, batch signing can easily adapt to load variations by gathering more messages in a single Merkle tree, which reduces the risk of denial-of-service attacks that flood the signer with messages.

Second, the overall memory footprint for a signer transmitting signatures to N recipients is reduced to a single signature and a Merkle tree with N leaves, instead of N signatures. Here again, the marginal memory cost of signing an additional message is in the order of a few hashes, thwarting denial-of-service attacks.

Multi-authentication

Additionally, Pavlovski's batch signing allows to authenticate a subset of the messages with a single signature. Consider the case of a software repository with daily updates, where each user wants a subset of the packages. After downloading the packages, they only need to fetch one signature for the day and a Merkle authentication path for each package in their chosen subset. They can even use an authentication octopus, as described in Section 5. This amortizes the signature size compared to fetching a full signature for each package. The marginal cost for authenticating an additional package is $O(\log N)$ where N is the total number of packages in the repository.

3.6 Summary

We now summarize to which extent our proposals reduce signature size compared to SPHINCS.

[2] In the case of SPHINCS, signing takes around 50 million cycles [BHH+15]

	SPHINCS	Gravity	Gravity + batching
signature	41000	28043 (average) 28928 (worst)	19469 (average) 20354 (worst)
public key	1056	32	32
secret key	1088	64	64
secret cache	0	2^{21}	2^{22}

Table 3.1: Comparison of SPHINCS and Gravity signatures and keys sizes (bytes), for similar computation times. These parameters are suitable for up to 2^{50} signatures per key pair.

- Secret key caching allows to remove 3 WOTS signatures (with 15 levels of cache), due to a large Merkle tree at the top of the hyper-tree. This saves 201 hashes per signature, i.e. 6432 bytes, at the expense of a 2 MiB cache.
- Deterministic leaf selection with our PORS construction increases the security level, so 2 layers of WOTS and their associated Merkle trees can be removed. This saves 144 hashes and a 64-bit index per signature, i.e. 4616 bytes.
- Batch signing with batches of 2^{10} messages allows to remove 2 layers of WOTS and their Merkle trees, and replace them by a batch Merkle tree below the signature. This saves 134 hashes per signature but a 2-byte index must be included, so 4286 bytes are saved. Additionally, the remaining 5 layers of Merkle trees each of height 5 can be replaced by 3 layers of height 8 (and increasing the secret cache by one level), thanks to amortization of the computation cost over many messages. This saves again 134 hashes per signature, i.e. 4288 bytes.
- Octopus multi-authentication in the PORS tree saves 1909 bytes per signature on average, and 1024 bytes in the worst case.
- Mask-less constructions remove 32 masks, i.e. 1024 bytes in the public key and in the secret key.

In total, 21531 bytes are saved on average in each signature, and 1024 bytes are removed in the public key. These optimizations reduce signature size by more than half! Even in a pessimistic scenario where batch signing is not applicable and the worst case is considered in octopus authentication, at least 12072 bytes are saved in each signature. Table 3.1 summarizes the size of signatures and keys for SPHINCS and Gravity. We give a more formal specification draft for Gravity in Chapter 7.

Chapter 4

Clarifying the subset-resilience problem

In 2002, Reyzin and Reyzin [RR02] presented the HORS signature scheme, a few-time signature scheme based on the *hash to obtain a random subset* construction. This construction uses a hash function to select a subset of a finite set; the signature is then derived from this subset. The security of HORS relies on the *subset-resilience* of the hash function, a property defined by Reyzin and Reyzin for the purpose of this scheme. Several signature schemes are variants of this idea, e.g. HORS++ [PWX03] and HORST [BHH+15]. The latter is an essential building block of SPHINCS, a practical stateless hash-based signature scheme [BHH+15].

However, we note that the *subset-resilience* problem was only partially studied. In particular, although Reyzin and Reyzin proposed two security notions, to capture the security against both adaptive and non-adaptive attacks, they only studied non-adaptive attacks. Yet, nothing prevents adaptive attacks in the textbook version of HORS. In this chapter, we investigate adaptive attacks and show that the security level decreases dramatically when HORS is used more than once. In particular, we propose a greedy algorithm that breaks textbook HORS and we give examples of forgeries.

We also investigate the SPHINCS construction. Contrary to textbook HORS, SPHINCS is not vulnerable to simple adaptive attacks, because the hash is unpredictable to attackers. Yet, we show that both HORS and SPHINCS are susceptible to what we call *weak messages* attacks, due to the way that HORS maps messages to subsets. Indeed, HORS maps some messages to small subsets – due to index collisions in the output of the hash function – and these small subsets are easier to cover. This yields an improved classical attack against SPHINCS, of complexity 2^{270} instead of 2^{277}. Yet, this new attack does not improve over known quantum attacks, based on Grover's search algorithm.

To mitigate these attacks, we propose the PORS construction – *PRNG to obtain a random subset* – which has no *weak messages*. We show that this new construction increases the security level for HORS and SPHINCS, against both classical and quantum attacks. Further, in the case of SPHINCS, we extend PORS to also select the hyper-tree leaf, instead of having this leaf freely chosen by a potential forger. We show that with this last improvement, the security margin of SPHINCS increases to 171 bits (post-quantum). Consequently, we propose smaller parameters for SPHINCS-PORS, to save 4616 bytes per signature.

4.1 The subset-resilience problem

In this section, we recall known results about the subset-resilience problem.

4.1.1 General definitions

Given integer parameters k and τ such that $0 < k \leq 2^\tau$, we let $t = 2^\tau$ and denote by T the set $\{0, \ldots, t-1\}$. We denote by $\mathcal{P}_k(T)$ the set of subsets of T of size at most k. We consider a finite key space $\mathcal{K}$, a message space $\mathcal{M}$ and a family of functions to *obtain random subsets* (ORS) from messages $\mathcal{O}_n = \{\mathsf{ORS}_K : \mathcal{M} \to \mathcal{P}_k(T) | K \in \mathcal{K}\}$. Given a key K and $r+1$ messages $M_1, \ldots, M_{r+1}$ we define the *r-subsets-cover* relation C_K as:

$$C_K(M_1, \ldots, M_{r+1}) \Leftrightarrow \mathsf{ORS}_K(M_{r+1}) \subseteq \bigcup_{j=1}^{r} \mathsf{ORS}_K(M_j)$$

which means that the image of M_{r+1} by ORS_K is covered by the images of $M_1, \ldots, M_r$.

Hash to obtain a random subset

Unless specified otherwise, we restrict our analysis to function families $\mathcal{O}_n$ that follow the *hash to obtain a random subset* (HORS) construction, that we now describe. Given a PRF $\mathcal{H}_n = \{H_K : \mathcal{M} \to \{0,1\}^{k\tau} | K \in \mathcal{K}\}$ that maps messages to $k\tau$-bit strings, the associated family $\mathcal{O}_n$ is constructed as follows. To compute $\mathsf{ORS}_K(M)$, take the string $x := H_K(M)$, split it into τ-bit blocks $x_1 || \ldots || x_k := x$ and interpret each block x_i as the encoding of a number $\overline{x_i} \in T$. The result $\mathsf{ORS}_K(M)$ is the subset $\mathsf{split}(x, k) := \{\overline{x_1}, \ldots, \overline{x_k}\}$.

We note that given a uniformly distributed key K and an arbitrary message M, the $\overline{x_i}$ are independent and uniformly distributed in T, because $\mathcal{H}_n$ is a PRF.

Once a subset is obtained, it can be turned into a signature with the scheme described in Section 2.3.2.

Practical parameters

To give some intuition about the scheme, we recall practical choices of (k, t). For the original HORS scheme, Reyzin and Reyzin proposed $k = 20, t = 256$ or $k = 16, t = 1024$. For HORST as used in SPHINCS, Bernstein et al. proposed more conservative parameters $k = 32, t = 2^{16}$.

4.1.2 Subset-resilience

Informally, $\mathcal{O}_n$ is r-subset-resilient if it is hard for an adversary to find $r+1$ messages that form a r-subsets-cover, under a key uniformly chosen in $\mathcal{K}$. Two adversarial scenarios were considered by Reyzin and Reyzin [RR02]. In the adaptive scenario, an adversary is given a key K and can compute ORS_K on any messages before selecting the $r+1$ messages. In the non-adaptive scenario, also called *target-subset-resilience* (as it is a generalization of *target-collision-resistance* [BR97]), the adversary must first select r messages, after which they are given the key K to select the last message.

Formally, the definitions by Reyzin and Reyzin can be given in terms of adversarial advantage. We define the advantage of an adversary $\mathcal{A}$ against the *r-subset-resilience* property

of $\mathcal{O}_n$ as:

$$\textsc{Succ}^{r-\mathrm{SR}}_{\mathcal{O}_n}(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K}; (M_1, \ldots, M_{r+1}) \leftarrow \mathcal{A}(K) : C_K(M_1, \ldots, M_{r+1})\right]$$

where the probability is over the choice of K (uniform in the key space $\mathcal{K}$) and the internal coins of $\mathcal{A}$.

Similarly, we define the advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the *r-target-subset-resilience* of $\mathcal{O}_n$ as:

$$\textsc{Succ}^{r-\mathrm{TSR}}_{\mathcal{O}_n}(\mathcal{A}) = \Pr\Big[M_1, \ldots, M_r \xleftarrow{\$} \mathcal{A}_1; K \xleftarrow{\$} \mathcal{K}; M_{r+1} \leftarrow \mathcal{A}_2(K, M_1, \ldots M_r) : C_K(M_1, \ldots, M_{r+1})\Big]$$

Reyzin and Reyzin originally defined (target-)subset-resilience as negligibility of the advantage of probabilistic polynomial-time adversaries w.r.t. the variables (t, k), more precisely as $\textsc{Succ}^{r-(\mathrm{T})\mathrm{SR}}_{\mathcal{O}_n}(\mathcal{A}) < negl(t, k)$. However, we argue that such asymptotic definitions have little practical interest because for concrete schemes what matters is the security level for some fixed (t, k). Also, they did not explicitly define $negl(t, k)$, and for example an adversary could easily generate a subsets-cover if t and k go to infinity with $t = k$. Last, in practice r is not fixed and we are also interested in asymptotics in r, as in the case of SPHINCS. Instead, we consider concrete notions of (in)security, with the framework defined in Section 1.1.1.

In our analysis, we consider only generic attacks against $\mathcal{O}_n$. In practice, if a particular family $\mathcal{O}_n$ has structural weaknesses that allow more efficient attacks, this family is considered broken and one can replace it with another family $\mathcal{O}'_n$.

Note The HORST scheme [BHH+15] – for "HORS with trees" – simply adds a Merkle tree on top of HORS to compress the public key. This is irrelevant to attacks against subset-resilience, so unless otherwise specified the results in this chapter hold for both HORS and HORST.

Previous results

Reyzin and Reyzin have already studied the non-adaptive scenario (target-subset-resilience). They considered an adversary performing a brute-force attack, i.e. given $M_1, \ldots, M_r$ one iterates M_{r+1} over the message space until a match is found. In that case, given $r + 1$ arbitrary messages and a uniformly distributed key K, the probability that $\mathsf{ORS}_K(M_{r+1})$ is covered by the first r messages is at most $(kr/t)^k$. This is the probability that k elements chosen uniformly at random in T are a subset of $\cup_{j=1}^{r} \mathsf{ORS}_K(M_j)$, which contains at most kr elements of T. In other words, the security level against this generic attack is:

$$k(\log_2 t - \log_2 k - \log_2 r)$$

In this chapter, we study the adaptive scenario and show that the security level decreases much faster when r increases.

4.2 Adaptive attacks against subset-resilience

In this section, we give a new lower bound on the subset-resilience security level, assuming only generic attacks against $\mathcal{O}_n$. We show that compared to target-subset-resilience, the (logarithmic) security level decreases by a factor proportional to $r+1$.

Theorem 1. *Assuming only generic attacks against $\mathcal{O}_n$, we have the following bound on the r-subset-resilience against adversaries performing at most q queries:*

$$\textsc{InSec}^{r-\mathrm{SR}}(\mathcal{O}_n; q) \leq \frac{(q+r+1)^{r+1}}{r!}\left(\frac{kr}{t}\right)^k$$

Proof. Let $\mathcal{A}$ be an adversary against the subset-resilience property of $\mathcal{O}_n$. Given a key K, we assume that $\mathcal{A}$ makes q distinct (offline) queries to ORS_K and outputs $r+1$ messages. We construct an adversary $\mathcal{A}'$ that runs $\mathcal{A}$ and additionally queries the $r+1$ messages output by $\mathcal{A}$, before outputting them. Adversary $\mathcal{A}'$ has the same success probability as $\mathcal{A}$ and makes at most $q' = q + r + 1$ queries. We denote as $(M_i, \mathsf{ORS}_K(M_i))_{1\leq i\leq q'}$ the q' queries of $\mathcal{A}'$.

Under these assumptions, the advantage of $\mathcal{A}'$ is bounded by the probability that there exists a permutation σ of $\{1,\ldots,q'\}$ yielding the following r-subsets-cover:

$$C_K(M_{\sigma(1)},\ldots,M_{\sigma(r+1)})$$

Given a permutation σ, the probability that this condition holds is at most $(kr/t)^k$, as in the non-adaptive case. Besides, there are $q'\binom{q'-1}{r}$ distinct configurations: each configuration is given by a choice of $\sigma(r+1)$ among q' indices, followed by the choice of a subset $\{\sigma(1),\ldots,\sigma(r)\}$ of r indices among the remaining $q'-1$. By a union bound over all configurations, we obtain the following bound on the adversarial advantage.

$$\textsc{Succ}^{r-\mathrm{SR}}_{\mathcal{O}_n}(\mathcal{A}) = \textsc{Succ}^{r-\mathrm{SR}}_{\mathcal{O}_n}(\mathcal{A}') \leq q'\binom{q'-1}{r}\left(\frac{kr}{t}\right)^k \leq \frac{q'^{r+1}}{r!}\left(\frac{kr}{t}\right)^k$$

The result follows. □

We note that for practical parameters ($q \gg r^2$), we have the following approximation:

$$\frac{(q+r+1)^{r+1}}{r!}\left(\frac{kr}{t}\right)^k \approx \frac{q^{r+1}}{r!}\left(\frac{kr}{t}\right)^k$$

The associated security level corresponds to the number of queries necessary for the advantage to be close to 1, that is:

$$\begin{aligned}&\frac{k}{r+1}(\log_2 t - \log_2 k - \log_2 r) + \frac{\log_2 r!}{r+1}\\ \approx&\frac{k}{r+1}(\log_2 t - \log_2 k - \log_2 r) + \log_2 r\end{aligned}$$

As we can see, contrary to non-adaptive attacks, security degrades quickly with r due to the denominator, in a birthday-like manner. The case $r = 1$ is similar to collision resistance.

Universal forgeries

Although this goes beyond our definition of subset-resilience, we give a brief analysis of universal forgery. If the adversary wishes to make a universal forgery, the message $M_{\sigma(r+1)}$ is fixed. There are now $\binom{q'-1}{r}$ configurations for σ and the success probability is bounded by:

$$\frac{q'^r}{r!}\left(\frac{kr}{t}\right)^k$$

yielding a security level of:

$$\approx \frac{k}{r}(\log_2 t - \log_2 k - \log_2 r) + \log_2 r$$

The case $r = 1$ is similar to second-preimage resistance.

4.3 Practical attacks against HORS

In the previous section, we gave conservative bounds by considering only whether a subsets-cover exist, but we did not provide practical algorithms to find them. In particular, even though a r-subsets-cover exists, a naive algorithm that iterates over all configurations σ would take a time proportional to $q\binom{q-1}{r}$, introducing no improvement over a non-adaptive attack. Hence, we look for faster algorithms to find subsets-covers.

4.3.1 Reduction to *set cover*

Given q messages and their hashes, finding a subsets-cover reduces to the *set cover* problem. Indeed, for a given $M_{\sigma(r+1)}$, we try to cover the set $X = \mathsf{ORS}_K(M_{\sigma(r+1)})$ with a cover C of r elements from the family $\mathcal{Y} = \{Y_i\}_{i \neq \sigma(r+1)}$, where $Y_i = \mathsf{ORS}_K(M_i) \cap X$. Set cover is an NP-complete decision problem; the associated optimization problem (finding the smallest r) is NP-hard [Joh74, Fei98]. However, approximations can be computed with polynomial complexity.

Greedy algorithm

Given a set X to be covered by a family $\mathcal{Y}$, a simple heuristic is the following greedy algorithm.

1. Initialize an empty cover $C \leftarrow \emptyset$.
2. Select $Y_{\max} \in \mathcal{Y}$ such that $|Y_{\max}|$ is maximal.
3. Add $Y_{\max}$ to the cover C.
4. Update $X \leftarrow X \setminus Y_{\max}$ and $\mathcal{Y} \leftarrow \{Y_i \setminus Y_{\max} | Y_i \in \mathcal{Y}\}$, where $A \setminus B$ denotes the set difference.
5. Repeat steps 2-4 until C contains r subsets or X is empty.
6. If $X = \emptyset$, the algorithm succeeds and outputs C.

The worst-case complexity of this greedy algorithm is $O(qr)$, much better than the naive $O(q^r/r!)$. The question is now: depending on the parameters k, t, r, q, what is the success probability of this algorithm?

Previous work

In [Joh74], Johnson has shown that for the general *set cover* problem, this greedy algorithm achieves an approximation ratio of $\log k$ in the worst case. This means that if there exists an optimal cover C_{opt} of r sets in $\mathcal{Y}$, the greedy algorithm will find a cover of at most $r \log k$ sets. Later, Feige [Fei98] showed that this greedy algorithm is essentially the best one can hope for in the general case with probabilistic polynomial-time algorithms.

However, these results are for the worst-case scenario, where the family $\mathcal{Y}$ can be chosen in an adversarial manner w.r.t. the algorithm. In our case, $\mathcal{Y}$ is chosen according to the HORS construction, which yields a very specific probability distribution and we are interested in the *average* approximation ratio, which may well be better than $\log k$ for the greedy algorithm – and/or for other efficient algorithms.

Targeting weak messages

Instead of running the greedy algorithm on an arbitrary message, the adversary can select a better target. Indeed, due to the HORS construction, some messages have an image by ORS_K that contain only $\kappa < k$ elements. We call them *weak messages*. For example, following is the SHA-256 of "88681", grouped by blocks of 8 bits, with some repeated bytes underlined.

```
98 32 3d bf 2a 64 75 32 0f f6 64 7e 98 75 64 98 f6 f5 54 02 ...
```

Hence, we propose the following optimized algorithm: first scan through the q messages to find M such that $\mathsf{ORS}_K(M)$ has the least number of elements, then try to cover $\mathsf{ORS}_K(M)$ greedily. This new algorithm still has a complexity of $O(qr)$ and its success probability can be slightly better than the naive greedy algorithm.

To summarize, we have three algorithms:

- the **greedy algorithm**, that takes as input a set X to be covered by a family $\mathcal{Y}$,
- the **naive greedy algorithm**, that takes as input messages $M_1, \ldots, M_q$ and applies the greedy algorithm with an arbitrary $X = \mathsf{ORS}_K(M_1)$ and $\mathcal{Y} = \{\mathsf{ORS}_K(M_j) | j \neq 1\}$,
- the **optimized greedy algorithm**, that takes as input q messages and applies the greedy algorithm with the smallest $\mathsf{ORS}_K(M_i)$ as X, and $\mathcal{Y} = \{\mathsf{ORS}_K(M_j) | j \neq i\}$.

Probability distribution of weak messages

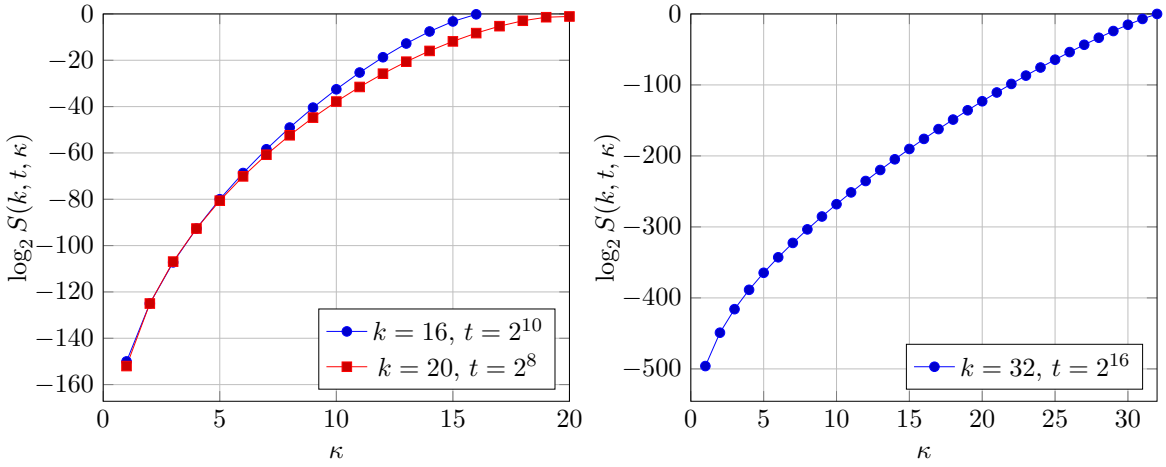
We now give a useful combinatorics result about the distribution of weak messages.

Lemma 1. *Let $S(k, t, \kappa)$ be the probability that when we throw k balls independently and uniformly into t bins, exactly κ bins are non-empty. Then we have the following equality:*

$$S(k, t, \kappa) = \frac{\kappa!}{t^k} \binom{t}{\kappa} \left\{ {k \atop \kappa} \right\}$$

where $\left\{ {k \atop \kappa} \right\}$ is the notation for Stirling numbers of the second kind.

The probability $\mathsf{Pr}[|\mathsf{ORS}_K(M)| = \kappa]$ that a message is mapped to exactly κ elements is precisely $S(k, t, \kappa)$. Figure 4.1 shows the evolution of $\log_2 S(k, t, \kappa)$ for the choices of the parameters (k, t) that were proposed for HORS [RR02] and SPHINCS [BHH+15].



(a) Textbook HORS.

(b) HORST as used in SPHINCS.

Figure 4.1: Evolution of $S(k,t,\kappa)$ for various choices of (k,t). $S(k,t,\kappa)$ is the probability that when we throw k balls independently and uniformly into t bins, exactly κ bins are non-empty (Lemma 1).

Proof. There are t^k equiprobable combinations of k balls into t bins. Out of those, we count the number of combinations for which exactly κ bins are non-empty:

- there are $\binom{t}{\kappa}$ ways to choose κ bins out of t;
- there are $\left\{ {k \atop \kappa} \right\}$ partitions of k labelled balls into κ classes (this is precisely the definition of Stirling numbers of the second kind);
- there are $\kappa!$ ways to associate the κ classes of balls to the κ bins.

This yields the result. □

4.3.2 Complexity analysis

We now give a complexity analysis of the greedy and optimized greedy algorithms, to estimate the security level that one can obtain against this generic attack. Given ORS parameters (k,t), a family $\mathcal{Y}$ of q elements, a number of iterations r and an initial subset X of size κ, we denote by $P^{k,t}_{\mathsf{Greedy}}(\kappa,r,q)$ the success probability of the greedy algorithm. Given q queries, we denote by $P^{k,t}_{\mathsf{GreedyNaive}}(r,q)$ the success probability of the naive greedy algorithm that takes an arbitrary message as the initial subset X. We denote by $P^{k,t}_{\mathsf{GreedyOptim}}(r,q)$ the success probability of the optimized greedy algorithm that selects the weakest of the q messages as the initial subset X.

Naive greedy algorithm

We first establish a recurrence relation on $P^{k,t}_{\mathsf{Greedy}}(\kappa,r,q)$.

Theorem 2. *Given a target subset of κ elements, the success probability of the greedy algorithm $P^{k,t}_{\mathsf{Greedy}}(\kappa, r, q)$ verifies the following recurrence relation.*

$$P^{k,t}_{\mathsf{Greedy}}(\kappa, r, q) = \begin{cases} 1 & \text{if } \kappa = 0 \\ 0 & \text{if } \kappa > 0 \wedge (r = 0 \vee q = 0) \\ \sum_{\ell=1}^{\kappa} \mathsf{Pr}[|Y_{\max}| = \ell | k, t, \kappa, q] P^{k,t}_{\mathsf{Greedy}}(\kappa - \ell, r - 1, q - 1) & \text{if } \kappa, r, q > 0 \end{cases}$$

Proof. First, the greedy algorithm always succeeds if there is nothing to cover ($\kappa = 0$), and always fails if there is something to cover ($\kappa > 0$) but there is no iteration or message left ($r = 0$ or $q = 0$).

Second, if the first iteration finds a maximal subset of size ℓ, the success probability of the following iterations is equal to the success probability of the greedy algorithm on a subset of size $\kappa - \ell$ with one less iteration and one less available message. □

We now aim at evaluating the probability $\mathsf{Pr}[|Y_{\max}| = \ell | k, t, \kappa, q]$ that the largest intersection $|\mathsf{ORS}_K(M) \cap X|$ contains ℓ elements. We start with the following lemma.

Lemma 2. *For $0 \leq \ell \leq \kappa$ and M a message uniformly distributed in $\mathcal{M}$, we denote by $P(k, t, \kappa, \ell)$ the probability that $|\mathsf{ORS}_K(M) \cap X| = \ell$ given that $|X| = \kappa$. This probability is equal to:*

$$P(k, t, \kappa, \ell) = \sum_{\lambda=\ell}^{k} B\left(\lambda, k, \frac{\kappa}{t}\right) S(\lambda, \kappa, \ell)$$

where S is defined in Lemma 1 and $B(\lambda, k, p)$ is the binomial distribution:

$$B(\lambda, k, p) = \binom{k}{\lambda} p^{\lambda} (1 - p)^{k - \lambda}$$

Proof. We recall that $\mathsf{ORS}_K(\cdot)$ simulates throwing k balls uniformly and independently into t bins. With that in mind, $B(\lambda, k, \kappa/t)$ is the probability that λ balls out of k fall into the κ elements of X. Then, $S(\lambda, \kappa, \ell)$ is the probability that these λ balls cover exactly ℓ bins out of κ.

The sum starts at $\lambda = \ell$ because $S(\lambda, \kappa, \ell) = 0$ when $\lambda < \ell$ (it is impossible to cover ℓ bins with less than ℓ balls). □

We can now relate $\mathsf{Pr}[|Y_{\max}| = \ell | k, t, \kappa, q]$ to $P(k, t, \kappa, \ell)$.

Lemma 3. *We have the following equality:*

$$\mathsf{Pr}[|Y_{\max}| \geq \ell | k, t, \kappa, q] = 1 - \left(\sum_{\lambda=0}^{\ell-1} P(k, t, \kappa, \lambda) \right)^q$$

Proof. By independence of the Y_j:

$$\begin{aligned} \mathsf{Pr}[|Y_{\max}| \geq \ell | k, t, \kappa, q] &= \mathsf{Pr}[\exists j \in \{1, \ldots, q\} \ |Y_j| \geq \ell | k, t, \kappa] \\ &= 1 - \mathsf{Pr}[\forall j \in \{1, \ldots, q\} \ |Y_j| < \ell | k, t, \kappa] \\ &= 1 - \prod_{j=1}^{q} \mathsf{Pr}[|Y_j| < \ell | k, t, \kappa] \\ &= 1 - \mathsf{Pr}[|Y_1| < \ell | k, t, \kappa]^q \end{aligned}$$

where

$$\mathsf{Pr}[|Y_1| < \ell | k, t, \kappa] = \sum_{\lambda=0}^{\ell-1} P(k, t, \kappa, \lambda)$$

The result follows. □

Corollary 1. *We can then compute* $\mathsf{Pr}[|Y_{\max}| = \ell | k, t, \kappa, q]$ *as:*

$$\mathsf{Pr}[|Y_{\max}| = \ell | k, t, \kappa, q] = \mathsf{Pr}[|Y_{\max}| \geq \ell | k, t, \kappa, q] - \mathsf{Pr}[|Y_{\max}| \geq \ell + 1 | k, t, \kappa, q]$$

Although finding a more explicit analytic formula seems challenging, Theorem 2 and Lemmas 2 and 3 allow to compute $P^{k,t}_{\mathsf{Greedy}}(\kappa, r, q)$ by dynamic programming for concrete values of the parameters.

We can now estimate the success probability of the naive algorithm.

Theorem 3. *The success probability of the naive greedy algorithm is equal to:*

$$P^{k,t}_{\mathsf{GreedyNaive}}(r, q) = \sum_{\kappa=1}^{k} S(k, t, \kappa) P^{k,t}_{\mathsf{Greedy}}(\kappa, r, q-1)$$

Proof. The naive algorithm selects an arbitrary message, that has a probability $S(k, t, \kappa)$ of containing κ distinct elements. Given that, there is a probability $P^{k,t}_{\mathsf{Greedy}}(\kappa, r, q-1)$ that the algorithm succeeds. □

Optimized greedy algorithm

We now relate the success probability of the optimized greedy algorithm to the success probability of the greedy algorithm.

Theorem 4. *The success probability of the optimized greedy algorithm is equal to:*

$$P^{k,t}_{\mathsf{GreedyOptim}}(r, q) = \sum_{\kappa=1}^{k} \mathsf{Pr}[|Y_{\min}| = \kappa | k, t, q] P^{k,t}_{\mathsf{Greedy}}(\kappa, r, q-1)$$

where $|Y_{\min}|$ *is the size of the weakest message:*

$$|Y_{\min}| = \min_{Y \in \mathcal{Y}} |Y|$$

Proof. For $1 \leq \kappa \leq k$, there is a probability $\mathsf{Pr}[|Y_{\min}| = \kappa | k, t, q]$ that the weakest message found by the algorithm contains κ distinct elements. Given that, the algorithm succeeds with probability $P^{k,t}_{\mathsf{Greedy}}(\kappa, r, q-1)$. □

We now relate $\mathsf{Pr}[|Y_{\min}| = \kappa | k, t, q]$ to $S(k, t, \kappa)$.

Lemma 4. *We have the following equality:*

$$\mathsf{Pr}[|Y_{\min}| \leq \kappa | k, t, q] = 1 - \left(\sum_{\lambda=\kappa+1}^{k} S(k, t, \lambda) \right)^q$$

Proof. The proof is similar to Lemma 3. By independence of the Y_j:

$$\begin{aligned}\mathsf{Pr}[|Y_{\min}| \leq \kappa | k,t,q] &= \mathsf{Pr}[\exists j \in \{1,\ldots,q\} \ |Y_j| \leq \kappa | k,t] \\ &= 1 - \mathsf{Pr}[\forall j \in \{1,\ldots,q\} \ |Y_j| > \kappa | k,t] \\ &= 1 - \mathsf{Pr}[|Y_1| > \kappa | k,t]^q\end{aligned}$$

Besides,

$$\mathsf{Pr}[|Y_1| > \kappa | k,t] = \sum_{\lambda=\kappa+1}^{k} S(k,t,\lambda)$$

which yields the result. □

Corollary 2. *We can then compute* $\mathsf{Pr}[|Y_{\min}| = \kappa | k,t,q]$ *as:*

$$\mathsf{Pr}[|Y_{\min}| = \kappa | k,t,q] = \mathsf{Pr}[|Y_{\min}| \leq \kappa | k,t,q] - \mathsf{Pr}[|Y_{\min}| \leq \kappa - 1 | k,t,q]$$

Theorem 4 and Lemma 4 allow to compute $P^{k,t}_{\mathsf{GreedyOptim}}(r,q)$.

Practical security level

In the previous section, we studied how to compute the success probability of the naive and optimized greedy algorithms. We now reduce the complexity of these attacks to these success probabilities.

Given q queries and a target r, the naive greedy algorithm has a complexity proportional to $1+r(q-1)$ (one query for the covered message and r iterations over the $q-1$ other queries) and a success probability of $P^{k,t}_{\mathsf{GreedyNaive}}(k,r,q)$, so we expect to repeat it $1/P^{k,t}_{\mathsf{GreedyNaive}}(k,r,q)$ times on average to obtain a r-subsets-cover. The optimal number of queries q is the one that minimizes the attack complexity:

$$\frac{1+r(q-1)}{P^{k,t}_{\mathsf{GreedyNaive}}(k,r,q)}$$

Likewise, given q and r, the optimized greedy algorithm has a complexity proportional to $q + r(q-1)$ (q queries to find the weakest message and r iterations over the remaining $q-1$ queries to cover it) and a success probability of $P^{k,t}_{\mathsf{GreedyOptim}}(r,q)$. The optimal number of queries q is the one that minimizes the attack complexity:

$$\frac{q+r(q-1)}{P^{k,t}_{\mathsf{GreedyOptim}}(r,q)}$$

We estimated these attack complexities by taking the minimum over $q \in \{2^n | n \in \mathbb{N}\}$. Figure 4.2 shows a comparison of the lower bounds on adaptive and non-adaptive attacks, and the complexity of the naive and optimized greedy algorithms, for some of the parameters originally proposed for HORS ($k = 16, t = 1024$). We can see that the optimized greedy algorithm is very close to the lower bound on adaptive attacks, despite its simplicity. Also, the difference between the naive and optimized algorithms becomes small as r grows.

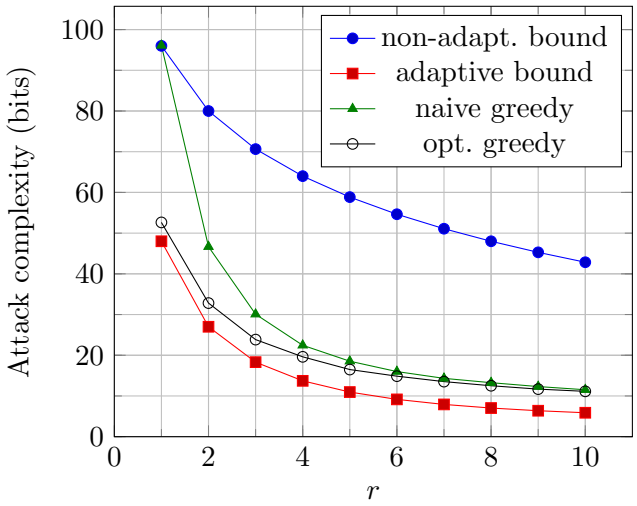


Figure 4.2: Security level for $k = 16, t = 1024$. Lower bounds for adaptive and non-adaptive attacks and complexity of naive and optimized greedy algorithms.

4.3.3 Forgeries on textbook HORS

In [RR02], Reyzin and Reyzin proposed several combinations of (k, t) for the HORS signature scheme: $k = 20, t = 256$ and $k = 16, t = 1024$. Considering only non-adaptive attacks, they claimed a security level of 53 bits in the first case with $r = 2$ signatures and 64 bits in the second case with $r = 4$ signatures. We challenge these claims in the case of adaptive attacks – which are totally possible in this textbook version of HORS – and give forgeries for these sets of parameters. We computed these forgeries with the optimized greedy algorithm.

Reyzin and Reyzin originally proposed to instantiate $\mathcal{O}_n$ with a hash function such as SHA-1, but given that this function is not collision-resistant [SBK+17] we used SHA-256 instead for our proof-of-concept attack, trimming the output to 160 bits to obtain compatible parameters. In other words, we used $\mathsf{ORS}_K(\cdot) = \mathsf{split}(\mathsf{trim}(\text{SHA-256}(\cdot), 160), k)$ where $\mathsf{trim}(\cdot, 160)$ returns the first 160 bits of its input, and split groups the output in blocks of k bits. We limited ourselves to $q = 2^{22}$ queries, with each query M_i being the decimal representation of i as an ASCII string.

First case $k = 20, t = 256$ With only $q = 2^{21}$ queries, we found a 2-subsets-cover with the optimized greedy algorithm (Table 4.1). This is much smaller than the advertised security level of 53 bits against non-adaptive attacks [RR02], and a little larger than our adaptive lower bound of 18.1 bits estimated in Section 4.2. We note that the target subset $\mathsf{ORS}_K(M_{88681})$ contains only $\kappa = 13$ distinct elements.

Second case $k = 16, t = 1024$ With only $q = 2^{22}$ queries, we found a 3-subsets-cover with the optimized greedy algorithm (Table 4.2). This is much smaller than the advertised security level of 70 bits against non-adaptive attacks [RR02], and a little larger than our adaptive lower bound of 18.3 bits estimated in Section 4.2. The target subset $\mathsf{ORS}_K(M_{88817})$ contains only $\kappa = 12$ distinct elements.

i	160 first bits of SHA-256(i) (by groups of 8 bits)
88681	98 32 3d bf 2a 64 75 32 0f f6 64 7e 98 75 64 98 f6 f5 54 02
1468639	54 7e 64 39 f6 61 1f 4d 02 32 3d 23 68 62 9d 3e 38 bc 75 5c
80937	4b 98 e7 f5 05 9b ee 8d f4 0f 89 bb 13 7e bf 08 89 fc be 2a

Table 4.1: Example of 2-subsets-cover for $k = 20, t = 256$. The first line is the covered message. Underlined blocks are repeated in the same line. Colored blocks are common to several lines.

i	160 first bits of SHA-256(i) (by groups of 10 bits)
88817	3f0 2b4 193 087 38c 1e0 2b4 193 1b6 116 1c1 087 046 33a 243 243
2530852	1a9 2c6 1a2 0fe 15f 279 33a 026 2b4 3e6 2a9 116 09f 087 111 3f0
2351182	00e 02a 39c 243 292 378 152 22c 201 0ab 06d 1b6 1ff 204 1e0 1c1
216522	277 38c 06d 39c 193 124 376 2a9 08a 046 351 2df 072 219 34e 1fe

Table 4.2: Example of 3-subsets-cover for $k = 16, t = 1024$.

We also found a 4-subsets-cover with 2^{17} queries, again much smaller than the advertised 64 bits of security (our adaptive lower bound is 13.7 bits in this case).

4.4 Application to SPHINCS

We now study attacks against subset-resilience in the context of the SPHINCS construction. At first glance, SPHINCS is not susceptible to adaptive attacks, because the signer selects for each message M_i a key K_i in a manner non-predictable by an adversary. More precisely, $K_i = \mathsf{hash}(\mathsf{salt}, M)$ where salt is a secret value known only by the signer. That way, an adversary cannot compute subsets-covers in advance; they have to first query the signatures of some messages and then try to find a key-message pair (K', M') such that $\mathsf{ORS}_{K'}(M')$ is covered by the previous signatures.

However, the SPHINCS construction uses $N = 2^h$ instances of HORST in parallel (typically 2^{60}). For each query, the signer also selects a HORST instance i in a deterministic but non-predictable manner. The problem is then different than target-subset-resilience because an adversary can attack multiple instances in parallel. In particular, although looking for weak messages does not seem to help to break a single HORST instance in a non-adaptive attack, this can provide a speed-up for the multi-target scenario.

The adversary can also focus on a HORST instance of their choice, a strategy outlined in the original analysis of SPHINCS [BHH+15].

Search strategies

More formally, we let $\overline{\mathcal{M}} = \mathcal{K} \times \mathcal{M}$ (the extended message space that also includes the ORS key). Given q known signatures, the goal of an adversary is to find a pair $(i, (K, M)) \in \{1, \ldots, N\} \times \overline{\mathcal{M}}$ such that $\mathsf{ORS}_K(M)$ is covered by the known subsets of the i-th HORST instance. A naive strategy is to brute-force over the full space $\{1, \ldots, N\} \times \overline{\mathcal{M}}$.

However, each dimension of the search space $\{1,\ldots,N\} \times \overline{\mathcal{M}}$ has weaknesses. On the one hand, we have already seen that $\overline{\mathcal{M}}$ contains weak messages. On the other hand, $\{1,\ldots,N\}$ has weak HORST instances: the ones that produced many signatures. A more clever strategy, outlined in the SPHINCS paper [BHH+15] is to focus on weak HORST instances and try to find matching messages for these instances – via classical brute-force over the message space or Grover's quantum search algorithm. A symmetric strategy is to look for weak messages – via brute-force or Grover's algorithm – and try to find matching HORST instances. We now evaluate the complexity of each strategy.

Naive search

We let $\rho = q/N$ be the ratio of the number of queries by the number of HORST instances. Given an arbitrary pair $(i, \overline{M}) \in \{1,\ldots,N\} \times \overline{\mathcal{M}}$, the probability $P_\rho(r)$ that r messages were signed with the i-th HORST instance can be approximated by a Poisson distribution, as shown in [BHH+15].

$$P_\rho(r) \approx e^{-\rho}\rho^r/r!$$

Then, the probability that $\mathsf{ORS}_K(M)$ is covered by this instance is at most $(kr/t)^k$ – a tight approximation if $kr \ll t$. Hence, the success probability for the pair $(i, \overline{M})$ is at most:

$$\sum_{r=0}^{\infty} P_\rho(r)\left(\frac{kr}{t}\right)^k = \left(\frac{k}{t}\right)^k \sum_{r=0}^{\infty} P_\rho(r) r^k$$

The last sum corresponds to the k-th moment of a Poisson distribution and is equal to a Touchard polynomial T_k [Rio37].

$$\sum_{r=0}^{\infty} P_\rho(r) r^k = T_k(\rho) := \sum_{i=0}^{k} \left\{ {k \atop i} \right\} \rho^i$$

Hence, the classical complexity to obtain a forgery with this strategy is (in bits):

$$k(\log_2 t - \log_2 k) - \log_2 T_k(\rho)$$

With SPHINCS parameters ($k = 32, t = 2^{16}, \rho = 2^{-10}$), we obtain a classical complexity of 331 bits for this naive attack.

Quantum naive search

Given a quantum computer, one could use Grover's search algorithm, dividing by 2 the bit complexity.

However, this is neglecting memory access costs! Indeed, given a function $F : \mathcal{X} \mapsto \{0,1\}$ such that $\mathsf{Pr}[x \xleftarrow{\$} \mathcal{X} : F(x) = 1] = 2^{-b}$, Grover's algorithm can find a preimage $x \in F^{-1}(1)$ with $\Theta(2^{b/2} f)$ quantum operations on $\Theta(f)$ qubits where f is the cost of evaluating F [Gro96]. For the naive algorithm we consider $\mathcal{X} = \{1,\ldots,N\} \times \overline{\mathcal{M}}$ and $b = 331$ as shown in the previous section.

In our case, F must first compute $\mathsf{ORS}_K(M)$ and then compare this result to the q queries, outputting 1 if and only if a subsets-cover is found. While evaluating $\mathsf{ORS}_K(M)$ is somewhat

cheap, the comparison circuit to a large database of q queries is expensive. In particular, one cannot use classical algorithms that perform conditional memory access such as binary search, because the quantum comparison circuit operates on a *quantum superposition* of values (and each value would need to access distinct parts of the memory).

In practice, a quantum straight-line comparison circuit would need $\Theta(q)$ operations on $\Theta(q)$ qubits [Ber09]. Besides, communication costs must be taken into account in a realistic model of memory access, e.g. random access in a 2-dimensional table of size q takes time $\Theta(\sqrt{q})$. These remarks are similar to the analysis of quantum collision search by Bernstein [Ber09].

With SPHINCS parameters ($k = 32, t = 2^{16}, h = 60$) and $q = 2^{50}$, we obtain a quantum complexity of 215 bits, assuming an evaluation cost $f = q$, but we will see that more clever methods reduce memory costs and achieve better results.

Search over weak HORST instances

To speed up the search, one can focus on the weakest HORST instance i – found in time $\Theta(q)$ – and then look for messages $\overline{M} \in \overline{\mathcal{M}}$. For a given r, the probability that there exists an instance i that signed r messages can be approximated as (assuming $\rho \ll 1$):

$$1 - (1 - P_\rho(r))^{2^h} \approx 1 - \exp(-2^h P_\rho(r)) \lesssim \min\{1,\ 2^h P_\rho(r)\}$$

Then, the success probability for each message $\overline{M}$ is again close to $(kr/t)^k$. It follows that the classical complexity of this attack is (in bits):

$$\min_r \left[k(\log_2 t - \log_2 k - \log_2 r) + \max\left\{0,\ -h - \log_2 P_\rho(r)\right\}\right]$$

One can also use Grover's search algorithm over $\overline{\mathcal{M}}$. Contrary to the naive algorithm, memory access is limited to the selected HORST instance, so we neglect memory costs. The associated quantum complexity is:

$$\min_r \left[\frac{k}{2}(\log_2 t - \log_2 k - \log_2 r) + \max\left\{0,\ -h - \log_2 P_\rho(r)\right\}\right]$$

With SPHINCS parameters ($k = 32, t = 2^{16}, h = 60$) and $q = 2^{50}$, we find a classical complexity of 277 bits and a quantum complexity of 138 bits, both for $r = 5$. This is in accordance with the SPHINCS paper [BHH+15].

Worldwide attack An interesting question is whether one can benefit from this strategy to break *any* SPHINCS instance in the world. Indeed, if there are u SPHINCS instances, each containing 2^h HORST instances, and that each SPHINCS instance is used for at most q queries, then the probability that at least one HORST instance in the world was used for r messages is:

$$1 - (1 - P_\rho(r))^{2^h u}$$

The previous analysis can be adapted by replacing 2^h by $2^h u$. For example, if $u = 2^{40}$, this worldwide attack has a classical complexity of 256 bits and a quantum complexity of 128 bits (both for $r = 8$).

Search over weak messages

With a classical computer, an adversary can find a weak message of size κ in time $\Theta(1/S(k,t,\kappa))$. With a quantum computer, this can be reduced to $\Theta(\sqrt{1/S(k,t,\kappa)})$ with Grover's search algorithm.

Once a weak message $\overline{M}$ of subset size κ is found, it is covered by each HORST instance with probability $\approx (kr/t)^\kappa$, where r is the number of signatures already generated by this instance. The success probability of this message against any HORST instance is then:

$$\sum_{r=0}^{\infty} P_\rho(r)\left(\frac{kr}{t}\right)^\kappa = \left(\frac{k}{t}\right)^\kappa T_\kappa(\rho)$$

If the adversary only scans through the $\approx q = 2^h\rho$ HORST instances that have already been used to sign messages, the success probability for each instance increases to:

$$\frac{1}{1-P_\rho(0)}\left(\frac{k}{t}\right)^\kappa T_\kappa(\rho) \approx \rho^{-1}\left(\frac{k}{t}\right)^\kappa T_\kappa(\rho)$$

Each weak message $\overline{M}$ is processed with complexity $\Theta(\max\{q, 1/S(k,t,\kappa)\})$ and is successful with approximate probability:

$$1-\exp(-2^h(k/t)^\kappa T_\kappa(\rho)) \lesssim \min\{1,\ 2^h(k/t)^\kappa T_\kappa(\rho)\}$$

It follows that the classical complexity of this attack is (in bits):

$$\min_\kappa \Big[\max\Big\{0,\ \kappa(\log_2 t - \log_2 k) - \log_2 T_\kappa(\rho) - h\Big\} + \max\Big\{h + \log_2\rho,\ -\log_2 S(k,t,\kappa)\Big\}\Big]$$

One can also use Grover's search algorithm over $\overline{\mathcal{M}}$, yielding a quantum complexity of:

$$\min_\kappa \Big[\max\Big\{0,\ \kappa(\log_2 t - \log_2 k) - \log_2 T_\kappa(\rho) - h\Big\} + \max\Big\{h + \log_2\rho,\ \frac{-\log_2 S(k,t,\kappa)}{2}\Big\}\Big]$$

With SPHINCS parameters ($k = 32, t = 2^{16}, h = 60$) and $q = 2^{50}$, we find a classical complexity of 270 bits (for $\kappa = 26$) and a quantum complexity of 187 bits (for $\kappa = 5$). Compared to search over weak HORST instances, this attack is more efficient classically, but less efficient quantumly.

4.5 Fixing HORS: PRNG to obtain a random subset

As we have seen, the existence of weak messages for $\mathcal{O}_n$ allows to perform more efficient attacks. We propose to improve the HORS construction to remove weak messages and mitigate these attacks. Instead of splitting the output of a hash function into k possibly non-unique indices, we propose to use a PRNG and collect the first k distinct values.

We then extend this construction to also select the hyper-tree leaf in SPHINCS, and show that this improvement increases the security level of SPHINCS by a significant margin.

4.5.1 PORS construction

Given a PRF $\mathcal{H}_n = \{H_K : \mathcal{M} \to \{0,1\}^n | K \in \mathcal{K}\}$ that maps messages to n-bit strings, and a PRNG $\mathcal{G}$ that expands a n-bit seed into an arbitrary long stream of bits, we consider the following *PRNG to obtain a random subset* construction. Given a key K and a message M, we let $\mathsf{seed} = H_K(M)$. We define the sequence $(x_i)_{i>0}$ computed by grouping the output of the PRNG by blocks of τ bits, i.e. $x_1||x_2||\ldots = \mathcal{G}(\mathsf{seed})$. We then define $\mathsf{ORS}_K(M)$ as the set containing the first k distinct values of $(x_i)_{i>0}$.

Soundness of the construction

Since $\mathcal{G}$ is a PRNG, the x_i are indistinguishable from independent random elements uniformly chosen in $\{0,1\}^\tau$. Collecting k distinct elements among t is a variant of the coupon's collector problem, and succeeds in an average of $t(H_t - H_{t-k}) \approx k$ steps, where $H_n \approx \log n$ is the n-th harmonic number.

Besides, an upper bound of the failure probability after T steps is:

$$\mathsf{Pr}[\mathsf{failure}|k,t,T] \leq \binom{T}{k^-}\left(\frac{k^-}{t}\right)^{T-k^-} \leq \frac{T^{k^-}}{k^-!}\left(\frac{k^-}{t}\right)^{T-k^-} \quad \text{where} \quad k^- = k-1$$

Indeed, coupon collection fails after T steps if there are at least $T - k^-$ steps for which the obtained coupon is not new. There are $\binom{T}{k^-}$ choices for the $T - k^-$ failing steps, and each failing step happens with probability at most k^-/t.

This failure probability decreases exponentially with T and in practice a small number of steps is enough. For example, with SPHINCS parameters ($k = 32, t = 2^{16}$), the failure probability is less than 2^{-128} after $T = 47$ steps, i.e. the values $x_1, \ldots, x_{47}$ are enough with very high probability. It follows that the PRNG introduces a negligible overhead compared to HORS. This computational overhead is comparable to few hash function evaluations on small inputs, when SPHINCS signing requires hundreds of thousands of hash function calls [BHH+15].

Comparison to other algorithms

Other algorithms [BC92, RR02] have been proposed to generate k distinct elements among T, but they are not as simple and practical. First, they take as input an integer uniformly distributed between 0 (inclusive) and $\binom{t}{k}$ (exclusive), which is non-trivial because $\binom{t}{k}$ is in general not a power of 2. Second, they require arithmetic operations on large integers up to $\binom{t}{k}$, i.e. 395-bit integers for SPHINCS parameters ($k = 32, t = 2^{16}$). Consequently, Reyzin and Reyzin's algorithm has a complexity of $O(k^2 \log t \log k)$ [RR02]. In contrast, our PORS construction has an average complexity of $O(k)$ for $k \ll t$. Indeed, we need to generate approximately k values and we can test if each value is new in $O(1)$ (e.g. with a hash table).

Besides, PORS offers greater flexibility on the choice of k, whereas HORS constrains it to $k = n/\log_2 t$. PORS can also be used to generate auxiliary data, as we will see in the case of SPHINCS.

Security

A non-adaptive adversary performing a brute-force attack against PORS has at most the following success probability for each message.

$$\frac{(kr)!}{(kr-k)!}\frac{(t-k)!}{t!} = \frac{kr}{t}\cdot\frac{kr-1}{t-1}\cdots\frac{kr-k+1}{t-k+1}$$

This corresponds to the probability that k distinct values uniformly distributed in a set of size t all fall into a subset of size kr. This is lower than the success probability bound against HORS, and the associated security level increases to:

$$\sum_{j=0}^{k-1}\log_2(t-j) - \log_2(kr-j) \geq k(\log_2 t - \log_2(kr))$$

The difference in security is especially large when r is small. For example, with SPHINCS parameters ($k = 32, t = 2^{16}$) and $r = 1$:

$$-\log_2\left[\left(\frac{kr}{t}\right)^k\right] = 352 \quad -\log_2\left[\frac{(kr)!}{(kr-k)!}\frac{(t-k)!}{t!}\right] \approx 394$$

Application to SPHINCS

If we apply the PORS construction to SPHINCS, the best attack of Section 4.4 (search over weak HORST instances) has an increased complexity, namely:

$$\min_r\left[\alpha\left(\sum_{j=0}^{k-1}\log_2(t-j) - \log_2(kr-j)\right) + \max\left\{0,\ -h - \log_2 P_\rho(r)\right\}\right]$$

where $\alpha = 1$ in the classical case and $1/2$ in the quantum case. We obtain 282 bits of classical complexity and 141 bits of quantum complexity (both for $r = 5$).

The naive search attack also has an increased complexity by a few bits, and is still non-competitive. The attack against weak messages (Section 4.4) does not apply to PORS, because there are no weak messages.

4.5.2 SPHINCS leaf selection

As we saw in Section 4.4, SPHINCS suffers from attacks against weak HORST instances. Indeed, even though a honest signer deterministically selects a HORST leaf as a function of the message, a potential forger has full control over this choice because the leaf index is part of the signature. To reduce the attack surface, we propose to extend PORS to also generate this index.

More precisely, to generate a h-bit index i – for a hyper-tree of size 2^h – one can use the first h bits of PORS's PRNG. Namely, we compute $i||x_1||x_2||\ldots = \mathcal{G}(\mathsf{seed})$ where seed is obtained from the message as before. A schematic comparison of textbook SPHINCS and our new construction is shown on Figure 4.3.

We note that our construction also improves flexibility, because h and k are not constrained by the output size of a hash function.

Original SPHINCS with HORST

proc GenPrivKey(1^n)
  $\mathsf{salt} \xleftarrow{\$} B_n$
proc Sign(M)
  $R||i \leftarrow \mathsf{hash}(\mathsf{salt}, M)$
  $x_1||\ldots||x_k \leftarrow \mathsf{split}(\mathsf{hash}(R, M))$
  $\sigma \leftarrow \mathsf{sign}(i, x_1, \ldots, x_k)$
  return (R, i, σ)
proc Verify(M, R, i, σ)
  $x_1||\ldots||x_k \leftarrow \mathsf{split}(\mathsf{hash}(R, M))$
  return $\mathsf{verify}(\sigma, i, x_1, \ldots, x_k)$

SPHINCS with PORST and leaf selection

proc GenPrivKey(1^n)
  $\mathsf{salt} \xleftarrow{\$} B_n$
proc Sign(M)
  $R \leftarrow \mathsf{hash}(\mathsf{salt}, M)$
  $i||x_1||x_2||\ldots \leftarrow \mathcal{G}(\mathsf{hash}(R, M))$
  $\sigma \leftarrow \mathsf{sign}(i, \mathsf{unique}_k(x_1, x_2, \ldots))$
  return (R, σ)
proc Verify(M, R, σ)
  $i||x_1||x_2||\ldots \leftarrow \mathcal{G}(\mathsf{hash}(R, M))$
  return $\mathsf{verify}(\sigma, i, \mathsf{unique}_k(x_1, x_2, \ldots))$

Figure 4.3: Simplified comparison of HORST and PORST in SPHINCS.

Increased security level

With this new construction, attacks targeting a single weak PORST instance have a much higher complexity. The success probability for each message is divided by 2^h, so the classical complexity increases by h bits, and the quantum complexity by $h/2$ bits (for Grover-like attacks). Applied to SPHINCS, the classical complexity is 342 bits and the quantum complexity is 171 bits.

Naive attacks become more efficient in the classical case, with 337 bits of complexity. Intuitively, targeting a single PORST instance is not worth it because the probability to hit it is too low. However, quantum naive search is not competitive, due to memory access and comparison costs, so targeting a single weak PORST instance is still the best strategy.

Overall, replacing HORST by PORST with leaf selection in SPHINCS yields a gain of 67 bits of classical security (from 270 to 337) and 33 bits of post-quantum security (from 138 to 171). As a side effect, removing the leaf index from the signature also saves 8 bytes.

Revised parameters for SPHINCS

Thanks to the better security margin offered by PORST, we propose to decrease SPHINCS parameters to reduce signature size. For an equivalent number of queries $q = 2^{50}$ and similar signature times, we propose to reduce the total height to $h = 50$, divided into $d = 10$ layers of Merkle trees (each of height 5 as in SPHINCS). Signatures for SPHINCS-PORST have a size of 36384 bytes instead of 41000 bytes for textbook SPHINCS, due to the removal of 2 layers of Merkle trees (and of the leaf index).

The corresponding security level is 267 bits classical (naive search) and 136 bits post-quantum (search on weak HORST instance), similar to the original SPHINCS. For a slight improvement, one can also choose to reduce k to 29 instead of 32 to save 1152 more bytes, with 128 bits of post-quantum security.

The total height h can be reduced if the total number of signatures q is smaller, and the number of layers d can be decreased if more computation time is allotted to each signature.

Chapter 5

Octopus: optimal multi-authentication in Merkle trees

Merkle trees [Mer89] are an efficient method to authenticate many values under a small public key consisting of a single hash. With a balanced binary tree, one can authenticate a leaf by revealing an *authentication path* containing at most $\lceil \log_2 N \rceil$ hashes, where N is the number of leaves, and it is easy to show that this construction is optimal.

Yet, in practice one may be interested in authenticating multiple leaves at the same time. This is for example the case in HORST (or PORST), where several values are revealed by the signer. This can also be useful for batch signing; for example one can think of a software repository where each leaf is the hash of a package, and each user subscribes to a set of packages: Alice is interested in updates for a web browser and an image manipulation program, while Bob is interested in a media player and a word processor.

In this multi-authentication scenario, authenticating k leaves can be done by using k full authentication paths. However, this method is suboptimal, as the paths may overlap, especially next to the root. We can visualize this set of paths from leaves to the root as an octopus, with long tentacles that merge close to the root (Figure 5.1). Of course, the shape of this octopus is not fixed but depends on the distribution of leaves to authenticate. If one wants to authenticate k consecutive leaves, the shape is more like a broom, but this extreme case is unlikely when the selected leaves are uniformly distributed.

In SPHINCS [BHH+15], Bernstein et al. proposed an optimization by choosing a threshold level x and only reveal the authentication paths up to this level, additionally revealing all nodes at level x. This removes $kx - 2^x$ hash values from the authentication, hence the optimal choice is $x \approx \log_2(k/\ln(2))$ (with the additional constraint that x must be an integer). However, this does not remove all redundancy, as most nodes at level x can already be inferred from the k paths and some paths may merge below the threshold.

In this chapter, we study the problem of *minimal octopus authentication*, i.e. how many values need to be revealed to authenticate k distinct leaves in a Merkle tree of height h, assuming that the selected leaves are uniformly distributed? We show that only $h - \log_2 k$ values need to be revealed in the best case, and $k(h - \log_2 k)$ values in the worst case. We also derive a recurrence relation to compute the average number of values, and apply it to the parameters proposed in SPHINCS ($k = 32, h = 16$). We conclude that octopus authentication saves 1909 bytes on average for SPHINCS signatures, and 1024 bytes in the worst case.

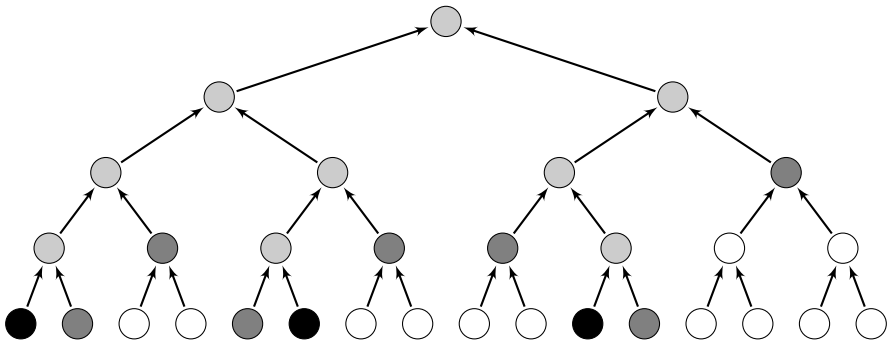


Figure 5.1: Authentication octopus. The leaves in black are the set to authenticate. The nodes in dark grey are revealed in the signature and form the authentication octopus. The nodes in light grey are computed by the verifier to obtain and verify the root.

5.1 Algorithm

We first give an algorithm to compute the optimal set of authentication nodes for a given set of leaf indices. This algorithm can easily be adapted to derive signing and verification algorithms. The algorithm works as described on Figure 5.2.

We start from a set of leaf indices, with the convention that we count indices from 0 to $2^h - 1$, from left to right. The algorithm first sorts this set to facilitate identification of siblings. Then, for each level of the Merkle tree, in a bottom-up order, the algorithm converts the sequence of indices to authenticate at level $\ell + 1$ into a sequence of authentication nodes at level $\ell + 1$ and a sequence of indices to authenticate at level ℓ (Figure 5.3).

More precisely, at a given level, for each index we add its parent to the list of indices at the upper level. We then compute the index of its sibling, by flipping the least significant bit. If the next index to authenticate happens to be the sibling, then we skip the sibling, to avoid adding their common parent twice. Otherwise, we add the sibling to the list of authentication nodes. Because the list of indices is always sorted in increasing order, checking the next index is sufficient to identify siblings.

5.2 Best and worst cases

To analyze the extreme cases, we can rephrase the problem as follows. Starting from k tentacles (i.e. authentication paths) at the bottom of the tree, we obtain a single root. This means that there must be $k - 1$ merges in the octopus. We can now analyze each merge individually.

First, we note that if two tentacles merge at level ℓ, they have identical authentication nodes between level ℓ and the root. Besides, their authentication nodes at level $\ell + 1$ are mutual siblings, hence redundant. This means that if a merge occurs at level ℓ then $\ell + 2$ authentication nodes are redundant (Figure 5.4). To count the total number of redundant nodes in an octopus, we can simply add the redundant nodes of each merge. Indeed, we can construct an octopus by successively adding tentacles; each new tentacle merges at some level ℓ and saves $\ell + 2$ nodes.

In the best case, all merges are close to the leaves, whereas in the worst case all merges

```
proc Octopus([x_1, ..., x_k], h)
    Indices ← sorted([x_1, ..., x_k])
    Auth ← []
    for ℓ = h − 1 down to 0
        NewIndices ← []
        j ← 0
        while j < Indices.length()
            x ← Indices[j]
            NewIndices.append(⌊x/2⌋)
            sibling ← x ⊕ 1
            if j + 1 < Indices.length() ∧ Indices[j + 1] = sibling
                j ← j + 1
            else
                Auth.append((ℓ + 1, sibling))
            j ← j + 1
        Indices ← NewIndices
    return Auth
```

Figure 5.2: Algorithm to compute the optimal authentication octopus. The inputs are the list of leaf indices to authenticate and the Merkle tree height; the result is the list of authentication nodes. Each authentication node contains a level $0 \leq \ell \leq h$ and an index $0 \leq i < 2^\ell$. The sorted() function takes as input a list of integers and this list sorted in increasing order.

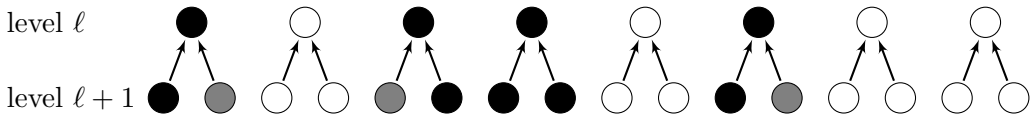


Figure 5.3: One iteration of the optimal octopus algorithm. Starting form a set of nodes to authenticate at level $\ell + 1$ (black), an iteration computes the set of authentication nodes at level $\ell + 1$ (dark grey), and the set of nodes to authenticate at level ℓ, by identification of siblings.

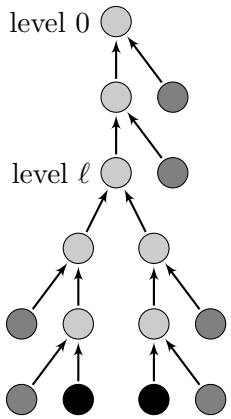


Figure 5.4: Merging of two tentacles at level ℓ. The merge removed $\ell+2$ authentication nodes (dark grey). The authentication nodes are identical at levels 1 to ℓ, and no authentication node is needed at level $\ell+1$.

are close to the root. There are however some constraints because the octopus is embedded in a Merkle tree:

- there cannot be more than 2^ℓ merges at level ℓ;
- if there are $k_{\ell+1}$ tentacles at level $\ell+1$, there cannot be more than $\lfloor k_{\ell+1}/2 \rfloor$ merges at level ℓ.

To simplify the analysis, we first assume that k is a power of two.

Lemma 5. *Let k and h be integers such that k is a positive power of 2 and $k \leq 2^h$. Then, given k leaves to authenticate in a Merkle tree of height h, the octopus authentication algorithm of Figure 5.2 outputs between $h - \log_2 k$ and $k(h - \log_2 k)$ authentication nodes (inclusive).*

Essentially, in the best case the octopus is shaped like a broom with a stick of length $h - \log_2 k$ at the top, whereas in the worst case it is shaped like a rake with k teeth of length $h - \log_2 k$ at the bottom.

Proof. In the worst case, all levels up to $\log_2 k - 1$ are saturated with merges, and the number of redundant nodes is:

$$\sum_{\ell=0}^{\log_2 k - 1} 2^\ell(\ell+2) = k \log_2 k$$

In the best case, there are $k/2$ merges at level $h-1$, $k/4$ merges at level $h-2$, ..., and 1 merge at level $h - \log_2 k$. The number of redundant nodes is:

$$\sum_{\ell=1}^{\log_2 k} \frac{k}{2^\ell}(h - \ell + 2) = (k-1)h + \log_2 k$$

The result follows. □

We now prove the following theorem, in the general case.

Theorem 5. *Let k and h be integers such that $k \leq 2^h$. Then, given k leaves to authenticate in a Merkle tree of height h, if the octopus authentication algorithm of Figure 5.2 outputs oct authentication nodes, then:*

$$h - \lceil \log_2 k \rceil \leq \mathsf{oct} \leq k(h - \lfloor \log_2 k \rfloor)$$

Proof. We let $k' = 2^{\lfloor \log_2 k \rfloor}$ be the largest power of two smaller than or equal to k. In the worst case, all levels up to $\log_2 k' - 1$ are saturated with merges, and level k' contains $k - k'$ merges. By Lemma 5, the number of redundant nodes is:

$$k' \log_2 k' + (k - k')(\log_2 k' + 2) \geq k \log_2 k'$$

so the number of authentication nodes is at most $k(h - \lfloor \log_2 k \rfloor)$

In the best case, all merges are at the bottom levels. In particular, it is possible to merge k tentacles in the $\lceil \log_2 k \rceil$ bottom levels. The only remaining tentacle at level $h - \lceil \log_2 k \rceil$ needs $h - \lceil \log_2 k \rceil$ authentication nodes. The lower bound follows. □

Corollary 3. *Compared to the SPHINCS construction, octopus authentication saves at least k hash values – assuming that x is rounded to $\log_2 k$ in SPHINCS.*

5.3 Average case

To study the average case, we denote by $X(h, k)$ the random variable equal to the minimal number of hash values necessary to authenticate k uniformly distributed leaves in a Merkle tree of height h. We denote by $T(h, k)$ the expectation of $X(h, k)$, i.e. the average number of hash values. We adopt a bottom-up approach to derive a recurrence relation between consecutive levels, i.e. $T(h, \cdot)$ and $T(h + 1, \cdot)$, and can then solve the problem by dynamic programming.

We also denote by $P(h, k, i)$ the probability that given k uniformly distributed tentacles at level $h + 1$, i merges occur at level h.

Lemma 6. *The probability $P(h, k, i)$ is equal to:*

$$P(h, k, i) = \binom{2^{h+1}}{k}^{-1} \binom{2^h}{k - i} \binom{k - i}{i} 2^{k-2i}$$

Proof. There are $\binom{2^{h+1}}{k}$ sets of k distinct indices at level $h + 1$. At level h, there are $\binom{2^h}{k-i}$ choices of $k - i$ merged slots, out of which $\binom{k-i}{i}$ choices of i slots that contain a merge. For each of the $k - 2i$ non merged slots at level h, there are two possible slots at level $h + 1$. □

Theorem 6. *$T(h, k)$ satisfies the following recurrence relation:*

$$T(0, 1) = 0$$
$$T(h + 1, k) = \sum_{i=0}^{\lfloor k/2 \rfloor} (k - 2i + T(h, k - i))\, P(h, k, i)$$

Proof. First, $T(0,1) = 0$, because no authentication node is needed for a tree reduced to one node.

We now remark that if k leaf indices are uniformly distributed, and that they have merged into t tentacles at some upper level ℓ, these t tentacles are also uniformly distributed at level ℓ. This is independent of how the k merged into t, so we can view the subtree above level ℓ as a standalone tree of height ℓ.

This allows to derive the recurrence relation between consecutive levels. Indeed, i merges occur at level h with probability $P(h,k,i)$. In that case, $k-2i$ authentication nodes are necessary at level $h+1$ to authenticate the $k-2i$ non-merged tentacles, and $T(h,k-i)$ authentication nodes are necessary at upper levels on average. □

Standard deviation We can also derive a recurrence relation to compute the standard deviation of $X(h,k)$. We recall that it is equal to $\sqrt{T^{(2)}(h,k) - T(h,k)^2}$, where $T^{(2)}(h,k)$ denotes the expectation of $X(h,k)^2$. We can compute $T^{(2)}$ with the following recurrence relation, and then derive the standard deviation.

$$T^{(2)}(h+1,k) = \sum_{i=0}^{\lfloor k/2 \rfloor} \Big((k-2i)^2 + 2(k-2i)T(h,k-i) + T^{(2)}(h,k-i) \Big) P(h,k,i)$$

5.4 Application

We can solve the recurrences by dynamic programming. For the proposed parameters in SPHINCS, we obtain $T(h=16,k=32) \approx 324.3$, with a standard deviation of 7.1. In contrast, the HORST construction in SPHINCS uses $k(h-x)+2^x = 384$ authentication values (both for $x=5$ and $x=6$). Even in the worst case, our octopus authentication uses only $k(h-\log_2 k) = 352$ authentication values.

Consequently, octopus authentication saves 1024 bytes in the worst case and 1909 bytes on average, compared to the threshold method proposed for HORST in SPHINCS. Compared to the naive method of using k full authentication paths, octopus authentication saves 6005 bytes on average.

Chapter 6

Back to collision resistance: security reductions of mask-less constructions

Early versions of Merkle trees and Winternitz OTS used a simple hash function to compute the value of each node from its children. More recently, masks were introduced in Merkle trees [DOTV08] and Winternitz OTS [Hül13] to reduce the security of such constructions to the second-preimage resistance instead of the collision resistance of the underlying hash function. Given an output of n bits, generic classical attacks against collision resistance – birthday attacks in $\Theta(2^{n/2})$ – are indeed faster than generic classical attacks against second-preimage resistance – in $\Theta(2^n)$ – so relying on second-preimage resistance allows to use a smaller output size n.

However, in the post-quantum world, Bernstein [Ber09] has shown that generic attacks against second-preimage resistance and collision resistance are both in $\Theta(2^{n/2})$, if one takes into account all costs (including hardware, communication, etc.). Yet, when we read carefully the proofs of security of Merkle trees [DOTV08], reduction to second-preimage resistance is less tight than collision resistance, with a factor being the total number of hash evaluations in the construction (e.g. 2^H for a Merkle tree of height H). The reason is that in order to transform a signature forger into a second preimage finder one has to "guess" where the forger will invert a value. A recent proof for the XMSS-T construction has gotten rid of this factor [HRS16], but this proof models the underlying hash function as an unbreakable random oracle, which does not appear in the security reduction. On the contrary, a collision finder can accept a collision anywhere.

Apart from these security considerations, masks introduce more complexity in the signature scheme, which is undesirable from an implementation point-of-view, as it increases the risk of implementation discrepancies, bugs, etc. Masks also slightly increase the size of public keys, although XMSS-T reduced them to a single seed. Requiring collision resistance may rule out potentially faster second-preimage-resistant functions such as 5-round Haraka-v2 [KLMR16]. But according to its designers, Haraka-v2 becomes collision-resistant with only one additional round. Essentially, switching from second-preimage resistance to collision resistance replaces a XOR mask by a round. Even though there may be room for improvement in the design on second-preimage-resistant functions, it seems that their performance should

remain similar to that of collision-resistant functions for the same security level.

In short, coming back to collision-resistant hash functions allows a simpler design and harms neither security nor performance in the post-quantum world. In this chapter, we review constructions based on collision-resistant hash functions, and clarify their proofs of security in the standard model. We first review basic building blocks: Winternitz OTS (Section 6.1) and schemes based on the *obtain a random subset* (ORS) construction (Section 6.2). We then study generic transformations: seed-based secret key (Section 6.3), Merkle tree (Section 6.4), hyper trees (Section 6.5) and batch signing (Section 6.6).

6.1 Mask-less Winternitz OTS

In [Hül13], Hülsing proposed WOTS$^+$, a version of Winternitz OTS with XOR masks in the hash chains, and provided a quite complex security proof. Even though a proper security analysis of WOTS is non-trivial, we aim at providing a simpler proof for mask-less Winternitz OTS, with tighter constants. A tight and simple security reduction is desirable for WOTS, because it is a building block of much more complex schemes such as XMSS [BDH11] and SPHINCS [BHH$^+$15].

In [DSS05], Dods et al. proposed a generic security reduction of OTS schemes based on directed acyclic graphs (DAGs) to one-wayness, collision resistance and undetectability of the underlying functions. The simple version of WOTS, which iterates a hash function without introducing masks, is a particular case of DAG-based OTS schemes. However, Dods et al.'s analysis can be refined in the case of Winternitz' graph to obtain a tighter security reduction.

6.1.1 Reduction of WOTS to collision resistance

In this chapter, we use the notations defined in Section 1.1. In particular, we recall that B_n is the set of n-bit blocks $\{0,1\}^n$.

We now define the Winternitz checksum with the $\mathsf{checksummed}$ function, and give a definition of the WOTS signature scheme.

Definition 13 (Winternitz checksum)**.** *Given a security parameter n and a power-of-2 chain length w, we let:*

$$\mu = n/\log_2 w$$
$$\ell = \mu + \lfloor \log_2 \mu(w-1)/\log_2 w \rfloor + 1$$

We define the function $\mathsf{checksummed} : B_n \to \{0,\ldots,w-1\}^\ell$ as follows. Given an n-bit input x, split x into μ blocks of $\log_2 w$ bits $(x_1,\ldots,x_\mu)$, compute the checksum $C(x) = \sum_{i=1}^{\mu} w-1-x_i$ encoded as a $(\ell-\mu)\log_2 w$ bit string and return $x||C(x)$ split into ℓ blocks of $\log_2 w$ bits.

Definition 14 (Mask-less Winternitz OTS)**.** *Given a security parameter n, a power-of-2 chain length w, and a family $\mathcal{F}_n$ of functions $F : B_n \to B_n$, the mask-less Winternitz OTS is defined over:*

- *the message space $\mathcal{M} = B_n$,*

- *the public key space* $\mathcal{PK} = B_n^\ell$,
- *the secret key space* $\mathcal{SK} = B_n^\ell$,
- *the signature space* $\mathcal{SG} = B_n^\ell$,

by the following algorithms:

- *key generation* $\mathcal{KG}(1^n)$ *is* $(sk \xleftarrow{\$} \mathcal{SK}; pk \leftarrow (F^{w-1}(sk_i))_{1\leq i\leq \ell}; (pk, sk))$;
- *signing* $\mathcal{S}(sk, M)$ *is* $((x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M); \sigma \leftarrow (F^{x_i}(sk_i))_{1\leq i\leq \ell}; \sigma)$;
- *public key extraction* $\mathcal{E}(M, \sigma)$ *is* $((x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M); (F^{w-1-x_i}(\sigma_i))_{1\leq i\leq \ell})$.

We now reduce EU-CMA security of WOTS to the security properties of $\mathcal{F}_n$. Security reductions in [DSS05] and [Hül13] essentially rely on the same principles: an adversary that outputs a forgery either finds a collision in the underlying hash function or inverts it, provided that the hash function is undetectable. We clarify the reduction in the specific case of maskless WOTS.

Theorem 7. *Let $\mathcal{F}_n$ be a hash function family. We consider the following resources ξ: the time τ, the number of queries to the signature scheme q and the number of queries to $\mathcal{F}_n$ q_F. The unforgeability of the Winternitz OTS based on $\mathcal{F}_n$ can be bounded by the undetectability, one-wayness and collision resistance of $\mathcal{F}_n$:*

$$\begin{aligned}&\textsc{InSec}^{\text{EU-CMA}}(\text{WOTS}(\mathcal{F}_n); \tau, q = 1, q_F)\\ &\leq (w-1)\ell \left[\frac{w-2}{2}\textsc{InSec}^{\text{UD}}(\mathcal{F}_n; \tau', q'_F) + \textsc{InSec}^{\text{OW}}(\mathcal{F}_n; \tau', q'_F) + \textsc{InSec}^{\text{CR}}(\mathcal{F}_n; \tau', q'_F)\right]\end{aligned}$$

where $q'_F = q_F + (w-1)(2\ell+1)$ and $\tau' = \tau + c \cdot (w-1)\ell$ for some constant c.

Essentially, security degrades with a factor $(w-1)\ell$ because an adversary can try to invert a value or find a collision anywhere in the hash chains. Additionally, security decreases with the chain length w if the hash function is not undetectable: intuitively if the hash function output is not uniformly distributed, this bias grows after many iterations and it becomes easier to invert.

Compared to previous results [Hül13], we obtained a tighter bound with: a main factor of $w-1$ instead of w, a factor of $(w-2)/2$ instead of w for the undetectability reduction, and a term of 1 times the collision resistance instead of w times the second-preimage resistance.

Proof. Let $\mathcal{A}$ be a WOTS adversary using at most resources $(\tau, q = 1, q_F)$. We consider the games WOTS, G_1 and G_2 on Figure 6.1. WOTS corresponds to existential unforgeability of Winternitz OTS; variants G_1 and G_2 allow to reduce the security to properties of $\mathcal{F}_n$.

By Lemmas 7, 8 and 9, we obtain the following bounds:

$$\begin{aligned}\mathsf{Pr}[\mathcal{A}^{\text{WOTS}}_{\mathcal{F}_n} \Rightarrow 1] &\leq (w-1)\ell\mathsf{Pr}[\mathcal{A}^{G_1}_{\mathcal{F}_n} \Rightarrow 1]\\ \mathsf{Pr}[\mathcal{A}^{G_2}_{\mathcal{F}_n} \Rightarrow 1] &\leq \textsc{InSec}^{\text{OW}}(\mathcal{F}_n; \tau', q'_F) + \textsc{InSec}^{\text{CR}}(\mathcal{F}_n; \tau', q'_F)\\ \left|\mathsf{Pr}[\mathcal{A}^{G_2}_{\mathcal{F}_n} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}^{G_1}_{\mathcal{F}_n} \Rightarrow 1]\right| &\leq \frac{w-2}{2}\textsc{InSec}^{\text{UD}}(\mathcal{F}_n; \tau', q'_F)\end{aligned}$$

The result follows. □

Game WOTS

proc Initialize
  $F \xleftarrow{\$} \mathcal{F}_n$
proc GenKey(1^n)
  $(s_1, \ldots, s_\ell) \xleftarrow{\$} B_n^\ell$
  for $i \in \{1, \ldots, \ell\}$ **do** $p_i \leftarrow F^{w-1}(s_i)$
  return $(p_1, \ldots, p_\ell)$
proc Sign(M)
  $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M)$
  for $i \in \{1, \ldots, \ell\}$ **do** $y_i \leftarrow F^{x_i}(s_i)$
  return $(y_1, \ldots, y_\ell)$
proc Finalize$(M, y_1, \ldots, y_\ell)$
  $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M)$
  $\mathsf{win} \leftarrow \bigwedge_{i=1}^{\ell} F^{w-1-x_i}(y_i) = p_i$
  return win

Adversary $\mathcal{B}$

$F, v \leftarrow$ **Initialize**
Run $\mathcal{A}$
On query **GenKey**(1^n) of $\mathcal{A}$
  $(p_1, \ldots, p_\ell) \leftarrow$ WOTS.**GenKey**(1^n)
  $(\alpha, \beta) \xleftarrow{\$} \{1, \ldots, \ell\} \times \{1, \ldots, w-1\}$
  $p_\alpha \leftarrow F^{w-1-\beta}(v)$
  return $(p_1, \ldots, p_\ell)$ **to** $\mathcal{A}$
On query **Sign**(M) of $\mathcal{A}$
  return G_2.**Sign**(M) **to** $\mathcal{A}$
On **Finalize**$(M, y_1, \ldots, y_\ell)$ of $\mathcal{A}$
  $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M)$
  if $x_\alpha \geq \beta$ **then abort**
  $u \leftarrow F^{\beta - x_\alpha - 1}(y_\alpha)$
  if $F(u) = v$ **then output** (preimg, u)
  for $i \in \{1, \ldots, w-1-\beta\}$
    if $F^i(u) \neq F^{i-1}(v) \wedge F^{i+1}(u) = F^i(v)$
      output $(\mathsf{collision}, F^i(u), F^{i-1}(v))$
  abort

Games G_1 and $\boxed{G_2}$

proc Initialize
  $F \xleftarrow{\$} \mathcal{F}_n$
proc GenKey(1^n)
  $(p_1, \ldots, p_\ell) \leftarrow$ WOTS.**GenKey**(1^n)
  $(\alpha, \beta) \xleftarrow{\$} \{1, \ldots, \ell\} \times \{1, \ldots, w-1\}$
  $\boxed{u \xleftarrow{\$} B_n;\ v \leftarrow F(u)}$
  $\boxed{p_\alpha \leftarrow F^{w-1-\beta}(v)}$
  return $(p_1, \ldots, p_\ell)$
proc Sign(M)
  $(y_1, \ldots, y_\ell) \leftarrow$ WOTS.**Sign**(M)
  if $x_\alpha < \beta$ **then abort**
  $\boxed{y_\alpha \leftarrow F^{x_\alpha - \beta}(v)}$
  return $(y_1, \ldots, y_\ell)$
proc Finalize$(M, y_1, \ldots, y_\ell)$
  $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M)$
  if $x_\alpha \geq \beta$ **then abort**
  $\mathsf{win} \leftarrow \bigwedge_{i=1}^{\ell} F^{w-1-x_i}(y_i) = p_i$
  return win

Figure 6.1: Games WOTS, G_1 and G_2 and adversary $\mathcal{B}$ against preimage and collision resistance of $\mathcal{F}_n$. Boxed statements are present only in game G_2. An adversary against games WOTS, G_1 and G_2 is forbidden to submit to **Finalize** a message M that it already queried to **Sign**.

Lemma 7. *Let $\mathcal{A}$ be a* WOTS *adversary. We have the following relation between the success probability of $\mathcal{A}$ against games* WOTS *and G_1 (Figure 6.1).*

$$\mathsf{Pr}[\mathcal{A}^{\mathrm{WOTS}}_{\mathcal{F}_n} \Rightarrow 1] \leq (w-1)\ell \mathsf{Pr}[\mathcal{A}^{G_1}_{\mathcal{F}_n} \Rightarrow 1]$$

Essentially, games WOTS and G_1 differ by guessing a value that an adversary would invert in its forgery. Lemmas 8 and 9 will study the actual reduction to one-wayness, collision resistance and undetectability.

Proof. Let's first assume that $\mathcal{A}$ queries a message M yielding a signature $(y_1, \ldots, y_\ell)$ and successfully forges a signature $(y'_1, \ldots, y'_\ell)$ for $M' \neq M$ in game WOTS. Then, by construction of the Winternitz checksum, there exists α such that $x'_\alpha < x_\alpha$. If game G_1 picks $(\alpha, \beta = x_\alpha)$ – which happens with probability $1/(w-1)\ell$ and is independent of the choices of $\mathcal{A}$ – then $\mathcal{A}$ wins G_1.

Likewise, if $\mathcal{A}$ directly forges a signature $(y'_1, \ldots, y'_\ell)$ without querying a message, then by construction of the Winternitz checksum there exists α such that $x'_\alpha < w-1$. If game G_1 picks $(\alpha, \beta = w-1)$ then $\mathcal{A}$ wins. The result follows. □

Lemma 8. *Let $\mathcal{A}$ be a* WOTS *adversary. We have the following bound on the success probability of $\mathcal{A}$ against game G_2 (Figure 6.1).*

$$\mathsf{Pr}[\mathcal{A}^{G_2}_{\mathcal{F}_n} \Rightarrow 1] \leq \textsc{InSec}^{\mathrm{OW}}(\mathcal{F}_n; \tau', q'_F) + \textsc{InSec}^{\mathrm{CR}}(\mathcal{F}_n; \tau', q'_F)$$

where $q'_F = q_F + (w-1)(2\ell+1)$ and $\tau' = \tau + c \cdot (w-1)\ell$ for some constant c.

Essentially, game G_2 can only be won by inverting the target value v or by finding a collision.

Proof. We construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ against the collision resistance and one-wayness of the underlying hash function family $\mathcal{F}_n$ (Figure 6.1). $\mathcal{B}$ uses at most resources (τ', q'_F): besides $\mathcal{A}$'s queries to $\mathcal{F}_n$, it makes at most $(w-1)\ell$ queries in **GenKey**, $(w-1)\ell$ queries in **Sign** and $w-1$ in **Finalize**, i.e. $(w-1)(2\ell+1)$ additional queries to $\mathcal{F}_n$, in a time proportional to that.

Given a challenge v, $\mathcal{B}$ either returns a preimage of v for F, two colliding messages for F, or aborts. We note that when the challenge v is equal to $F(u)$ where u is uniformly distributed in B_n, $\mathcal{B}$ perfectly simulates game G_2 for $\mathcal{A}$. We now show that:

$$\mathsf{Pr}[\mathcal{A}^{G_2}_{\mathcal{F}_n} \Rightarrow 1] \leq \mathsf{Pr}[\mathcal{B}_{\mathcal{F}_n} \Rightarrow \mathsf{preimg} \vee \mathsf{collision}]$$

Indeed, if $\mathcal{A}$ wins game G_2 with a forgery $(y_1, \ldots, y_\ell)$, then $F^{w-1-x_\alpha}(y_\alpha) = p_\alpha$, which implies $F^{w-\beta}(u) = F^{w-\beta}(F^{\beta-x_\alpha-1}(y_\alpha)) = p_\alpha = F^{w-1-\beta}(v)$. Therefore it must be the case that either $F(u) = v$ (a preimage is found) or it exists $1 \leq i \leq w-1-\beta$ such that $F^i(u) \neq F^{i-1}(v)$ and $F^{i+1}(u) = F^i(v)$ (a collision is found). □

Lemma 9. *Let $\mathcal{A}$ be a* WOTS *adversary. We have the following relation between the success probabilities of $\mathcal{A}$ against games G_1 and G_2 (Figure 6.1).*

$$\left| \mathsf{Pr}[\mathcal{A}^{G_2}_{\mathcal{F}_n} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}^{G_1}_{\mathcal{F}_n} \Rightarrow 1] \right| \leq \frac{w-2}{2} \textsc{InSec}^{\mathrm{UD}}(\mathcal{F}_n; \tau', q'_F)$$

where $q'_F = q_F + (w-1)(2\ell+1)$ and $\tau' = \tau + c \cdot (w-1)\ell$ for some constant c.

Game $G_3(\beta, j)$

proc Initialize
  $F \xleftarrow{\$} \mathcal{F}_n$
proc GenKey(1^n)
  $(p_1, \ldots, p_\ell) \leftarrow$ WOTS.**GenKey**(1^n)
  $\alpha \xleftarrow{\$} \{1, \ldots, \ell\}$
  $u \xleftarrow{\$} B_n$; $v \leftarrow F(u)$
  $p_\alpha \leftarrow F^{w-1-j}(v)$
  return $(p_1, \ldots, p_\ell)$
proc Sign(M)
  $(y_1, \ldots, y_\ell) \leftarrow$ WOTS.**Sign**(M)
  if $x_\alpha < \beta$ **then abort**
  $y_\alpha \leftarrow F^{x_\alpha - j}(v)$
  return $(y_1, \ldots, y_\ell)$
proc Finalize$(M, y_1, \ldots, y_\ell)$
  $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M)$
  if $x_\alpha \geq \beta$ **then abort**
  $\mathsf{win} \leftarrow \bigwedge_{i=1}^{\ell} F^{w-1-x_i}(y_i) = p_i$
  return win

Adversary $\mathcal{C}_{\beta,j}$

$F, v \leftarrow$ **Initialize**
Run $\mathcal{A}$
On query **GenKey**(1^n) of $\mathcal{A}$
  $(p_1, \ldots, p_\ell) \leftarrow$ WOTS.**GenKey**(1^n)
  $\alpha \xleftarrow{\$} \{1, \ldots, \ell\}$
  $p_\alpha \leftarrow F^{w-1-j}(v)$
  return $(p_1, \ldots, p_\ell)$ **to** $\mathcal{A}$
On query **Sign**(M) of $\mathcal{A}$
  $(y_1, \ldots, y_\ell) \leftarrow$ WOTS.**Sign**(M)
  if $x_\alpha < \beta$ **then abort**
  $y_\alpha \leftarrow F^{x_\alpha - j}(v)$
  return $(y_1, \ldots, y_\ell)$ **to** $\mathcal{A}$
On **Finalize**$(M, y_1, \ldots, y_\ell)$ of $\mathcal{A}$
  $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(M)$
  if $x_\alpha \geq \beta$ **then abort**
  output $\bigwedge_{i=1}^{\ell} F^{w-1-x_i}(y_i) = p_i$

Figure 6.2: Games $G_3(\beta, j)$ for $1 \leq j \leq \beta \leq w-1$ and adversaries $\mathcal{C}_{\beta,j}$ against undetectability of $\mathcal{F}_n$. An adversary against these games is forbidden to submit to **Finalize** a message M that it already queried to **Sign**.

Essentially, if the hash function is not undetectable, putting a preimage challenge in the middle of a chain can be detected by an adversary, because the distribution of the associated public value will change.

Proof. For $1 \leq j \leq \beta \leq w-1$, we introduce the hybrid games $G_3(\beta, j)$ (Figure 6.2). We first note that $G_3(\beta, \beta)$ simulates G_2 and $G_3(\beta, 1)$ simulates G_1:

$$\mathsf{Pr}[\mathcal{A}^{G_2}_{\mathcal{F}_n} \Rightarrow 1] = \sum_{\beta=1}^{w-1} \frac{1}{w-1} \mathsf{Pr}[\mathcal{A}^{G_3(\beta,\beta)}_{\mathcal{F}_n} \Rightarrow 1]$$

$$\mathsf{Pr}[\mathcal{A}^{G_1}_{\mathcal{F}_n} \Rightarrow 1] = \sum_{\beta=1}^{w-1} \frac{1}{w-1} \mathsf{Pr}[\mathcal{A}^{G_3(\beta,1)}_{\mathcal{F}_n} \Rightarrow 1]$$

Consequently:

$$\left|\mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_2} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_1} \Rightarrow 1]\right| \leq \sum_{\beta=1}^{w-1} \frac{1}{w-1} \left|\mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_3(\beta,\beta)} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_3(\beta,1)} \Rightarrow 1]\right|$$
$$\leq \sum_{\beta=1}^{w-1} \frac{1}{w-1} \sum_{j=1}^{\beta-1} \left|\mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_3(\beta,j+1)} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_3(\beta,j)} \Rightarrow 1]\right|$$

Besides, given $1 \leq j < \beta \leq w-1$, we consider adversary $\mathcal{C}_{\beta,j}$ against undetectability of $\mathcal{F}_n$ (Figure 6.2). $\mathcal{C}_{\beta,j}$ uses at most resources (τ', q'_F). If $\mathcal{C}_{\beta,j}$ is given as input a challenge $v = F(u)$ where u is uniformly distributed in B_n, it perfectly simulates game $G_3(\beta, j)$ for $\mathcal{A}$. If $\mathcal{C}_{\beta,j}$ is given a challenge v uniformly distributed in B_n, it perfectly simulates game $G_3(\beta, j+1)$ for $\mathcal{A}$. Therefore the advantage of $\mathcal{C}_{\beta,j}$ against undetectability of $\mathcal{F}_n$ is the following:

$$\left|\mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_3(\beta,j+1)} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_3(\beta,j)} \Rightarrow 1]\right| = \textsc{Succ}^{\textsc{ud}}_{\mathcal{F}_n}(\mathcal{C}_{\beta,j}) \leq \textsc{InSec}^{\textsc{ud}}(\mathcal{F}_n; \tau', q'_F)$$

Consequently:

$$\left|\mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_2} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}_{\mathcal{F}_n}^{G_1} \Rightarrow 1]\right| \leq \sum_{\beta=1}^{w-1} \frac{\beta-1}{w-1} \textsc{InSec}^{\textsc{ud}}(\mathcal{F}_n; \tau', q'_F) = \frac{w-2}{2} \textsc{InSec}^{\textsc{ud}}(\mathcal{F}_n; \tau', q'_F)$$

The result follows. □

6.1.2 L-tree WOTS

In practice, more complex structures that use WOTS – such as XMSS or SPHINCS – use a so-called *L-tree* to compress the ℓ values of the public key into a single n-bit string. This *L-tree* is a binary tree, but not perfectly balanced because the number of leaves is not a power of two (due to the Winternitz checksum).

Definition 15 (L-tree)**.** *Given a hash function $H : B_n^2 \to B_n$, the function* L-tree *compresses a list of n-bit blocks into a single n-bit block. It is defined by recurrence as:*

$$\mathsf{L\text{-}tree}(H, x_1) = x_1$$
$$\mathsf{L\text{-}tree}(H, x_1, \ldots, x_{2i+2}) = \mathsf{L\text{-}tree}(H, H(x_1, x_2), \ldots, H(x_{2i+1}, x_{2i+2}))$$
$$\mathsf{L\text{-}tree}(H, x_1, \ldots, x_{2i+3}) = \mathsf{L\text{-}tree}(H, H(x_1, x_2), \ldots, H(x_{2i+1}, x_{2i+2}), x_{2i+3})$$

Remark For our security proofs, the exact shape of the L-tree is irrelevant as long as it is a binary tree, because in any case the root is computed from the ℓ leaves with $\ell - 1$ applications of the hash function.

Definition 16 (Mask-less LWOTS)**.** *Given parameters n, w, the associated Winternitz ℓ, a family $\mathcal{F}_n$ of functions $F : B_n \to B_n$, and a family $\mathcal{H}_n$ of hash functions $H : B_n^2 \to B_n$, the mask-less L-tree Winternitz OTS is defined over $\mathcal{M} = B_n$, $\mathcal{PK} = B_n$, $\mathcal{SK} = B_n^\ell$, $\mathcal{SG} = B_n^\ell$ by the following algorithms:*

- *key generation $\mathcal{KG}(1^n)$ is $((pk, sk) \xleftarrow{\$} \text{WOTS}.\mathcal{KG}(1^n); (\mathsf{L\text{-}tree}(H, pk), sk))$;*

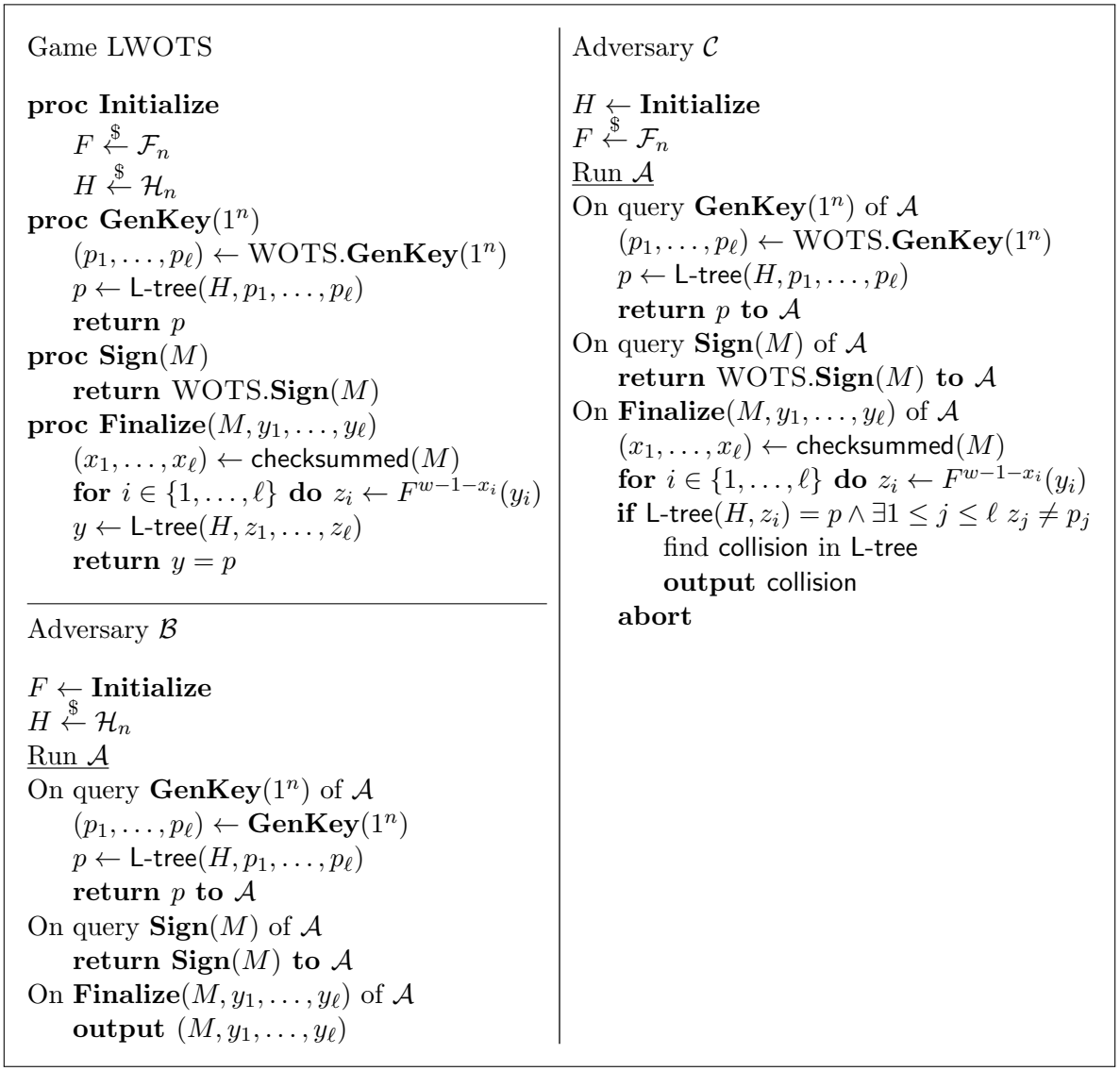


Figure 6.3: Game LWOTS, adversary $\mathcal{B}$ against game WOTS and adversary $\mathcal{C}$ against collision resistance of $\mathcal{H}_n$. An adversary against game LWOTS is forbidden to submit to **Finalize** a message M that it already queried to **Sign**.

- *signing* $\mathcal{S} = \text{WOTS}.\mathcal{S}$*;*
- *public key extraction* $\mathcal{E}(M, \sigma)$ *is* $((y_1, \ldots, y_\ell) \leftarrow \text{WOTS}.\mathcal{E}(M, \sigma); \mathsf{L\text{-}tree}(H, y_1, \ldots, y_\ell))$.

Theorem 8. *Let* $\mathcal{F}_n$ *and* $\mathcal{H}_n$ *be function families. We consider the following resources* ξ*: the time* τ*, the number of queries to the signature scheme* q *and the number of queries to* $\mathcal{F}_n$ *and* $\mathcal{H}_n$ *respectively* q_F *and* q_H*. The unforgeability of the L-tree Winternitz OTS based on* $\mathcal{F}_n$ *and* $\mathcal{H}_n$ *can be bounded by the unforgeability of* $\text{WOTS}(\mathcal{F}_n)$ *and the collision resistance of* $\mathcal{H}_n$*:*

$$\begin{aligned}&\textsc{InSec}^{\text{EU-CMA}}(\text{LWOTS}(\mathcal{F}_n, \mathcal{H}_n); \tau, q = 1, q_F, q_H)\\ &\leq \textsc{InSec}^{\text{EU-CMA}}(\text{WOTS}(\mathcal{F}_n); \tau', q = 1, q_F) + \textsc{InSec}^{\text{CR}}(\mathcal{H}_n; \tau'', q''_H)\end{aligned}$$

where $q''_H = q_H + 2(\ell - 1)$*,* $\tau' = \tau + c \cdot (\ell - 1)$ *and* $\tau'' = \tau + c \cdot (w - 1)\ell$ *for some constant* c*.*

Essentially, an adversary can forge a signature for LWOTS either by forging the underlying WOTS or by finding an alternative L-tree that hashes to the same root, hence finding a hash collision.

Proof. Let $\mathcal{A}$ be an adversary for LWOTS (Figure 6.3). We construct adversaries $\mathcal{B}$ and $\mathcal{C}$ (Figure 6.3). $\mathcal{B}$ uses at most resources (τ', q, q_F): it spends an additional time proportional to $\ell - 1$ to compute the L-tree root. $\mathcal{C}$ uses at most resources (τ'', q''_h): it makes $2(\ell - 1)$ additional calls to $\mathcal{H}_n$ to compute L-trees in **GenKey** and **Finalize**, and spends time proportional to $(w - 1)\ell$ to compute hash chains in **Finalize**.

We note that $\mathcal{B}$ and $\mathcal{C}$ both perfectly simulate game LWOTS for $\mathcal{A}$. Besides, if $\mathcal{A}$ wins game LWOTS, then the L-tree root of the forgery matches the public key and we can consider two cases:

- the L-tree leaves $(z_1, \ldots, z_\ell)$ correspond to the WOTS public values $(p_1, \ldots, p_\ell)$, and $\mathcal{B}$ wins game WOTS;
- there is a mismatch $z_i \neq p_i$, so by the following Lemma 10, a collision for H must exist in the L-tree, and $\mathcal{C}$ successfully outputs it.

Consequently:

$$\mathsf{Pr}[\mathcal{A}^{\text{LWOTS}}_{\mathcal{F}_n, \mathcal{H}_n} \Rightarrow 1] \leq \mathsf{Pr}[\mathcal{B}^{\text{WOTS}}_{\mathcal{F}_n} \Rightarrow 1] + \mathsf{Pr}[\mathcal{C}^{\text{CR}}_{\mathcal{H}_n} \Rightarrow 1]$$

The result follows. □

Definition 17 (Binary hash tree)**.** *Given a hash function* $H : B_n^2 \to B_n$*, a tree* $T = (t, t_1, \ldots, t_m)$ *of root* $t \in B_n$ *and leaves* $t_1, \ldots, t_m \in B_n$ *is called* binary hash tree *if each internal node* u *is the hash of its children* (u_ℓ, u_r) *by* H*, i.e.* $u = H(u_\ell, u_r)$*.*

For example, Merkle trees and L-trees are binary hash trees.

Lemma 10. *Let* A, B *be two binary hash trees of the same shape such that* $a = b$ *but* $(a_1, \ldots, a_m) \neq (b_1, \ldots, b_m)$*, i.e. their roots are equal but at least a pair of leaves* (a_i, b_i) *differ. Then there exist a pair of internal nodes* $(u, v) \in A \times B$ *such that* $u = v$ *and* $(u_\ell, u_r) \neq (v_\ell, v_r)$*. In particular,* $u_\ell || u_r$ *and* $v_\ell || v_r$ *are a collision for* H*.*

Proof. Let $(a_i^0, \ldots, a_i^p)$ and $(b_i^0, \ldots, b_i^p)$ be the sequences of ancestors of a_i and b_i. Then $a_i^0 = a_i \neq b_i = b_i^0$ and $a_i^p = a = b = b_i^p$, so there exists $1 \leq j \leq p$ such that $a_i^{j-1} \neq b_i^{j-1}$ and $a_i^j = b_i^j$. The pair $(u, v) = (a_i^j, b_i^j)$ verifies the property. □

6.2 Mask-less ORS

We now study the security of constructions from the ORS family (obtain a random subset), be it HORS (hash to obtain a random subset) or PORS (PNRG to obtain a random subset), with or without tree. We formalize and generalize results by Reyzin and Reyzin [RR02] and Bernstein et al. [BHH+15].

6.2.1 ORS (without tree)

Given parameters k and t, we let $T = \{1, \ldots, t\}$ and we recall that we denote by $\mathcal{P}_k(T)$ the subsets of T of size at most k.

Definition 18 (Non-adaptive subset resilience)**.** *Given a security parameter n, a set size t, a subset size $k \leq t$, a number of queries r, and a family of functions to obtain random subsets $\mathcal{O}_n = \{\mathsf{ORS}_K : B_n \to \mathcal{P}_k(T) | K \in B_n\}$, the advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{r+1})$ against r-non-adaptive subset resilience of $\mathcal{O}_n$ is:*

$$\begin{aligned}\textsc{Succ}^{r-\text{NASR}}_{\mathcal{O}_n}(\mathcal{A}) = \Pr\Big[& G \xleftarrow{\$} (B_n \to B_n); \\ & M_1 \leftarrow \mathcal{A}_1(); \\ & K_1 \leftarrow G(M_1); \\ & M_2 \leftarrow \mathcal{A}_2(K_1, M_1); \\ & \ldots; \\ & M_r \leftarrow \mathcal{A}_r(K_1, M_1, \ldots, K_{r-1}, M_{r-1}); \\ & K_r \leftarrow G(M_r); \\ & (K', M') \leftarrow \mathcal{A}_{r+1}(K_1, M_1, \ldots, K_r, M_r) \\ & : \mathsf{ORS}_{K'}(M') \subseteq \mathsf{ORS}_{K_1}(M_1) \cup \ldots \cup \mathsf{ORS}_{K_r}(M_r)\Big]\end{aligned}$$

Definition 19 (Mask-less ORS signature)**.** *Given a security parameter n, a set size t, a subset size $k \leq t$, a family $\mathcal{F}_n$ of functions $F : B_n \to B_n$, a family $\mathcal{G}_n$ of functions $G_{\mathsf{salt}} : B_n \to B_n$, and a family of functions to obtain random subsets $\mathcal{O}_n = \{\mathsf{ORS}_K : B_n \to \mathcal{P}_k(T) | K \in B_n\}$, the mask-less ORSS is defined over $\mathcal{M} = B_n$, $\mathcal{PK} = B_n^t$, $\mathcal{SK} = B_n \times B_n^t$, $\mathcal{SG} = B_n \times B_n^k$ by the following algorithms:*

- *key generation $\mathcal{KG}(1^n)$ is $((\mathsf{salt}, sk) \xleftarrow{\$} \mathcal{SK}; pk \leftarrow (F(sk_i))_{1 \leq i \leq t}; (pk, (\mathsf{salt}, sk)))$;*
- *signing $\mathcal{S}((\mathsf{salt}, sk), M)$ is*

$$(K \leftarrow G_{\mathsf{salt}}(M); (x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M); \sigma \leftarrow (sk_{x_i})_{1 \leq i \leq k}; (K, \sigma));$$

- *verification $\mathcal{V}(pk, M, (K, \sigma))$ is $((x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M); \bigwedge_{i=1}^k F(\sigma_i) = pk_{x_i})$.*

Remark Contrary to the previous hash-based signature schemes, ORSS is not extractable (Definition 2). Indeed, given a single signature one can only recover up to k public values. We will overcome this limitation in the next section with the ORST scheme.

Game ORSS and

proc Initialize
  $F \xleftarrow{\$} \mathcal{F}_n$
proc GenKey(1^n)
  $\mathsf{salt} \xleftarrow{\$} B_n$
  $(s_1, \ldots, s_t) \xleftarrow{\$} B_n^t$
  for $i \in \{1, \ldots, t\}$ **do** $p_i \leftarrow F(s_i)$
  return $(p_1, \ldots, p_t)$
proc Sign(M)
  $K \leftarrow G_{\mathsf{salt}}(M)$
  $(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
  for $j \in \{1, \ldots, k\}$ **do** $y_j \leftarrow s_{x_j}$
  return $(K, y_1, \ldots, y_k)$
proc Finalize$(M, K, y_1, \ldots, y_k)$
  $(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
  $\mathsf{win} \leftarrow \bigwedge_{j=1}^{k} F(y_j) = p_{x_j}$
  return win

Adversary $\mathcal{C}$

$G \leftarrow$ **Initialize**
$F \xleftarrow{\$} \mathcal{F}_n$
Run $\mathcal{A}$
On query **GenKey**(1^n) of $\mathcal{A}$
  $(p_1, \ldots, p_t) \xleftarrow{\$}$ ORSS.**GenKey**(1^n)
  return $(p_1, \ldots, p_t)$ **to** $\mathcal{A}$
On query **Sign**(M) of $\mathcal{A}$
  $K \leftarrow G(M)$
  $(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
  for $j \in \{1, \ldots, k\}$ **do** $y_j \leftarrow s_{x_j}$
  return $(K, y_1, \ldots, y_k)$ **to** $\mathcal{A}$
On **Finalize**$(M, K, y_1, \ldots, y_k)$ of $\mathcal{A}$
  $(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
  output $\bigwedge_{j=1}^{k} F(y_j) = p_{x_j}$

Games G_4 and $\boxed{G_5}$

proc Initialize
  $F \xleftarrow{\$} \mathcal{F}_n$
proc GenKey(1^n)
  $G \xleftarrow{\$} (B_n \to B_n)$
  $(p_1, \ldots, p_t) \xleftarrow{\$}$ ORSS.**GenKey**(1^n)
  $\boxed{\alpha \xleftarrow{\$} \{1, \ldots, t\}}$
  return $(p_1, \ldots, p_t)$
proc Sign(M)
  $K \leftarrow G(M)$
  $(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
  $\boxed{\textbf{if } \alpha \in \{x_1, \ldots, x_k\} \textbf{ then abort}}$
  for $j \in \{1, \ldots, k\}$ **do** $y_j \leftarrow s_{x_j}$
  return $(K, y_1, \ldots, y_k)$
proc Finalize$(M, K, y_1, \ldots, y_k)$
  $(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
  $\boxed{\textbf{if } \alpha \notin \{x_1, \ldots, x_k\} \textbf{ then abort}}$
  $\mathsf{win} \leftarrow \bigwedge_{j=1}^{k} F(y_j) = p_{x_j}$
  return win

Adversary $\mathcal{D}$

$F, v \leftarrow$ **Initialize**
Run $\mathcal{A}$
On query **GenKey**(1^n) of $\mathcal{A}$
  $(p_1, \ldots, p_t) \leftarrow G_5$.**GenKey**$(1^n)$
  $p_\alpha \leftarrow v$
  return $(p_1, \ldots, p_t)$ **to** $\mathcal{A}$
On query **Sign**(M) of $\mathcal{A}$
  return G_5.**Sign**(M) **to** $\mathcal{A}$
On **Finalize**$(M, K, y_1, \ldots, y_k)$ of $\mathcal{A}$
  $(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
  if $\exists i, \alpha = x_i$ **then**
    $u \leftarrow y_i$
    if $F(u) = v$ **then output** u
  abort

Figure 6.4: Games ORSS, G_4 and G_5, adversary $\mathcal{C}$ against pseudo-randomness of $\mathcal{G}_n$ and adversary $\mathcal{D}$ against one-wayness of $\mathcal{F}_n$. Boxed statements are present only in game G_5. An adversary against games ORSS, G_4 and G_5 is forbidden to submit to **Finalize** a message M that it already queried to **Sign**.

Theorem 9. *Let $\mathcal{F}_n$ and $\mathcal{G}_n$ be function families, and $\mathcal{O}_n$ be a function family to obtain a random subset. We consider the following resources ξ: the time τ, the number of queries to the signature scheme q and the number of queries to $\mathcal{F}_n$, $\mathcal{G}_n$ and $\mathcal{O}_n$ respectively q_F, q_G and q_O. The unforgeability of ORSS based on $\mathcal{F}_n$, $\mathcal{G}_n$ and $\mathcal{O}_n$ can be bounded by the subset-resilience of $\mathcal{O}_n$, the one-wayness of $\mathcal{F}_n$ and the pseudo-randomness of $\mathcal{G}_n$:*

$$\begin{aligned}&\textsc{InSec}^{\text{EU-CMA}}(\text{ORSS}(\mathcal{F}_n, \mathcal{G}_n, \mathcal{O}_n); \tau, q = r, q_F, q_O)\\ &\leq \textsc{InSec}^{r-\text{NASR}}(\mathcal{O}_n; \tau', q'_O) + t\textsc{InSec}^{\text{OW}}(\mathcal{F}_n; \tau', q'_F) + \textsc{InSec}^{\text{PRF}}(\mathcal{G}_n; \tau', q'_G)\end{aligned}$$

where $q'_O = q_O$, $q'_F = q_F + t$, $q'_G = r$ and $\tau' = \tau + c \cdot (t + (r+1)k)$ for some constant c.

Essentially, assuming that $\mathcal{G}_n$ is pseudo-random, an attacker can forge a signature either by breaking the subset-resilience of $\mathcal{O}_n$ or by inverting at least one public value. Since there are t target values for one-wayness, security degrades by a factor at most t.

Proof. The proof is similar to the proof of security of WOTS (Theorem 7). Let $\mathcal{A}$ be an adversary for ORSS (Figure 6.4) that queries at most r messages to the **Sign** oracle.

We first consider adversary $\mathcal{C}$ against pseudo-randomness of $\mathcal{G}_n$ (Figure 6.4). $\mathcal{C}$ uses at most resources (τ', q'_G): it makes at most r queries to $\mathcal{G}_n$ in **Sign**; it spends additional time proportional to t in **GenKey**, to rk in **Sign** (called r times) and to k in **Finalize**.

If $\mathcal{C}$ is given as input a challenge G uniformly distributed in $\mathcal{G}_n$, it perfectly simulates game ORSS for $\mathcal{A}$. If $\mathcal{C}$ is given as input a challenge G uniformly distributed in $(B_n \to B_n)$, it perfectly simulates game G_4 for $\mathcal{A}$. Therefore the advantage of $\mathcal{C}$ against pseudo-randomness of $\mathcal{G}_n$ is the following:

$$\textsc{Succ}^{\text{PRF}}_{\mathcal{G}_n}(\mathcal{C}) = \left|\mathsf{Pr}[\mathcal{A}^{\text{ORSS}}_{\mathcal{F}_n,\mathcal{G}_n,\mathcal{O}_n} \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}^{G_4}_{\mathcal{F}_n,\mathcal{G}_n,\mathcal{O}_n} \Rightarrow 1]\right|$$

Consequently:

$$\mathsf{Pr}[\mathcal{A}^{\text{ORSS}}_{\mathcal{F}_n,\mathcal{G}_n,\mathcal{O}_n} \Rightarrow 1] \leq \textsc{InSec}^{\text{PRF}}(\mathcal{G}_n; \tau', q'_G) + \mathsf{Pr}[\mathcal{A}^{G_4}_{\mathcal{F}_n,\mathcal{G}_n,\mathcal{O}_n} \Rightarrow 1]$$

We now assume that $\mathcal{A}$ queries r messages $M_1, \ldots, M_r$ yielding keys $K_1, \ldots, K_r$ and successfully forges a signature for some $M' \notin \{M_1, \ldots, M_r\}$ with key K' in game G_4. There are two cases:

- either $\mathsf{ORS}_{K'}(M') \subseteq \mathsf{ORS}_{K_1}(M_1) \cup \ldots \cup \mathsf{ORS}_{K_r}(M_r)$, and we can construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the r-subset-resilience of $\mathcal{O}_n$, with resources (τ', q'_O);
- or there exists $\alpha \in \mathsf{ORS}_{K'}(M')$ such that $\alpha \notin \mathsf{ORS}_{K_1}(M_1) \cup \ldots \cup \mathsf{ORS}_{K_r}(M_r)$, and if game G_5 picks this α, which happens with probability $1/t$ and is independent of the choices of $\mathcal{A}$, then $\mathcal{A}$ wins G_5.

Likewise, if $\mathcal{A}$ directly forges a signature without querying any message, then $\mathcal{A}$ wins G_5 as long as G_5 picks some $\alpha \in \mathsf{ORS}_{K'}(M')$, which happens with probability at least $1/t$. It follows that:

$$\mathsf{Pr}[\mathcal{A}^{G_4}_{\mathcal{F}_n,\mathcal{G}_n,\mathcal{O}_n} \Rightarrow 1] \leq \textsc{InSec}^{r-\text{NASR}}(\mathcal{O}_n; \tau', q'_O) + t\mathsf{Pr}[\mathcal{A}^{G_5}_{\mathcal{F}_n,\mathcal{G}_n,\mathcal{O}_n} \Rightarrow 1]$$

Last, the success probability of $\mathcal{A}$ against game G_5 is bounded by $\text{InSec}^{\text{OW}}(\mathcal{F}_n; \tau', q'_F)$.

$$\mathsf{Pr}[\mathcal{A}^{G_5}_{\mathcal{F}_n,\mathcal{G}_n,\mathcal{O}_n} \Rightarrow 1] \leq \text{InSec}^{\text{OW}}(\mathcal{F}_n; \tau', q'_F)$$

Indeed, we can construct an adversary $\mathcal{D}$ that uses $\mathcal{A}$ against the one-wayness of $\mathcal{F}_n$, that uses at most resources (τ', q'_F) (Figure 6.4). When given as input a challenge $v = F(u)$ where u is uniformly distributed in B_n, $\mathcal{D}$ perfectly simulates game G_5 for $\mathcal{A}$. The bound follows. □

6.2.2 ORS with tree

The ORS with tree construction allows to reduce the public key to a single n-bit hash and to make the scheme extractable. For this purpose, we need to define Merkle trees.

Definition 20 (Merkle tree)**.** *A Merkle tree [Mer89] is a balanced binary hash tree, with a power-of-2 number of leaves. We denote by* Merkle-root*,* Merkle-auth *and* Merkle-extract *three useful deterministic functions on Merkle trees, as follows.*

Given the set of leaves $x_1, \ldots, x_{2^h}$, Merkle-root$(H, x_1, \ldots, x_{2^h})$ *returns the root hash of the Merkle tree.*

Given the set of leaves $x_1, \ldots, x_{2^h}$ and a subset of indices $\mathcal{I} \subseteq \{1, \ldots, 2^h\}$, the function Merkle-auth$(H, x_1, \ldots, x_{2^h}, \mathcal{I})$ *returns a sufficient sequence of authentication nodes.*

Given a subset of leaves $(x_i)_{i\in\mathcal{I}}$ along with their indices i, and a sufficient sequence of authentication nodes A, Merkle-extract$(H, (i, x_i)_{i\in\mathcal{I}}, A)$ *returns the root hash of the Merkle tree.*

For conciseness, we do not explicit Merkle-root, Merkle-auth and Merkle-extract here, but they are quite straightforward to define and implement [Mer89]. We assume that they verify the correctness property: for all $x_1, \ldots, x_{2^h}$, and all $\mathcal{I} \subseteq \{1, \ldots, 2^h\}$,

$$\mathsf{Merkle\text{-}extract}(H, (i, x_i)_{i\in\mathcal{I}}, \mathsf{Merkle\text{-}auth}(H, x_1, \ldots, x_{2^h}, \mathcal{I})) = \mathsf{Merkle\text{-}root}(H, x_1, \ldots, x_{2^h})$$

Remark Merkle-auth can be implemented with several strategies: full authentication paths, optimal octopus (Chapter 5), authentication paths with threshold as in HORST [BHH+15], etc. Which strategy is used is irrelevant for our security analysis, as long as enough authentication nodes are revealed.

Definition 21 (Mask-less ORST)**.** *Given a security parameter n, a power-of-2 set size t, a subset size $k \leq t$, a family $\mathcal{F}_n$ of functions $F : B_n \to B_n$, a family $\mathcal{G}_n$ of functions $G_{\mathsf{salt}} : B_n \to B_n$, a family $\mathcal{H}_n$ of hash functions $H : B_n^2 \to B_n$, and a family of functions to obtain random subsets $\mathcal{O}_n = \{\mathsf{ORS}_K : B_n \to \mathcal{P}_k(T) | K \in B_n\}$, the mask-less ORST is defined over $\mathcal{M} = B_n$, $\mathcal{PK} = B_n$, $\mathcal{SK} = B_n^t$, $\mathcal{SG} = B_n \times B_n^k \times B_n^{k \log_2 t}$ by the following algorithms:*

- *key generation $\mathcal{KG}(1^n)$ is $((pk, sk) \xleftarrow{\$} \text{ORSS}.\mathcal{KG}(1^n); (\mathsf{Merkle\text{-}root}(H, pk), sk))$;*
- *signing $\mathcal{S}((\mathsf{salt}, sk), M)$ is*

$$\begin{aligned}(K \leftarrow G_{\mathsf{salt}}(M); (x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M); \sigma \leftarrow (sk_{x_i})_{1\leq i\leq k};\\ A \leftarrow \mathsf{Merkle\text{-}auth}(H, pk_1, \ldots, pk_t, (x_1, \ldots, x_k)); (K, \sigma, A));\end{aligned}$$

Game ORST

proc Initialize
$F \xleftarrow{\$} \mathcal{F}_n$
$H \xleftarrow{\$} \mathcal{H}_n$
proc GenKey(1^n)
$(p_1, \ldots, p_t) \leftarrow \text{ORSS.}\mathbf{GenKey}(1^n)$
$p \leftarrow \mathsf{Merkle\text{-}root}(H, p_1, \ldots, p_t)$
return p
proc Sign(M)
$K \leftarrow G_{\mathsf{salt}}(M)$
$(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
for $j \in \{1, \ldots, k\}$ **do** $y_j \leftarrow s_{x_j}$
$A \leftarrow \mathsf{Merkle\text{-}auth}(H, p_1, \ldots, p_t, (x_j))$
return $(K, y_1, \ldots, y_k, A)$
proc Finalize$(M, K, y_1, \ldots, y_k, A)$
$(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
$y \leftarrow \mathsf{Merkle\text{-}extract}(H, (x_j, F(y_j)), A)$
return $y = p$

Adversary $\mathcal{B}$

$F, H \leftarrow$ **Initialize**
Run $\mathcal{A}$
On query **GenKey**(1^n) of $\mathcal{A}$
$(p_1, \ldots, p_t) \leftarrow \text{ORSS.}\mathbf{GenKey}(1^n)$
$p \leftarrow \mathsf{Merkle\text{-}root}(H, p_1, \ldots, p_t)$
return p **to** $\mathcal{A}$
On query **Sign**(M) of $\mathcal{A}$
$(K, y_1, \ldots, y_k) \leftarrow \text{ORSS.}\mathbf{Sign}(M)$
$(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
$A \leftarrow \mathsf{Merkle\text{-}auth}(H, p_1, \ldots, p_t, (x_j))$
return $(K, y_1, \ldots, y_k, A)$ **to** $\mathcal{A}$
On **Finalize**$(M, K, y_1, \ldots, y_k, A)$ of $\mathcal{A}$
$(x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M)$
if $\bigwedge_{j=1}^{k} F(y_j) = p_{x_j}$ **then**
output $(\mathsf{forgery}, M, y_1, \ldots, y_k)$
$y \leftarrow \mathsf{Merkle\text{-}extract}(H, (x_j, F(y_j)), A)$
if $y = p$ **then**
find collision in Merkle-tree
output collision
abort

Figure 6.5: Game ORST and adversary $\mathcal{B}$ against game ORSS and collision-resistance of $\mathcal{H}_n$. An adversary against game ORST is forbidden to submit to **Finalize** a message M that it already queried to **Sign**.

- *public key extraction* $\mathcal{E}(M, (K, \sigma, A))$ *is*

$$((x_1, \ldots, x_k) \leftarrow \mathsf{ORS}_K(M); \mathsf{Merkle\text{-}extract}(H, (x_i, F(\sigma_i))_{1 \le i \le k}, A)).$$

Theorem 10. *Let $\mathcal{F}_n$, $\mathcal{G}_n$ and $\mathcal{H}_n$ be function families, and $\mathcal{O}_n$ be a function family to obtain a random subset. We consider the following resources ξ: the time τ, the number of queries to the signature scheme q and the number of queries to $\mathcal{F}_n$, $\mathcal{H}_n$ and $\mathcal{O}_n$ respectively q_F, q_H and q_O. The unforgeability of ORST based on $\mathcal{F}_n$, $\mathcal{G}_n$, $\mathcal{H}_n$ and $\mathcal{O}_n$ can be bounded by the unforgeability of* $\text{ORSS}(\mathcal{F}_n, \mathcal{G}_n, \mathcal{O}_n)$ *and the collision resistance of $\mathcal{H}_n$:*

$$\begin{aligned}&\textsc{InSec}^{\text{EU-CMA}}(\text{ORST}(\mathcal{F}_n, \mathcal{G}_n, \mathcal{H}_n, \mathcal{O}_n); \tau, q = r, q_F, q_H, q_O)\\ &\le \textsc{InSec}^{\text{EU-CMA}}(\text{ORSS}(\mathcal{F}_n, \mathcal{G}_n, \mathcal{O}_n); \tau', q = r, q_F, q'_O) + \textsc{InSec}^{\text{CR}}(\mathcal{H}_n; \tau', q'_H)\end{aligned}$$

where $q'_H = q_H + (r+2)(t-1)$, $q'_O = q_O + r$, and $\tau' = \tau + c \cdot (r+2)t$ for some constant c.

Proof. The proof is similar to the security reduction of LWOTS (Theorem 8). Let $\mathcal{A}$ be an adversary against game ORST (Figure 6.5). We can construct an adversary $\mathcal{B}$ (Figure 6.5) that tries to win game ORSS or to find a collision for $\mathcal{F}_n$. Regarding resources, each call to

Merkle-root, Merkle-auth or Merkle-extract consists of at most $t-1$ calls to H, and $\mathcal{B}$ makes at most $r+2$ such calls.

If $\mathcal{A}$ wins the game, then the Merkle-tree root of the forgery matches the public key, and there are two cases:

- the Merkle tree leaves $(F(y_i))_{1\leq i\leq k}$ correspond to the ORSS public values $(p_{x_i})_{1\leq i\leq k}$, and adversary $\mathcal{B}$ wins game ORSS,
- there is a mismatch, i.e. for some $1 \leq j \leq k$, $F(y_j) \neq p_{x_j}$. In that case we consider the trees A_j and B_j that contain the authentication paths of $F(y_j)$ and p_{x_j} (respectively). By Lemma 10 a collision for H must exist between trees A_j and B_j, and adversary $\mathcal{B}$ successfully outputs it.

The result follows. □

6.3 Seed-based secret key

We now analyze the security of signature schemes that reduce the size of the secret key to a small seed, using a PRF to expand it.

Definition 22 (Pseudo-random transformation)**.** *Let N be an integer smaller than or equal to 2^n and $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ be a signature scheme such that $\mathcal{KG}$ contains statements to sample N values $s_0, \ldots, s_{N-1}$ uniformly at random from B_n, i.e. $s_0 \xleftarrow{\$} B_n; \ldots; s_{N-1} \xleftarrow{\$} B_n$. Given an integer $i < N$, we denote by $\lceil i \rfloor_n$ a n-bit encoding of i (e.g. big-endian, little-endian, etc.).*

Given a family $\mathcal{G}_n$ of functions $G_{\mathsf{seed}} : B_n \to B_n$, the pseudo-random transformation of $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ is a signature scheme $(\mathcal{KG}', \mathcal{S}, \mathcal{V})$ where key generation $\mathcal{KG}'$ contains the statement $\mathsf{seed} \xleftarrow{\$} B_n$ followed by algorithm $\mathcal{KG}$ where each statement $s_i \xleftarrow{\$} B_n$ is replaced by $s_i \leftarrow G_{\mathsf{seed}}(\lceil i \rfloor_n)$.

Theorem 11. *Let $\mathcal{G}_n$ be a function family and $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ be a signature scheme. We consider the following resources: the time τ, the number of queries to $\mathcal{G}_n$ q_G, and unspecified resources ξ specific to $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$. The unforgeability of the pseudo-random transformation of $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ under $\mathcal{G}_n$ can be bounded by the unforgeability of $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ and the pseudo-randomness of $\mathcal{G}_n$:*

$$\begin{aligned}&\textsc{InSec}^{\text{EU-CMA}}(\text{SEED}(\mathcal{G}_n, (\mathcal{KG}, \mathcal{S}, \mathcal{V})); \tau, \xi)\\ &\leq \textsc{InSec}^{\text{EU-CMA}}((\mathcal{KG}, \mathcal{S}, \mathcal{V}); \tau, \xi) + \textsc{InSec}^{\text{PRF}}(\mathcal{G}_n; \tau', q'_G)\end{aligned}$$

where $q'_G = N$ and $\tau' = \tau + c \cdot N$ for some constant c.

Proof. The original key generation $\mathcal{KG}$ is equivalent to sampling a function G uniformly at random from $(B_n \to B_n)$, and then computing $s_i \leftarrow G(\lceil i \rfloor_n)$. It follows that an adversary $\mathcal{A}$ for EU-CMA of $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ can be turned into a distinguisher $\mathcal{B}$ for pseudo-randomness of $\mathcal{G}_n$ that uses resources (τ', q'_G): given a function G sampled either from $(B_n \to B_n)$ or from $\mathcal{G}_n$, $\mathcal{B}$ simulates the signature scheme for $\mathcal{A}$ and outputs 1 if and only if $\mathcal{A}$ successfully forges a signature.

Depending on whether G was sampled from $(B_n \to B_n)$ or from $\mathcal{G}_n$, $\mathcal{B}$ simulates $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ or $\mathrm{SEED}(\mathcal{G}_n, (\mathcal{KG}, \mathcal{S}, \mathcal{V}))$ for $\mathcal{A}$. Consequently:

$$\left|\mathrm{Succ}^{\text{EU-CMA}}_{\mathrm{SEED}(\mathcal{G}_n,(\mathcal{KG},\mathcal{S},\mathcal{V}))}(\mathcal{A}) - \mathrm{Succ}^{\text{EU-CMA}}_{(\mathcal{KG},\mathcal{S},\mathcal{V})}(\mathcal{A})\right| = \mathrm{Succ}^{\text{PRF}}_{\mathcal{G}_n}(\mathcal{B}) \leq \mathrm{InSec}^{\text{PRF}}(\mathcal{G}_n; \tau', q'_G)$$

The result follows. □

Lazy evaluation In practice, a seed-transformed scheme can be lazily evaluated: values generated from the seed need not necessarily be computed at key generation time, nor stored in the secret key. Instead, they can be computed on demand in the signature algorithm, which is especially relevant when N is large.

6.4 Mask-less Merkle tree

The Merkle tree construction allows to transform a one-time signature scheme into a multiple-time signature scheme by creating a tree with 2^h instances of the one-time scheme. The new scheme is indexed (Definition 3).

Definition 23 (Merkle tree signature)**.** *Given a security parameter n, a height parameter h, an extractable signature scheme $(\mathcal{KG}, \mathcal{S}, \mathcal{E})$ defined over spaces $(\mathcal{M}, \mathcal{PK}, \mathcal{SK}, \mathcal{SG})$, and a family $\mathcal{H}_n$ of hash functions $H : B_n^2 \to B_n$, the Merkle tree indexed signature scheme is defined over:*

- *the message space $\mathcal{M}' = \mathcal{M}$, the public key space $\mathcal{PK}' = B_n$,*
- *the secret key space $\mathcal{SK}' = \mathcal{SK}^{2^h}$,*
- *the signature space $\mathcal{SG}' = \mathcal{SG} \times B_n^h$,*
- *the index space $\mathcal{I}' = \{1, \ldots, 2^h\}$,*

by the following algorithms:

- *key generation $\mathcal{KG}'(1^n)$ is*

$$((\textbf{\textit{for}}\ i \in \mathcal{I}'\ \textbf{\textit{do}}\ (sk_i, pk_i) \xleftarrow{\$} \mathcal{KG}(1^n)); pk \leftarrow \textsf{Merkle-root}(H, pk_1, \ldots, pk_{2^h}); (pk, sk));$$

- *indexed signing $\mathcal{S}'(sk, i, M)$ is*

$$(\sigma \leftarrow \mathcal{S}(sk_i, M); A \leftarrow \textsf{Merkle-auth}(H, pk_1, \ldots, pk_{2^h}, i); (\sigma, A))$$

- *public key extraction $\mathcal{E}'(i, M, (\sigma, A))$ is $(e \leftarrow \mathcal{E}(M, \sigma); \textsf{Merkle-extract}(H, (i, e), A))$.*

Theorem 12. *Let $\mathcal{H}_n$ be a hash function family and $(\mathcal{KG}, \mathcal{S}, \mathcal{E})$ be an extractable signature scheme. We consider the following resources: the time τ, the number of signature queries q, the maximal number of signature queries per index q_i, the number of queries to $\mathcal{H}_n$ q_H, and unspecified resources ξ specific to $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$. The unforgeability of the Merkle tree indexed signature scheme based on $\mathcal{H}_n$ and $(\mathcal{KG}, \mathcal{S}, \mathcal{E})$ can be bounded by the unforgeability of $(\mathcal{KG}, \mathcal{S}, \mathcal{E})$ and the collision resistance of $\mathcal{H}_n$:*

$$\begin{aligned}&\mathrm{InSec}^{\text{EU-CMA}}(\mathrm{MT}(\mathcal{H}_n, (\mathcal{KG}, \mathcal{S}, \mathcal{E})); \tau, q, q_i, q_H, \xi)\\ &\leq 2^h \mathrm{InSec}^{\text{EU-CMA}}((\mathcal{KG}, \mathcal{S}, \mathcal{E}); \tau', q', \xi) + \mathrm{InSec}^{\text{CR}}(\mathcal{H}_n; \tau', q'_H)\end{aligned}$$

where $q' = q_i$, $q'_H = q_H + 2^h$ and $\tau' = \tau + c \cdot qh$ for some constant c.

Essentially, the security degrades with a factor 2^h because an adversary can try to break any of the 2^h instances of the underlying signature scheme.

Proof. We follow the proof by Dahmen et al. [DOTV08]. Given an adversary $\mathcal{A}$ against the Merkle signature scheme, a successful forgery happens in two cases:

- the forged signature σ at index i matches with pk_i, and we can construct an adversary $\mathcal{B}$ against the underlying scheme $(\mathcal{KG}, \mathcal{S}, \mathcal{E})$; more precisely $\mathcal{B}$ guesses i and places the challenge at leaf i, the guess being correct with probability 2^{-h},
- the signature does not match and by Lemma 10 we can obtain a hash collision in the Merkle tree.

The bound follows. □

6.5 Product composition and hyper-trees

The product composition was analyzed by Malkin et al. [MMM02]. It allows to compose indexed signature schemes to multiply their index spaces, with a limited overhead for the signer. This composition is the basis of hyper-trees as in XMSS and SPHINCS.

Definition 24 (Product composition)**.** *Given indexed signature schemes* $(\mathcal{KG}_1, \mathcal{S}_1, \mathcal{V}_1)$ *and* $(\mathcal{KG}_2, \mathcal{S}_2, \mathcal{E}_2)$ *defined over spaces* $(\mathcal{M}_1, \mathcal{PK}_1, \mathcal{SK}_1, \mathcal{SG}_1, \mathcal{I}_1)$ *and* $(\mathcal{M}_2, \mathcal{PK}_2 = \mathcal{M}_1, \mathcal{SK}_2, \mathcal{SG}_2, \mathcal{I}_2)$*, their product composition is an indexed signature scheme defined over:*

- *the message space* $\mathcal{M} = \mathcal{M}_2$*,*
- *the public key space* $\mathcal{PK} = \mathcal{PK}_1$*,*
- *the secret key space* $\mathcal{SK} = \mathcal{SK}_1 \times (\mathcal{PK}_2 \times \mathcal{SK}_2)^{\mathcal{I}_1}$*,*
- *the signature space* $\mathcal{SG} = \mathcal{SG}_1 \times \mathcal{SG}_2$*,*
- *the index space* $\mathcal{I} = \mathcal{I}_1 \times \mathcal{I}_2$*,*

by the following algorithms:

- *key generation* $\mathcal{KG}(1^n)$ *is*

$$((pk, sk) \xleftarrow{\$} \mathcal{KG}_1(1^n); (\textbf{\textit{for}}\ i \in \mathcal{I}_1\ \textbf{\textit{do}}\ (sk'_i, pk'_i) \xleftarrow{\$} \mathcal{KG}_2(1^n)); (pk, (sk, (pk'_i, sk'_i)_{i\in\mathcal{I}_1})));$$

- *indexed signing* $\mathcal{S}((sk, (pk'_i, sk'_i)_{i\in\mathcal{I}_1}), (i_1, i_2), M)$ *is*

$$(\sigma_1 = \mathcal{S}_1(sk, i_1, pk'_{i_1}); \sigma_2 = \mathcal{S}_2(sk'_{i_1}, i_2, M); (\sigma_1, \sigma_2));$$

- *public key extraction (if applicable)* $\mathcal{E}((i_1, i_2), M, (\sigma_1, \sigma_2))$ *is*

$$(e_2 \leftarrow \mathcal{E}_2(i_2, M, \sigma_2); \mathcal{E}_1(i_1, e_2, \sigma_1));$$

- *verification* $\mathcal{V}(pk, (i_1, i_2), M, (\sigma_1, \sigma_2))$ *is* $(e_2 \leftarrow \mathcal{E}_2(i_2, M, \sigma_2); \mathcal{V}_1(pk, i_1, e_2, \sigma_1))$*.*

Remark Thanks to the seed transformation (Section 6.3), the $|\mathcal{I}_1|$ key pairs of scheme 2 can be computed on demand by the signing algorithm. This makes the scheme more practical than a basic scheme over $\mathcal{I}$ (e.g. a Merkle tree of height $\log_2 |\mathcal{I}|$), because key generation is $|\mathcal{I}_1|$ times faster and the secret key is reduced to a small seed. We note that the seed transformation works as long as all seed-based secret values in the construction use distinct indices. For example, one can use the addressing scheme of SPHINCS [BHH+15].

Theorem 13. *Let $(\mathcal{KG}_1, \mathcal{S}_1, \mathcal{V}_1)$ and $(\mathcal{KG}_2, \mathcal{S}_2, \mathcal{V}_2)$ be indexed signature schemes. We consider the following resources: the time τ, the number of signature queries q, the maximal number of signature queries per index q_i, and unspecified resources ξ specific to $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$. The unforgeability of their product composition $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ can be bounded by the unforgeability of $(\mathcal{KG}_1, \mathcal{S}_1, \mathcal{V}_1)$ and $(\mathcal{KG}_2, \mathcal{S}_2, \mathcal{V}_2)$:*

$$\begin{aligned}&\textsc{InSec}^{\text{EU-CMA}}((\mathcal{KG}, \mathcal{S}, \mathcal{V}); \tau, q, q_i, \xi)\\ &\leq \textsc{InSec}^{\text{EU-CMA}}((\mathcal{KG}_1, \mathcal{S}_1, \mathcal{V}_1); \tau', q, q_i', \xi) + |\mathcal{I}_1| \textsc{InSec}^{\text{EU-CMA}}((\mathcal{KG}_2, \mathcal{S}_2, \mathcal{V}_2); \tau', q, q_i, \xi)\end{aligned}$$

where $q_i' = 1$ and $\tau' = \tau + c \cdot q$ for some constant c.

Proof. We follow the proof by Malkin et al. [MMM02]. Given an adversary $\mathcal{A}$ against the product composition, a successful forgery happens in two cases:

- the extracted public key $\mathcal{E}_2(i_2, M, \sigma_2)$ matches with pk'_{i_1}, and we can construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ against scheme 2; more precisely, given a challenge for the scheme 2, $\mathcal{B}$ guesses an index $i_1 \in \mathcal{I}_1$ and places the challenge at this index; the index is correct w.r.t. the forgery with probability $1/|\mathcal{I}_1|$,
- the extracted public key $\mathcal{E}_2(i_2, M, \sigma_2)$ does not match, and we can construct an adversary $\mathcal{C}$ that uses $\mathcal{A}$ against scheme 1; more precisely, $\mathcal{C}$ simulates the product scheme and replaces the instance of scheme 1 by its challenge.

The bound follows. □

6.6 Batch signing

We now analyze the batch signing scheme that uses a Merkle tree to gather several messages, the root of which is signed by another signature scheme.

Definition 25 (Batch Merkle tree signature)**.** *Given a security parameter n, a height parameter h, a signature scheme $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ defined over spaces $(\mathcal{M} = B_n, \mathcal{PK}, \mathcal{SK}, \mathcal{SG})$, and a family $\mathcal{H}_n$ of hash functions $H : B_n^2 \to B_n$, the batch Merkle tree signature scheme is defined over:*

- *the message space $\mathcal{M} = B_n$, the key spaces $\mathcal{PK}' = \mathcal{PK}$ and $\mathcal{SK}' = \mathcal{SK}$,*
- *the signature space $\mathcal{SG}' = \mathcal{SG} \times \{1, \ldots, 2^h\} \times B_n^h$,*
- *the batch size $N = 2^h$,*

by the following algorithms:

- *key generation* $\mathcal{KG}' = \mathcal{KG}$*;*
- *multi-message signing* $\mathcal{S}'(sk, (M_1, \ldots, M_{2^h}))$ *is*

$$(R \leftarrow \mathsf{Merkle\text{-}root}(H, M_1, \ldots, M_{2^h}); \sigma \leftarrow \mathcal{S}(sk, R);$$
$$(\sigma, i, \mathsf{Merkle\text{-}auth}(H, M_1, \ldots, M_{2^h}, i))_{1 \le i \le 2^h});$$

- *public key extraction (if applicable)* $\mathcal{E}'(M, (\sigma, i, A))$ *is*

$$(R \leftarrow \mathsf{Merkle\text{-}extract}(H, (i, M), A); \mathcal{E}(R, \sigma));$$

- *verification* $\mathcal{V}'(pk, M, (\sigma, i, A))$ *is* $(R \leftarrow \mathsf{Merkle\text{-}extract}(H, (i, M), A); \mathcal{V}(pk, R, \sigma))$.

Remark Although the above definition requires a fixed number of messages 2^h, one can in practice sign a variable number of messages, e.g. by using a L-tree instead of a Merkle tree.

Theorem 14. *Let* $\mathcal{H}_n$ *be a hash function family and* $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ *be a signature scheme. We consider the following resources: the time* τ*, the number of messages queried to the signature scheme* q*, the number of batch queries* q_b*, the number of queries to* $\mathcal{H}_n$ q_H*, and unspecified resources* ξ *specific to* $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$*. The unforgeability of the batch Merkle tree signature scheme based on* $\mathcal{H}_n$ *and* $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ *can be bounded by the unforgeability of* $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ *and the collision resistance of* $\mathcal{H}_n$*:*

$$\begin{aligned}&\textsc{InSec}^{\text{EU-CMA}}(\text{BATCH}(\mathcal{H}_n, (\mathcal{KG}, \mathcal{S}, \mathcal{V})); \tau, q, q_b, q_H, \xi)\\ &\le \textsc{InSec}^{\text{EU-CMA}}((\mathcal{KG}, \mathcal{S}, \mathcal{V}); \tau', q', \xi) + \textsc{InSec}^{\text{CR}}(\mathcal{H}_n; \tau', q'_H)\end{aligned}$$

where $q' = q_b$*,* $q'_H = q_H + q - q_b$ *and* $\tau' = \tau + c \cdot q$ *for some constant* c*.*

Essentially, a forger either breaks the underlying signature scheme or finds a collision in one of the signed Merkle trees.

Proof. Let adversary $\mathcal{A}$ be a successful forger, that outputs a signature (σ, i, A) for a message M_i. If we let $R = \mathsf{Merkle\text{-}extract}(H, (i, M_i), A)$, there are two cases:

- R was not queried to the underlying signature oracle $\mathcal{S}$, and we can construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to forge a signature for $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$,
- R was queried to $\mathcal{S}$ with some Merkle tree, and in particular a message $N_i \neq M_i$ was queried at index i with some authentication path B. Then by Lemma 10 there exists a collision for H between the trees (M_i, A) and (N_i, B) and we can construct an adversary $\mathcal{C}$ that uses $\mathcal{A}$ to find this collision.

Regarding resources, adversary $\mathcal{B}$ requests q_b signatures to $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$, and both $\mathcal{B}$ and $\mathcal{C}$ make $q - q_b$ calls to H to generate the Merkle trees, with an additional time proportional to the number of messages. The bound follows. □

Chapter 7

Gravity: a family of stateless hash-based signature schemes

In this section, we present Gravity, a family of stateless hash-based signature schemes which benefits from our improvements described in Chapter 3. We first describe algorithms for key generation, signing and verification, then propose practical sets of parameters. Last, we explain strategies to optimize the implementation of Gravity.

7.1 Specification

7.1.1 Parameters

An instance of the Gravity signature scheme requires the following parameters:

- the **hash output bit length** n, a positive integer,
- the **Winternitz depth** w, a power of two such that $w \geq 2$ and $\log_2 w$ divides n,
- the **PORS set size** t, a positive power of two,
- the **PORS subset size** k, a positive integer such that $k \leq t$,
- the **internal Merkle tree height** h, a positive integer,
- the **number of internal Merkle trees** d, a non-negative integer,
- the **cache height** c, a non-negative integer,
- the **batching height** b, a non-negative integer,
- the **message space** $\mathcal{M}$, usually a subset of bit strings $\{0,1\}^*$.

From these parameters are derived:

- the **Winternitz width** $\ell = \mu + \lfloor \log_2 (\mu(w-1)) / \log_2 w \rfloor + 1$ where $\mu = n / \log_2 w$,
- the **PORS set** $T = \{0, \ldots, t-1\}$,

- the **address space** $\mathcal{A} = \{0, \ldots, d\} \times \{0, \ldots, 2^{c+dh} - 1\} \times \{0, \ldots, \max(\ell, t) - 1\}$,
- the **public key space** $\mathcal{PK} = B_n$,
- the **secret key space** $\mathcal{SK} = B_n^2$,
- the **signature space** $\mathcal{SG} = B_n \times B_n^k \times B_n^{\leq k(\log_2 t - \lfloor \log_2 k \rfloor)} \times \left(B_n^\ell \times B_n^h\right)^d \times B_n^c$,
- the **batched signature space** $\mathcal{SG}_B = B_n^b \times \{0, \ldots, 2^b - 1\} \times \mathcal{SG}$,
- the **public key size**, of n bits,
- the **secret key size**, of $2n$ bits,
- the **maximal signature size**, of

$$\mathsf{sigsz} = (1 + k + k(\log_2 t - \lfloor \log_2 k \rfloor) + d(\ell + h) + c)n \text{ bits}$$

- the **maximal batched signature size**, of $\mathsf{sigsz} + bn + b$ bits.

We recall that $B_n = \{0, 1\}^n$ denotes the set of n-bit strings.

7.1.2 Primitives

An instance of the Gravity signature scheme is based on four primitives, that depend on the parameters n and $\mathcal{M}$:

- a length-preserving hash function $F : B_n \to B_n$,
- a length-halving hash function $H : B_n^2 \to B_n$,
- a pseudo-random function $G : B_n \times \mathcal{A} \to B_n$ (that takes as input a seed and address),
- a general-purpose hash function $H^* : \mathcal{M} \to B_n$.

For example, with $n = 256$, one can take: 6-round Haraka-v2-256 as F, 6-round Haraka-v2-512 as H, a variant of AES-256 as G, and SHA-256 as H^*.

7.1.3 Useful algorithms

We first define useful algorithms that are building blocks for the Gravity scheme.

Operations on addresses

We now detail the addressing scheme in the hyper-tree structure. Each WOTS and PORST instance is given a unique address that allows to generate its secret values on demand. Each address contains:

- a layer $0 \leq i \leq d$ in the hyper-tree, where 0 is the root layer, $d - 1$ is the last WOTS layer and d is the PORST layer;
- an instance index j in the layer, with $0 \leq j < 2^{c+(i+1)h}$ if $i < d$ and $0 \leq j < 2^{c+dh}$ if $i = d$;

- a counter λ in the instance, with $0 \leq \lambda < \ell$ if $i < d$ and $0 \leq \lambda < t$ if $i = d$.

We define the following functions to manipulate addresses.

- The function $\mathsf{make\text{-}addr} : \{0, \ldots, d\} \times \mathbb{N} \to \mathcal{A}$ takes as input a layer $i \in \{0, \ldots, d\}$ and an index $j \in \mathbb{N}$ and returns the address $a = (i, j \mod 2^{c+dh}, 0) \in \mathcal{A}$.

- The function $\mathsf{incr\text{-}addr} : \mathcal{A} \times \mathbb{N} \to \mathcal{A}$ takes as input an address $a = (i, j, \lambda)$ and an integer x and returns the address $a' = (i, j, \lambda + x) \in \mathcal{A}$ with the counter incremented by x.

L-tree

We recall the function $\mathsf{L\text{-}tree} : B_n^+ \to B_n$ that takes as input a sequence of hashes $x_i \in B_n$ and returns the associated L-tree root $r \in B_n$ (Definition 15). It is defined by recurrence as follows.

$$\begin{cases} \mathsf{L\text{-}tree}(x_1) = x_1 \\ \mathsf{L\text{-}tree}(x_1, \ldots, x_{2i+2}) = \mathsf{L\text{-}tree}(H(x_1, x_2), \ldots, H(x_{2i+1}, x_{2i+2})) \\ \mathsf{L\text{-}tree}(x_1, \ldots, x_{2i+3}) = \mathsf{L\text{-}tree}(H(x_1, x_2), \ldots, H(x_{2i+1}, x_{2i+2}), x_{2i+3}) \end{cases}$$

Winternitz checksum

We recall the function $\mathsf{checksummed} : B_n \to \{0, \ldots, w-1\}^\ell$ that takes as input a hash $x \in B_n$ and returns ℓ integers x_i, computed as follows (Definition 13).

- For $i \in \{1, \ldots, \mu\}$ compute $z_i \leftarrow \mathsf{substr}(x, (i-1)\log_2 w, \log_2 w)$, where $\mathsf{substr}(x, j, m)$ denotes the substring of x of length m bits starting at bit index $0 \leq j < |x|$.

- For $i \in \{1, \ldots, \mu\}$ interpret z_i as the big-endian encoding of a number $0 \leq x_i < w$.

- Compute the checksum $C = \sum_{i=1}^{\mu} w - 1 - x_i$.

- For $i \in \{\mu+1, \ldots, \ell\}$ compute $x_i = \lfloor C / w^{i-\mu-1} \rfloor \mod w$. In other words, $(x_{\mu+1}, \ldots, x_\ell)$ is the base-w little-endian encoding of the checksum C.

Winternitz public key generation

The function $\mathsf{WOTS\text{-}genpk} : B_n \times \mathcal{A} \to B_n$ takes as input a secret $\mathsf{seed} \in B_n$ and a base address $a \in \mathcal{A}$, and outputs the associated Winternitz public key $p \in B_n$, computed as follows.

- For $i \in \{1, \ldots, \ell\}$ compute the secret value $s_i \leftarrow G(\mathsf{seed}, \mathsf{incr\text{-}addr}(a, i-1))$.

- For $i \in \{1, \ldots, \ell\}$ compute the public value $p_i \leftarrow F^{w-1}(s_i)$ where the F^{w-1} denotes the function F iterated $w-1$ times.

- Compute $p \leftarrow \mathsf{L\text{-}tree}(p_1, \ldots, p_\ell)$.

Winternitz signature

The function $\mathsf{WOTS\text{-}sign} : B_n \times \mathcal{A} \times B_n \to B_n^\ell$ takes as input a secret $\mathsf{seed} \in B_n$, a base address $a \in \mathcal{A}$ and a hash $x \in B_n$, and outputs the associated Winternitz signature $\sigma \in B_n^\ell$, computed as follows.

- For $i \in \{1, \ldots, \ell\}$ compute the secret value $s_i \leftarrow G(\mathsf{seed}, \mathsf{incr\text{-}addr}(a, i-1))$.
- Compute $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(x)$.
- For $i \in \{1, \ldots, \ell\}$ compute the signature value $\sigma_i \leftarrow F^{x_i}(s_i)$.

Winternitz public key extraction

The function $\mathsf{WOTS\text{-}extractpk} : B_n \times B_n^\ell \to B_n$ takes as input a hash $x \in B_n$ and a signature $\sigma \in B_n^\ell$, and outputs the associated Winternitz public key $p \in B_n$, computed as follows.

- Compute $(x_1, \ldots, x_\ell) \leftarrow \mathsf{checksummed}(x)$.
- For $i \in \{1, \ldots, \ell\}$ compute the public value $p_i \leftarrow F^{w-1-x_i}(\sigma_i)$.
- Compute $p \leftarrow \mathsf{L\text{-}tree}(p_1, \ldots, p_\ell)$.

Merkle tree root

The function $\mathsf{Merkle\text{-}root}_h : B_n^{2^h} \to B_n$ takes as input 2^h leaf hashes x_i, and outputs the associated Merkle tree root $r \in B_n$. It is defined by recurrence on h as:

- $\mathsf{Merkle\text{-}root}_0(x_0) = x_0$,
- $\mathsf{Merkle\text{-}root}_{h+1}(x_0, x_1, \ldots, x_{2i}, x_{2i+1}) = \mathsf{Merkle\text{-}root}_h(H(x_0, x_1), \ldots, H(x_{2i}, x_{2i+1}))$.

Merkle tree authentication

The function $\mathsf{Merkle\text{-}auth}_h : B_n^{2^h} \times \{0, \ldots, 2^h - 1\} \to B_n^h$ takes as input 2^h leaf hashes x_i and a leaf index $0 \leq j < 2^h$, and outputs the associated Merkle tree authentication path $(a_1, \ldots, a_h) \in B_n^h$. It is defined by recurrence on h as:

- $\mathsf{Merkle\text{-}auth}_1(x_0, x_1, j) = a_1 \leftarrow x_{j \oplus 1}$ where $\oplus$ denotes the bitwise XOR operation on non-negative integers,
- $\mathsf{Merkle\text{-}auth}_{h+1}(x_0, x_1, \ldots, x_{2i}, x_{2i+1}, j)$ is

$$\begin{cases} a_1 \leftarrow x_{j \oplus 1} \\ a_2, \ldots, a_{h+1} \leftarrow \mathsf{Merkle\text{-}auth}_h(H(x_0, x_1), \ldots, H(x_{2i}, x_{2i+1}), \lfloor j/2 \rfloor) \end{cases}$$

Merkle tree root extraction

The function $\mathsf{Merkle\text{-}extract}_h : B_n \times \{0, \ldots, 2^h - 1\} \times B_n^h \to B_n$ takes as input a leaf hash $x \in B_n$, a leaf index $0 \leq j < 2^h$ and an authentication path $(a_1, \ldots, a_h) \in B_n^h$, and outputs the associated Merkle tree root $r \in B_n$. It is defined by recurrence on h as:

- $\mathsf{Merkle\text{-}extract}_0(x, j) = x$,
- $\mathsf{Merkle\text{-}extract}_{h+1}(x, j, a_1, \ldots, a_{h+1}) = \mathsf{Merkle\text{-}extract}_h(x', \lfloor j/2 \rfloor, a_2, \ldots, a_{h+1})$ where

$$x' = \begin{cases} H(x, a_1) & \text{if } j \mod 2 = 0 \\ H(a_1, x) & \text{if } j \mod 2 = 1 \end{cases}$$

Octopus authentication

The function $\mathsf{Octopus\text{-}auth}_h : B_n^{2^h} \times \{0, \ldots, 2^h - 1\}^k \to B_n^* \times B_n$ takes as input 2^h leaf hashes x_i and $1 \leq k \leq 2^h$ distinct leaf indices $0 \leq j_i < 2^h$ sorted in increasing order, and outputs the associated octopus authentication nodes $oct \in B_n^*$ and the octopus root $r \in B_n$. It is defined by recurrence on h as:

- $\mathsf{Octopus\text{-}auth}_0(x_0, j_1) = (\emptyset, x_0)$,
- $\mathsf{Octopus\text{-}auth}_{h+1}(x_0, x_1, \ldots, x_{2i}, x_{2i+1}, j_1, \ldots, j_k)$ is computed as

$$\begin{cases} j'_1, \ldots, j'_\kappa \leftarrow \mathsf{unique}(\lfloor j_1/2 \rfloor, \ldots, \lfloor j_k/2 \rfloor) \\ oct', r \leftarrow \mathsf{Octopus\text{-}auth}_h(H(x_0, x_1), \ldots, H(x_{2i}, x_{2i+1}), j'_1, \ldots, j'_\kappa) \\ z_1, \ldots, z_{2\kappa-k} \leftarrow (j_1 \oplus 1, \ldots, j_k \oplus 1) \setminus (j_1, \ldots, j_k) \\ a_1, \ldots, a_{2\kappa-k} \leftarrow (x_{z_1}, \ldots, x_{z_{2\kappa-k}}) \\ oct \leftarrow (a_1, \ldots, a_{2\kappa-k}, oct') \end{cases}$$

  where $\mathsf{unique}()$ removes duplicates in a sequence, and $A \setminus B$ denotes the set difference.

Octopus root extraction

The function $\mathsf{Octopus\text{-}extract}_{h,k} : B_n^k \times \{0, \ldots, 2^h - 1\}^k \times B_n^* \to B_n \cup \{\bot\}$ (with $1 \leq k \leq 2^h$) takes as input k leaf hashes $x_i \in B_n$, k leaf indices $0 \leq j_i < 2^h$ and an authentication octopus $oct \in B_n^*$, and outputs the associated Merkle tree root $r \in B_n$, or $\bot$ if the number of hashes in the authentication octopus is invalid. It is defined by recurrence on h as:

- $\mathsf{Octopus\text{-}extract}_{0,1}(x_1, j_1, oct) = \begin{cases} x_1 & \text{if } oct = \emptyset \\ \bot & \text{otherwise} \end{cases}$,
- $\mathsf{Octopus\text{-}extract}_{h+1,k}(x_1, \ldots, x_k, j_1, \ldots, j_k, oct)$ is computed as

$$\begin{cases} j'_1, \ldots, j'_\kappa \leftarrow \mathsf{unique}(\lfloor j_1/2 \rfloor, \ldots, \lfloor j_k/2 \rfloor) & \\ L \leftarrow \mathsf{Octopus\text{-}layer}((x_1, j_1), \ldots, (x_k, j_k), oct) & \\ \bot & \text{if } L = \bot \\ \mathsf{Octopus\text{-}extract}_{h,\kappa}(x'_1, \ldots, x'_\kappa, j'_1, \ldots, j'_\kappa, oct') & \text{if } L = (x'_1, \ldots, x'_\kappa, oct') \end{cases}$$

where Octopus-layer() is defined by recurrence as:

- $\mathsf{Octopus\text{-}layer}(x_1, j_1, oct) = \begin{cases} \bot & \text{if } oct = \emptyset \\ H(x_1, a), oct' & \text{if } oct = (a, oct') \wedge j_1 \mod 2 = 0 \\ H(a, x_1), oct' & \text{if } oct = (a, oct') \wedge j_1 \mod 2 = 1 \end{cases}$
- $\mathsf{Octopus\text{-}layer}(x_1, j_1, x_2, j_2, \ldots, x_k, j_k, oct)$ is

$$\begin{cases} H(x_1, x_2), \mathsf{Octopus\text{-}layer}(x_3, j_3, \ldots, x_k, j_k, oct) & \text{if } j_1 \oplus 1 = j_2 \\ \bot & \text{if } j_1 \oplus 1 \neq j_2 \wedge oct = \emptyset \\ H(x_1, a), \mathsf{Octopus\text{-}layer}(x_2, j_2, \ldots, x_k, j_k, oct') & \text{if } oct = (a, oct') \wedge j_1 \mod 2 = 0 \\ H(a, x_1), \mathsf{Octopus\text{-}layer}(x_2, j_2, \ldots, x_k, j_k, oct') & \text{if } oct = (a, oct') \wedge j_1 \mod 2 = 1 \end{cases}$$

PRNG to obtain a random subset

The function $\mathsf{PORS} : B_n \times B_n \to \mathbb{N} \times T^k$ takes as input a salt $s \in B_n$ and a hash $x \in B_n$, and outputs a hyper-tree index $\lambda \in \mathbb{N}$ and k distinct indices x_i, computed as follows.

- Compute $g \leftarrow H(s, x)$.
- Let $a \leftarrow \mathsf{make\text{-}addr}(0, 0)$.
- Compute $b \leftarrow G(g, a)$ and interpret it as the big-endian encoding of an integer $\beta \in \{0, \ldots, 2^n - 1\}$.
- Compute $\lambda \leftarrow \beta \mod 2^{c+dh}$. In other words, λ is the big-endian interpretation of the $c + dh$ last bits of the block b.
- Initialize $X \leftarrow \emptyset$ and $j \leftarrow 0$.
- While $|X| < k$ do the following:
  - increment $j \leftarrow j + 1$,
  - compute $b \leftarrow G(g, \mathsf{incr\text{-}addr}(a, j))$,
  - split b into $\nu = \lfloor n / \log_2 t \rfloor$ blocks of $\log_2 t$ bits, as $b_i = \mathsf{substr}(b, (i-1)\log_2 t, \log_2 t)$,
  - for $i \in \{1, \ldots, \nu\}$ interpret b_i as the big-endian encoding of an integer $\overline{b_i} \in T$,
  - for $i \in \{1, \ldots, \nu\}$, if $|X| < k$ compute $X \leftarrow \mathsf{unique}(X, \overline{b_i})$.
- Compute $(x_1, \ldots, x_k) \leftarrow \mathsf{sorted}(X)$.

PORST signature

The function $\mathsf{PORST\text{-}sign} : B_n \times \mathcal{A} \times T^k \to B_n^k \times B_n^* \times B_n$ takes as input a secret $\mathsf{seed} \in B_n$, a base address $a \in \mathcal{A}$ and k sorted indices $x_i \in T$, and outputs the associated PORST signature $(\sigma, oct) \in B_n^k \times B_n^*$ and PORST public key $p \in B_n$, computed as follows.

- For $i \in \{1, \ldots, t\}$ compute the secret value $s_i \leftarrow G(\mathsf{seed}, \mathsf{incr\text{-}addr}(a, i-1))$.
- For $j \in \{1, \ldots, k\}$ set the signature value $\sigma_j = s_{x_j}$.
- Compute the authentication octopus and root as

$$oct, p \leftarrow \mathsf{Octopus\text{-}auth}_{\log_2 t}(s_1, \ldots, s_t, x_1, \ldots, x_k)$$

PORST public key extraction

The function $\mathsf{PORST\text{-}extractpk} : T^k \times B_n^k \times B_n^* \to B_n \cup \{\perp\}$ takes as input k indices $x_i \in T$ and a PORST signature $(\sigma, oct) \in B_n^k \times B_n^*$, and outputs the associated PORST public key $p \in B_n$, or $\perp$ if the authentication octopus is invalid, computed as follows.

- Compute the octopus root $p \leftarrow \mathsf{Octopus\text{-}extract}_{\log_2 t,k}(\sigma, x_1, \ldots, x_k, oct)$.

7.1.4 Signature scheme

We now specify the $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ algorithms for GRAVITY, as well as batched variants $(\mathcal{KG}, \mathcal{S}_B, \mathcal{V}_B)$. To simplify, we specify them without secret key caching by the signer. Indeed, this caching optimization is internal to the signer – to increase signing speed – and does not change the public results (public key, signature). We discuss this optimization in Section 7.3.

Key generation

$\mathcal{KG}$ takes as input $2n$ bits of randomness and outputs the secret key $sk \in B_n^2$ and the public key $pk \in B_n$.

- Generate the secret key from $2n$ bits of randomness $sk = (\mathsf{seed}, \mathsf{salt}) \xleftarrow{\$} B_n^2$.
- For $0 \leq i < 2^{c+h}$ generate a Winternitz public key

$$p_i \leftarrow \mathsf{WOTS\text{-}genpk}(\mathsf{seed}, \mathsf{make\text{-}addr}(0, i))$$

- Generate the public key $pk \leftarrow \mathsf{Merkle\text{-}root}_{c+h}(x_0, \ldots, x_{2^{c+h}-1})$.

Signature

$\mathcal{S}$ takes as input a hash $m \in B_n$ and a secret key $sk = (\mathsf{seed}, \mathsf{salt})$, and outputs a signature computed as follows.

- Compute the public salt $s \leftarrow H(\mathsf{salt}, m)$.
- Compute the hyper-tree index and random subset as $j, (x_1, \ldots, x_k) \leftarrow \mathsf{PORS}(s, m)$.
- Compute the PORST signature and public key

$$(\sigma_d, oct, p) \leftarrow \mathsf{PORST\text{-}sign}(\mathsf{seed}, \mathsf{make\text{-}addr}(d, j), x_1, \ldots, x_k)$$

- For $i \in \{d-1, \ldots, 0\}$ do the following:
  - compute the WOTS signature $\sigma_i \leftarrow \mathsf{WOTS\text{-}sign}(\mathsf{seed}, \mathsf{make\text{-}addr}(i, j), p)$,
  - compute $p \leftarrow \mathsf{WOTS\text{-}extractpk}(p, \sigma_i)$,
  - set $j' \leftarrow \lfloor j/2^h \rfloor$,
  - for $u \in \{0, \ldots, 2^h - 1\}$ compute the WOTS public key

$$p_u \leftarrow \mathsf{WOTS\text{-}genpk}(\mathsf{seed}, \mathsf{make\text{-}addr}(i, 2^h j' + u))$$

- compute the Merkle authentication $A_i \leftarrow \mathsf{Merkle\text{-}auth}_h(p_0, \ldots, p_{2^h-1}, j - 2^h j')$,
- set $j \leftarrow j'$.

- For $0 \leq u < 2^{c+h}$ compute the WOTS public key

$$p_u \leftarrow \mathsf{WOTS\text{-}genpk}(\mathsf{seed}, \mathsf{make\text{-}addr}(0, u))$$

- Compute the Merkle authentication

$$(a_1, \ldots, a_{h+c}) \leftarrow \mathsf{Merkle\text{-}auth}_{h+c}(p_0, \ldots, p_{2^{h+c}-1}, 2^h j)$$

- Set $A \leftarrow (a_{h+1}, \ldots, a_{h+c})$.
- The signature is $(s, \sigma_d, oct, \sigma_{d-1}, A_{d-1}, \ldots, \sigma_0, A_0, A)$.

Verification

$\mathcal{V}$ takes as input a hash $m \in B_n$, a public key $pk \in B_n$ and a signature

$$(s, \sigma_d, oct, \sigma_{d-1}, A_{d-1}, \ldots, \sigma_0, A_0, A)$$

and verifies it as follows.

- Compute the hyper-tree index and random subset as $j, (x_1, \ldots, x_k) \leftarrow \mathsf{PORS}(s, m)$.
- Compute the PORST public key $p \leftarrow \mathsf{PORST\text{-}extractpk}(x_1, \ldots, x_k, \sigma_d, oct)$.
- If $p = \perp$, then abort and return 0.
- For $i \in \{d-1, \ldots, 0\}$ do the following:
  - compute the WOTS public key $p \leftarrow \mathsf{WOTS\text{-}extractpk}(p, \sigma_i)$,
  - set $j' \leftarrow \lfloor j/2^h \rfloor$,
  - compute the Merkle root $p \leftarrow \mathsf{Merkle\text{-}extract}_h(p, j - 2^h j', A_i)$,
  - set $j \leftarrow j'$.
- Compute the Merkle root $p \leftarrow \mathsf{Merkle\text{-}extract}_c(p, j, A)$.
- The result is 1 if $p = pk$, and 0 otherwise.

Batch signature

$\mathcal{S}_B$ takes as input a sequence of messages $(M_1, \ldots, M_i) \in \mathcal{M}^i$ with $0 < i \leq 2^b$ and a secret key $sk = (\mathsf{seed}, \mathsf{salt})$ along with its secret cache, and outputs i signatures σ_j, computed as follows.

- For $j \in \{1, \ldots, i\}$ compute the message digest $m_j \leftarrow H^*(M_j)$.
- For $j \in \{i+1, \ldots, 2^b\}$ set $m_j \leftarrow m_1$.
- Compute $m \leftarrow \mathsf{Merkle\text{-}root}_b(m_1, \ldots, m_{2^b})$.
- Compute $\sigma \leftarrow \mathcal{S}(sk, m)$.
- For $j \in \{1, \ldots, i\}$ the j-th signature is $\sigma_j \leftarrow (j, \mathsf{Merkle\text{-}auth}_b(m_1, \ldots, m_{2^b}, j), \sigma)$.

For $b = 0$, we simplify $\mathcal{S}_B(sk, M)$ to $\mathcal{S}(sk, H^*(M))$.

Batch verification

$\mathcal{V}_B$ takes as input a public key pk, a message $M \in \mathcal{M}$ and a signature (j, A, σ), and verifies it as follows.

- Compute the message digest $m \leftarrow H^*(M)$.
- Compute the Merkle root $m \leftarrow \mathsf{Merkle\text{-}extract}_b(m, j, A)$.
- The result is $\mathcal{V}(pk, m, \sigma)$.

For $b = 0$, we simplify $\mathcal{V}_B(pk, M, \sigma)$ to $\mathcal{V}(pk, H^*(M), \sigma)$.

7.2 Proposed instances

We now propose concrete instances of parameters and primitives for Gravity.

7.2.1 Parameters

We propose the following parameters.

- Hash output $n = 256$ bits, to aim for 128 bits of security for collision-resistance, both classical and quantum.
- Winternitz depth $w = 16$, a good trade-off between size and speed often chosen in similar constructions (XMSS, SPHINCS).
- A PORS set size $t = 2^{16}$, here again a good trade-off between size and speed chosen in SPHINCS.

Given these, we propose several sets of parameters, summarized on Table 7.1.

- Three modes suitable for the NIST call for proposals for post-quantum signature schemes. Submission requirements mandate a capacity of at least 2^{64} messages per key pair [NIS16, Section 4.A.4]. We propose several trade-offs between signing time and signature size.
- A mode suitable to sign up to 2^{50} messages, for comparison with SPHINCS [BHH+15].
- A batched mode, suitable to sign up to 2^{40} batches. This is a reasonable alternative for a capacity of 2^{50} messages (with batches of 2^{10} messages), for applications that can handle batching.
- A small mode with a capacity of 2^{10} messages, for applications that don't need to sign many messages.

Verification times are similar in all cases, and much faster than signing.

name	$\log_2 t$	k	h	d	c	sigsz	capacity
NIST-fast	16	28	5	10	14	35168	2^{64}
NIST	16	28	8	6	16	26592	2^{64}
NIST-slow	16	28	14	4	8	22304	2^{64}
fast	16	32	5	7	15	28928	2^{50}
batched	16	32	8	3	16	20032	2^{40}
small	16	24	5	1	10	12640	2^{10}

Table 7.1: Proposed parameters for Gravity, suitable for 128 bits of post-quantum security. The capacity is the number of messages (or batches of messages) that can be signed per key pair. The maximal signature size sigsz is given in bytes, and does not include batching. Public keys are always 32 bytes, with secret keys of 64 bytes.

7.2.2 Primitives

For the hash functions, we propose to use a 6-round version of Haraka-v2-256 as F, and a 6-round version of Haraka-v2-512 as H. We extend the original Haraka-v2 construction [KLMR16] with an additional round; the round constants for this new round are recalled in Section 3.4.1.

For the general-purpose hash function H^*, we propose to use BLAKE2b [ANWW13].

Construction of G

We propose a construction based on AES-256 for G, valid as long as the parameters verify the constraints $c + dh \leq 64$ and $\max(\ell, t) \leq 2^{31}$. More precisely, given a seed $s \in B_n$ and an address $a = (i, j, \lambda) \in \mathcal{A}$, we compute $G(s, a)$ as follows.

- Compute $P_0 \leftarrow \lceil j \rfloor_{64} || \lceil i \rfloor_{32} || \lceil 2\lambda \rfloor_{32}$ and $P_1 \leftarrow \lceil j \rfloor_{64} || \lceil i \rfloor_{32} || \lceil 2\lambda + 1 \rfloor_{32}$, where $\lceil x \rfloor_m$ denotes the (bytewise) little-endian encoding of x as an m-bit number.

- The result is AES-256(s, P_0)||AES-256(s, P_1), i.e. the encryption of P_0 and P_1 with key s.

We recall that due to the constraints on (c, d, h), the in-layer index j verifies $0 \leq j < 2^{c+dh} \leq 2^{64}$, and the counter λ verifies $2\lambda + 1 < 2^{32}$.

This construction is essentially AES-256 in counter mode, except that we need two AES blocks for a 256-bit result. We use little-endian encoding of the counter λ because this allows to use optimized addition arithmetic on processors supporting SSE2 instructions. Also, we note that the seed s is the same throughout the hyper-tree, which allows a signer to cache the AES round keys.

7.2.3 Security

Security of Gravity relies on the collision resistance of F, H, H^*, on the undetectability and one-wayness of F and on the pseudo-randomness of G. Security reductions of Chapter 6 give lower bounds on the complexity of attacks. We now outline some concrete attacks.

First, one can try to find two messages that collide for H^*, because their signatures would be identical. A generic birthday attack has a complexity of 128 bits for $n = 256$.

Second, one can try to break the non-adaptive subset-resilience of G. Here again, our choices of parameters guarantee a complexity of at least 128 bits for known generic attacks (see Chapter 4).

Third, if two secret values (in any of the WOTS and PORST instances) are identical, knowing one allows to forge another. With $n = 256$, this gives 128 bits of security if the secret values are chosen independently and uniformly. However, our construction with AES-256 guarantees that all secret values are distinct throughout the construction, because G is in fact a permutation!

7.3 Optimized implementation

We now outline strategies that can be applied to implement Gravity in an optimized fashion.

7.3.1 Primitives

We can first apply optimizations specific to the chosen primitives.

Caching of AES round keys

To generate secret values with G, the same seed is used throughout the construction. With our implementation based on AES-256 in counter mode, this seed is the AES key. To avoid recomputing them for each block, we can cache the AES round keys throughout the scheme.

Haraka pipelining

The Haraka hash function [KLMR16] was designed to support parallel computation on several inputs for CPUs supporting optimized instructions, e.g. Intel's Haswell and Skylake micro-architectures. Typically, a CPU core can evaluate Haraka on 4 to 8 inputs at the same time, depending on the micro-architecture. Hence, careful scheduling of hash evaluations is a way to improve the speed of Gravity.

In particular, the WOTS construction uses many long chains of hashes. A naive implementation evaluates the chains one after another, which does not leverage 4-way (or 8-way) hashing. On the other end, a more clever implementation evaluates the chains level by level, using the pipelined versions of Haraka. Likewise, Merkle trees can be compressed level by level.

An even more efficient strategy is to fully compute the first 4 chains (using 4-way Haraka), then the next 4 chains, and so on. Indeed, this avoids expensive loads and stores between CPU registers and the rest of the memory. This is even more effective for mask-less constructions, because there is no need to load a mask from memory after each iteration. We improved the optimized Haraka implementation[1] to support computation of mask-less hash chains, removing useless store and load instructions at each iteration. This proved to be the most efficient strategy.

[1] Available at `https://github.com/kste/haraka`

AES-NI

Recent processors (e.g. from Intel) support so-called AES-NI instructions that compute AES rounds or key schedule faster than manually (and in constant time). This can be used to speed-up computation of hash function such as Haraka. This also speeds-up computation of AES-256 in counter mode, and in this case the counter can be incremented directly in a 128-bit register thanks to SSE2 instructions – e.g. with the `_mm_add_epi32` intrinsic[2] for a 32-bit counter. The performance benefits of AES-NI have been an important factor in our choice of primitives.

7.3.2 Secret cache

As mentioned in Section 3.2, the top levels of the root Merkle tree can be cached by the signer, as they are shared among all signatures. In particular, given the threshold c, the 2^c hash values at level c can be cached with $2^c n$ bits of memory. Further, the levels above it total only $2^c - 1$ additional hash values, so a good strategy is to save all values from levels 0 to c, with $(2^{c+1} - 1)n$ bits of memory. For our sets of parameters proposed in Table 7.1, this represents 16 KiB to 4 MiB of secret cache.

7.3.3 Multi-threading

To reduce signature size, the slower versions of Gravity use larger Merkle trees, at the expense of key generation and signing times. To reduce the latency of these operations, we can leverage multi-threading, especially in Merkle trees. Indeed, computing the root of Merkle tree of height h can be distributed among 2^τ threads as follows: split the tree into 2^τ subtrees of height $h - \tau$ (starting from the leaves), compute each subtree in a different thread, and then compute the top τ layers in a single thread. The latency of this computation is now in the order of $2^{h-\tau} + 2^\tau$, instead of 2^h.

This strategy is especially relevant in a batching context: instead of computing many independent signatures in parallel the signer computes a single signature, which means that many parallel threads of computation are available for one signature.

7.3.4 Cost estimation

We briefly estimate the cost of each operation (key generation, signing and verification) in terms of function calls. We let aside calls to the general-purpose hash function H^*, whose performance depends on the length of the message being signed.

Key generation We compute the top Merkle tree:

- $2^{c+h}\ell$ calls to G to generate the WOTS secret values,
- $2^{c+h}\ell(w-1)$ calls to F to evaluate WOTS chains,
- $2^{c+h} - 1$ calls to H to compress the Merkle tree.

The bottleneck is the evaluation of WOTS hash chains.

[2] Intel's intrinsics definitions are available at `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`

Signing Assuming that the top c levels of the hyper-tree are cached, we compute a PORST signature and d Merkle trees:

- 2 calls to H and a few calls to G to obtain the random subset of PORST,
- t calls to G to generate the PORST secret values,
- $t - 1$ calls to H to compress the PORST tree,
- $d2^h\ell$ calls to G to generate the WOTS secret values,
- $\leq d2^h\ell(w-1)$ calls to F to evaluate partial WOTS chains,
- $d(2^h - 1)$ calls to H to compress the d Merkle trees.

Here again the bottleneck is the evaluation of many WOTS hash chains.

Verification We verify a PORST instance and d Merkle trees:

- 1 call to H and a few calls to G to obtain the random subset of PORST,
- k calls to F to compute PORST public values,
- $\leq k(\log_2 t - \lfloor \log_2 k \rfloor)$ calls to H to compress octopus authentication nodes,
- $\leq d\ell(w-1)$ calls to F to evaluate partial WOTS chains,
- $c + dh$ calls to H to compress Merkle authentication paths.

The bottleneck is again the evaluation of WOTS hash chains, but verification is much faster than signing and key generation.

Take-away We can see that in all cases the bottleneck is on WOTS hash chains, so it is worth focusing optimization efforts at this level.

Conclusion

In this thesis, we have studied how to improve stateless hash-based signature schemes. These schemes rely on a limited number of assumptions, that are the existence of collision-resistant, one-way, undetectable and pseudo-random function families. This means that their security is relatively well-understood, even against hypothetical adversaries with a quantum computer. Besides, the *stateless* property gives some "misuse-resistance" guarantees for signers that cannot reliably maintain a state over the lifetime of a key pair.

Another advantage of hash-based signatures is speed and simplicity of verification. On the signer side, we have seen that several trade-offs are available between signature size, computational resources, memory resources, as well as the planned number of signatures issued by a key pair.

In this project, we proposed more secure constructions to reduce the size of stateless hash-based signatures, without compromising on speed.

We first gave an extensive analysis of the *subset-resilience* problem. We showed that adaptive attacks are much more efficient than non-adaptive ones, and gave examples of forgeries against the original HORS construction [RR02]. We then introduced *weak message attacks*, that also apply in a non-adaptive scenario. We reviewed the security of multi-instance subset-resilience as in SPHINCS, comparing several attack strategies.

We then proposed two new constructions. First, *PRNG to obtain a random subset* is a variant of *hash to obtain a random subset* designed to avoid attacks against weak messages. An version for SPHINCS allows to reduce the degrees of freedom of an adversary, by forcing the selection of a hyper-tree leaf, which further increases the security margin. Second, *octopus* allows to optimally authenticate multiple leaves in a Merkle tree. We evaluated the expected, best-case and worst-case size benefits, showing that this can save around 2 KiB per signature for SPHINCS on average.

We also explored new paradigms for hash-based signatures. Batch signing allows to reduce the total number of message signed, and in turn decrease signature sizes. We studied how removing XOR masks in constructions leads to a simpler design without compromising security, given that second-preimage resistance and collision resistance have similar generic security levels against quantum computers [Ber09]. We showed that at the expense of a reasonable secret cache, signature sizes can be reduced by a significant amount.

We leave open some fundamental questions for future work. First, is LWOTS an optimal one-time signature scheme, in terms of signature size for a given message space and computation time? A conjecture by Bleichenbacher and Maurer [BM96a] is related to the question, although it considers the number of hash evaluations rather than the signature size. Second, is there a better few-time signature scheme than PORST with octopus authentication?

There is little hope that a variant of BiBa [Per01] can perform better, given that HORS was described as "better than BiBa", and simple estimations seem to agree with this statement.

Regarding implementation, it would be interesting to study whether large Merkle trees of WOTS public keys could benefit from graphics cards, thanks to general-purpose GPU (GPGPU) programming. Indeed, these trees consist of a large number of similar computations on independent data: only a small secret seed is shared, and each WOTS instance is fully determined by its address. This could further speed-up computations for the signer, allowing in turn to use larger trees for smaller signature sizes.

The choice of primitives is also paramount to the performance of hash-based signature schemes. Haraka [KLMR16] was designed specifically for this target application in mind, and as such it came with performance improvements over general-purpose hash functions. This is why an important question is whether there exist more efficient (yet secure!) constructions, especially in the context of hash chains. More tests on more platforms would be needed to determine whether 5-round Haraka with masks loaded from memory is slower or faster than 6-round Haraka without masks. Further cryptanalysis would also be welcomed in this direction, as special-purpose hash functions such as Haraka are still young.

This thesis will be the basis for a proposal to the post-quantum cryptography standardization process held by NIST [NIS16].

Bibliography

[AHMP08] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. SHA-3 proposal BLAKE. *Submission to NIST*, 2008.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, pages 119–135, 2013.

[BC92] Jurjen N. Bos and David Chaum. Provably unforgeable signatures. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, pages 1–14, 1992.

[BDE+11] Johannes A. Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. In *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, pages 363–378, 2011.

[BDH11] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, pages 117–129, 2011.

[BDK+07] Johannes A. Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle signatures with virtually unlimited signature capacity. In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, pages 31–45, 2007.

[BDS08] Johannes A. Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, pages 63–78, 2008.

[Ber09] Daniel J Bernstein. Cost analysis of hash collisions: Will quantum computers make sharcs obsolete? *SHARCS'09 Special-purpose Hardware for Attacking Cryptographic Systems*, page 105, 2009.

[BGD+06] Johannes A. Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. CMSS - an improved merkle signature scheme. In *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings*, pages 349–363, 2006.

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 236–250, 1998.

[BH16] Leon Groot Bruinderink and Andreas Hülsing. "oops, I did it again" - security of one-time signatures under two-message attacks. *IACR Cryptology ePrint Archive*, 2016:1042, 2016.

[BHH+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 368–397, 2015.

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN '98: Theoretical Informatics, Third Latin American Symposium, Campinas, Brazil, April, 20-24, 1998, Proceedings*, pages 163–169, 1998.

[BM94] Daniel Bleichenbacher and Ueli M. Maurer. Directed acyclic graphs, one-way functions and digital signatures. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, pages 75–82, 1994.

[BM96a] Daniel Bleichenbacher and Ueli M. Maurer. On the efficiency of one-time digital signatures. In *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, pages 145–158, 1996.

[BM96b] Daniel Bleichenbacher and Ueli M Maurer. Optimal tree-based one-time digital signature schemes. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 361–374. Springer, 1996.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 470–484, 1997.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology -*

EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings, pages 409–426, 2006.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[DOTV08] Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, pages 109–123, 2008.

[DSS05] C. Dods, Nigel P. Smart, and Martijn Stam. Hash based digital signature schemes. In *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, pages 96–115, 2005.

[Dwo15] Morris J Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. *Federal Inf. Process. Stds.(NIST FIPS)-202*, 2015.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[Fia89] Amos Fiat. Batch RSA. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 175–185, 1989.

[GM16] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the AES round function. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 95–125, 2016.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219, 1996.

[HBB12] Andreas Hülsing, Christoph Busold, and Johannes A. Buchmann. Forward secure signatures on smart cards. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, pages 66–80, 2012.

[HM02] Alejandro Hevia and Daniele Micciancio. The provable security of graph-based one-time signatures and extensions to algebraic signature schemes. In *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, pages 379–396, 2002.

[HRB13] Andreas Hülsing, Lea Rausch, and Johannes A. Buchmann. Optimal parameters for XMSS MT. In *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*, pages 194–208, 2013.

[HRS16] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, pages 387–416, 2016.

[Hül13] Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, pages 173–188, 2013.

[Jea16] Jérémy Jean. Cryptanalysis of haraka. *IACR Trans. Symmetric Cryptol.*, 2016(1):1–12, 2016.

[JLMS03] Markus Jakobsson, Frank Thomson Leighton, Silvio Micali, and Michael Szydlo. Fractal merkle tree representation and traversal. In *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, pages 314–326, 2003.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[Kat16] Jonathan Katz. Analysis of a proposed hash-based signature standard. In *Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings*, pages 261–273, 2016.

[KLMR16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka v2 - efficient short-input hashing for post-quantum applications. *IACR Trans. Symmetric Cryptol.*, 2016(2):1–29, 2016.

[Kö17] Stefan Kölbl. Putting wings on SPHINCS. `https://2017.pqcrypto.org/conference/slides/recent-results/koelbl.pdf`, 2017.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

[LM95] Frank T Leighton and Silvio Micali. Large provably fast and secure digital signature schemes based on secure hash functions, July 11 1995. US Patent 5,432,852.

[Mer89] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 218–238, 1989.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 400–417, 2002.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[NA12] Samuel Neves and Jean-Philippe Aumasson. Implementing BLAKE with AVX, AVX2, and XOP. *IACR Cryptology ePrint Archive*, 2012:275, 2012.

[NIS16] Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. NIST, December 2016. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf`.

[NSW05] Dalit Naor, Amir Shenhav, and Avishai Wool. One-time signatures revisited: Have they become practical? *IACR Cryptology ePrint Archive*, 2005:442, 2005.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989.

[PB99] Chris Pavlovski and Colin Boyd. Efficient batch signature generation using tree structures. In *International Workshop on Cryptographic Techniques and E-Commerce, CrypTEC*, volume 99, pages 70–77. Citeseer, 1999.

[Per01] Adrian Perrig. The BiBa one-time signature and broadcast authentication protocol. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 28–37, 2001.

[Por13] T. Pornin. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). RFC 6979 (Informational), August 2013.

[PWX03] Josef Pieprzyk, Huaxiong Wang, and Chaoping Xing. Multiple-time signature schemes against adaptive chosen message attacks. In *International Workshop on Selected Areas in Cryptography*, pages 88–100. Springer, 2003.

[RED+08] Sebastian Rohde, Thomas Eisenbarth, Erik Dahmen, Johannes A. Buchmann, and Christof Paar. Fast hash-based signatures on constrained devices. In *Smart Card Research and Advanced Applications, 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, pages 104–117, 2008.

[Rio37] John Riordan. Moment recurrence relations for binomial, poisson and hypergeometric frequency distributions. *The Annals of Mathematical Statistics*, 8(2):103–111, 1937.

[Rog06] Phillip Rogaway. Formalizing human ignorance: Collision-resistant hashing without the keys. *IACR Cryptology ePrint Archive*, 2006:281, 2006.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394, 1990.

[RR02] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Information Security and Privacy, 7th Australian Conference, ACISP 2002, Melbourne, Australia, July 3-5, 2002, Proceedings*, pages 144–153, 2002.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. *IACR Cryptology ePrint Archive*, 2017:190, 2017.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134, 1994.

[Spe28] Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.

[sta02] Fips pub 180-2, secure hash standard. *National Institute of Standards and Technology*, 2002.

[Szy04] Michael Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 541–554, 2004.

[vOW94] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *CCS '94, Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 2-4, 1994.*, pages 210–218, 1994.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 17–36, 2005.