Secure and Efficient Post-Quantum Cryptographic Digital Signature Algorithms

by

Mahmoud Yehia Ahmed Mahmoud
B.Sc., Military Technical College, 2004
M.Sc., Military Technical College, 2011

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

Secure and Efficient Post-Quantum Cryptographic Digital Signature Algorithms

by

Mahmoud Yehia Ahmed Mahmoud
B.Sc., Military Technical College, 2004
M.Sc., Military Technical College, 2011

Supervisory Committee

Dr. T. Aaron Gulliver, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Riham AlTawy, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Bruce Kapron, Outside Member
(Department of Computer Science)

ABSTRACT

Cryptographic digital signatures provide authentication to communicating parties over communication networks. They are integral asymmetric primitives in cryptography. The current digital signature infrastructure adopts schemes that rely on the hardness of finding discrete logarithms and factoring in finite groups. Given the recent advances in physics which point towards the eventual construction of large scale quantum computers, these hard problems will be solved in polynomial time using Shor's algorithm. Hence, there is a clear need to migrate the cryptographic infrastructure to post-quantum secure alternatives. Such an initiative is demonstrated by the PQCRYPTO project and the current Post-Quantum Cryptography (PQC) standardization competition run by the National Institute of Standards and Technology (NIST).

This dissertation considers hash-based digital signature schemes. Such algorithms rely on simple security notions such as preimage, and weak and strong collision resistances of hash functions. These notions are well-understood and their security against quantum computers has been well-analyzed. However, existing hash-based signature schemes have large signature sizes and high computational cost. Moreover, the signature size increases with the number of messages to be signed by a key pair.

The goal of this work is to develop hash-based digital signature schemes to overcome the aforementioned limitations. First, FORS, the underlying few-time signature scheme of the NIST PQC alternate candidate SPHINCS$^+$ is analyzed against adaptive chosen message attacks and DFORS, a few-time signature scheme with adaptive chosen message security, is proposed. Second, a new variant of SPHINCS$^+$ is introduced that improves the computational cost and security level. A security analysis for the new variant is presented. In addition, the hash-based group digital signature schemes, Group Merkle (GM) and Dynamic Group Merkle (DGM), are studied and their security is analyzed. Group Merkle Multi-Tree (GM$^{\mathrm{MT}}$) is proposed to solve some of the limitations of the GM and DGM hash-based group signature schemes.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

ACMA	Adaptive Chosen-Message Attacks
BiBa	Bins and Balls
DFORS	Dynamic Forest Of Random Subsets
DGM	Dynamic Group Merkle
DM	Distinct-Function Multi-Target
EU-CMA	Existential Unforgeability under Chosen-Message Attacks
FORS	Forest Of Random Subsets
FTS	Few-Time Signature
GM	Group-Merkle
GM^{MT}	Group-Merkle Multi-Tree
GSS	Group Signature Scheme
HORS	Hash to Obtain Random Subset
IRTF	Internet Research Task Force
ITSR	Interleaved Target Subset Resilience
KEM	Key Encapsulation Mechanism
MM	Multi-Function Multi-Target
MSS	Merkle Signature Scheme
NIST	National Institute of Standards and Technology
ORS	Obtained Random Subset

OTS	One-Time Signature
PORS	PRNG to Obtain Random Subset
PPT	Probabilistic Polynomial Time
PQ	Post-Quantum
PRNG	Pseudo Random Number Generator
PRF	Pseudo Random Function
RFC	Request For Comments
SM	Single-Function Multi-Target
SR	Subset Resilience
TSR	Target Subset Resilience
WOTS	Winternitz One-Time Signature
XMSS	eXtended Merkle Signature Scheme
XMSS-T	eXtended Merkle Signature Scheme with Tightened security

ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude and thanks to the Almighty Allah, the most gracious and the most merciful, who blessed us with the ability to think, everything we have, and gave me the power to finish this work.

I wish to address my sincere thanks to my supervisors Professor T. Aaron Gulliver and Professor Riham AlTawy for their help, support, and valuable and insightful discussions and comments throughout my Ph.D. research.

I wish to express my gratitude towards the Government of Egypt for funding my Ph.D. research, and the Government of Canada for supporting international students.

More personally, I cannot forget those who supported me during my Ph.D. research. I would also like to thank my dear wife for her support and taking care of me and our children during my Ph.D. research. There are no words that can express my appreciation and love to my parents for their encouragement, support, and for their sincere wishes. Every time I talk to them, they pray for me, which gives me the desire and power to do better.

Finally, I would like to thank every single person and friend who helped me; had sincere wishes for me; gave me useful advice; or prayed for me. My best wishes to everyone for a wonderful life and a better world.

Chapter 1

Introduction and Motivation

A signature scheme allows a signer to generate a secret-public key pair. The secret key is kept secret and the public key is distributed between the users. For authenticated communication, the message is signed by the signer's secret (private) key, and it is verified by one of the other users called the verifier using the signer's public key.

The current digital signature infrastructure adopts schemes that rely on the hardness of finding discrete logarithms and factoring in finite groups. Given recent advances in physics which point towards the eventual construction of large scale quantum computers, these hard problems will be solved in polynomial time using Shor's algorithm. Hence, there is a clear need to migrate the cryptographic infrastructure to post-quantum secure alternatives. Such an initiative is demonstrated by the current Post-Quantum Cryptography (PQC) standardization competition run by the National Institute of Standards and Technology (NIST).

The proposals submitted to NIST for the post-quantum cryptography competition indicate that there are five approaches that claim to achieve security against quantum computer attacks. These approaches are based on lattices, coding, multivariate polynomials, isogeny, and hash functions. Hash functions have been extensively researched and they have security

properties that are easy to understand and their security with respect to quantum adversaries has been formally analyzed in [1], thus they can provide security against quantum attacks. Accordingly, this research considers hash-based digital signature schemes. The main advantage of these digital signature schemes is that their security depends on the security of the hash function that the schemes are instantiated with. Thus, in the event of a security attack on the hash function used in the digital signature scheme, a failure in the infrastructure that uses hash-based digital signature schemes is mitigated by re-parameterizing the schemes with another secure hash function. In addition, the secret and public key sizes are short compared to other post-quantum proposals. Note that the security of cryptographic algorithms that rely on computational assumptions such as factoring and finding discrete logarithms may be proved in the standard model. On the other hand, the security of cryptographic algorithms that require the randomness of hash functions is very hard to prove in the standard model. In such cases, the random oracle model is adopted where the hash function is replaced with an ideal theoretical blackbox (random function). Since standard assumptions are considered weaker (better) than those that rely on random oracles, number theoretic digital signature algorithms offer high security assurance than their hash-based counterparts that rely on random oracles. In the quantum world, such number theoretic assumption are broken in polynomial time and hash function standard security assumptions still stand. The disadvantages of hash-based digital signature schemes are their large signature sizes and high computational complexity that grow with the number of messages to be signed. Because of these limitations, the focus of this research is on improving the performance of hash-based digital signature schemes.

Hash-based digital signature algorithms are comprised of two schemes, an underlying signing scheme and an extension algorithm. The former defines the main signing procedure

where a key pair can be used to sign one (Lamport [2], Winternitz one-time signature scheme (WOTS), WOTS++ [3,4]) or a few messages (Biba [5], HORS [6], HORS++ [7], PORS [8], and FORS [9]), after which a new key pair should be generated to maintain security against forgery attacks. More precisely, the security of hash-based few-time (HBFT) signature schemes decreases after revealing each signature, and hence their bit security is given under the condition that re-keying is required after r signatures. Accordingly, translating this constraint to attack models implies that a maximum of r queries are allowed to the signing oracle.

An extension algorithm is a top-level construction that employs several instances of an underlying signing scheme (OTS or HBFT) in a Merkle tree structure. Such an algorithm enables signing multiple messages with signatures verified using one public key (Merkle root). Extension algorithms can be stateful such as Merkle Signature Scheme MSS [10], eXtended Merkle Signature Scheme (XMSS) [11], XMSS+ [12], Multi Tree XMSS (XMSS^{MT}) [13], and XMSS with tightened security (XMSS-T) [14], or stateless such as SPHINCS [15], SPHINCS$^+$ [9, 16], and Gravity SPHINCS [17]. Stateless signature algorithms conform to the basic definition of digital signatures where no state updates are required to guarantee security, and only keys are needed to securely generate valid signatures at any time.

1.1 Problem Statement and Motivation

SPHINCS$^+$ is a third-round alternate candidate in the NIST PQC competition. NIST announced that the hash-based signature scheme, SPHINCS$^+$ is the least likely of the post-quantum signature candidates to be broken by cryptanalysis techniques [18]. However, it has two major limitations that restrict its deployment in infrastructure, large signature sizes and slow signing performance. Specifically, the signing procedure of SPHINCS$^+$ is

considered slow compared to other candidates [18], and the resulting signatures are very large. For example, compared to the NIST finalist Crystals-Dilithium [19], the smallest SPHINCS$^+$ signature is four times larger, and signing is a thousand times slower [18]. For this reason, NIST considers SPHINCS$^+$ a conservative candidate but kept it as an alternate for standardization in the event there are applications that can tolerate longer signatures and slower signing. Despite SPHINCS$^+$ being a conservative choice for standardization, NIST may standardize it if the confidence in the security of final candidates is compromised by new cryptanalysis techniques. Furthermore, if some applications need very high security and can tolerate low performance, then NIST may standardize it in the future.

Given that group signature primitives are integral components of the cryptographic infrastructure, it is natural for researchers to explore the design of post-quantum group signature schemes. In 2018, El Bansarkhani and Misoczki introduced Group Merkle (GM), the first stateful hash-based group signature scheme [20]. GM has small signature sizes but large keys. It is a one-layer Merkle tree construction, thus limiting the maximum achievable tree height and accordingly, limiting the number of signatures that can be issued by the whole group. They claimed that multi-tree schemes are not applicable for group-based schemes without providing an explanation. In addition, GM does not have a revocation mechanism.

Buser *et al.* introduced the Dynamic Group Merkle (DGM) signing scheme [21], where the group manager assigns OTS keys to the group members who have used all their keys, and can add new group members after the group public key has been generated. DGM allows renewing signing trees as they are used up. It requires that the group manager store all OTS keys assigned to the users in order to reveal their identity when needed. Each group member needs to store their assigned OTS keys with the authentication paths which increases the

storage requirements. One security requirement for group signature schemes is that the group manager should not be able to generate a valid signature on behalf of any group member. In DGM, the group manager generates the signing keys of all group members, so the group manager can generate a valid signature on behalf of any group member. Further, GM and DGM were designed as generic constructions that can employ any stateful Merkle hash-based signature scheme, but their setup phases do not enable drop-in instantiation by XMSS-T, the latest and most efficient MSS variant.

1.2 Research Objectives and Organization

The goal of this research is to improve the security and performance of hash-based signature schemes and provide solutions to the limitations of current hash-based group signature schemes. The research objectives and organization of this dissertation are as follows.

Background (Chapter 2). Chapter 2 presents the basic definitions and security notions for hash function families. The related work on hash-based signature schemes is also surveyed.

A Dynamic FORS with Adaptive Chosen Message Security (Chapter 3). FORS, the most recent FTS, has been adopted in SPHINCS$^+$. The security of FORS against Adaptive Chosen Message Attacks (ACMA) is analyzed. It is shown that its bit security with respect to ACMA decreases significantly when compared to its security in a non-adaptive setting. Then DFORS, an FTS, is proposed as a new HORS variant that resists adaptive chosen message attacks. It is shown that the bit security of DFORS with respect to ACMA is more than that of FORS by a factor of $r+1$, where r is the number of signed messages per key under a given security level. DFORS theoretical computational and communication performance is also given and compared with FORS and other HORS variants. The contributions of this chapter were published in [22].

Verifiable ORS Generation: Improving The Performance of SPHINCS$^+$ (Chapter 4). v-ORS, a new mechanism that is used in the signing procedure, is proposed to enhance the security and performance of SPHINCS$^+$. As v-ORS strengthens the security of SPHINCS$^+$, different parameter sets are explored for the underlying few-time signing scheme, FORS, and new instances are given that achieve up to a 27% reduction in the signing computational complexity of SPHINCS$^+$ while maintaining the claimed security. The contributions of this chapter were published in [23].

Security Analysis Of DGM and GM Group Signature Schemes Instantiated With XMSS-T (Chapter 5). The setup phases of both GM and DGM restrict them from being directly instantiated using XMSS-T, the latest and most efficient MSS variant, which negatively affects the performance of both schemes because they may use earlier MSS versions with sub-optimal parameters. Accordingly, simple changes in the GM and DGM setup phases are proposed that enable their instantiation with XMSS-T. The bit security of DGM is analyzed when it is instantiated with XMSS-T and it is shown that it is vulnerable to multi-target attacks due to allowing multiple signing trees to branch out from the same internal node of the initial Merkle tree. Thus, a DGM variant is proposed that mitigates this multi-target attack and maintains the same bit security as the utilized scheme. The contributions of this chapter were published in [24].

New Revocable Group Merkle Multi-Tree Signature Scheme (Chapter 6). GM^{MT}, a revocable hash-based group signature scheme, is proposed that enables 2^{64} signatures per group public key. It utilizes an adaptively growing multi-tree Merkle approach which periodically creates a new GM tree. Consequently, GM^{MT} enables group members to renew their signing leaves without changing the group public key. A revocation algorithm is introduced that maintains the anonymity of revoked members while enabling the linkability

of their revoked signatures. $\mathsf{GM}^{\mathrm{MT}}$ relies on symmetric encryption and hashing such that the membership verification cost is logarithmic in the number of revoked signatures and the required storage at the group manager is linear in the number of members. Detailed comparisons between $\mathsf{GM}^{\mathrm{MT}}$ and GM and DGM are given. To demonstrate the validity of $\mathsf{GM}^{\mathrm{MT}}$, its procedures are implemented using the C language and the performance in terms of the number of clock cycles is evaluated.

Conclusion (Chapter 7). Some conclusions are given as well as some suggestions for future work.

Chapter 2

Hash-Based Digital Signature Schemes

In this chapter we first recall the security notions for hash functions. Then, we provide a summary of the previous work on hash-based digital signature schemes.

2.1 Hash Function Families

In this section, we provide the notation and security definitions of hash functions that will be used throughout the dissertation. In addition to the standard one-wayness and strong and weak collision resistance security notions, we consider security notions of hash function families which have been introduced in [14]. In what follows, let $n \in \mathbb{N}$ be the security parameter, $k = poly(n)$, $m = poly(n)$, $\mathcal{H}_n = \{H_K(M) : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n\}$ be a keyed hash function family, $K \in \{0,1\}^k$ is the hash key, and $M \in \{0,1\}^m$ is the message. Hash-based signature schemes usually adopt parameterized hash functions with $m, k \geq n$. In the security definitions given below, we refer to H as a multi-function if the hash keys are chosen at random, and we call it a distinct function when the keys are distinct but not necessarily random.

2.1.1 Quantum Accessible Random Oracle Model (QROM)

In the security analysis throughout the dissertation, we assume the QROM model [25], where all honest parties perform classical computations and only the adversary has quantum capabilities. Hence, all oracles that reply on behalf of unknown keyed function work in the classical setting where no superposition queries to the quantum oracle are allowed. For the unkeyed functions which an adversary is assumed to be able to evaluate independently, the quantum adversary is assumed to have access to these quantum oracles that reply on behalf of unkeyed functions. The reader is referred to [25] and [14, 26] for more details on QROM model. Considering hash functions where a quantum adversary is searching for (second) preimages in an unstructured space, it is assumed that Grover's search algorithm is used. The generic security of the following security notions of hash function families against quantum attacks based on Grover's algorithm are formally analyzed in [14].

2.1.2 Security Notions for Hash Function Families

Definition 1 ((Post-Quantum) Preimage Resistance (PQ-OW))**.** *Given a (quantum) adversary $\mathcal{A}$ who is provided with the hash function key K and an image Y of a message M, $Y = H_K(M)$, the success probability that $\mathcal{A}$ finds a preimage of Y is given by*

$$Succ_{H_n}^{\text{PQ-OW}}(\mathcal{A}) = \mathsf{Pr}[K \leftarrow \{0,1\}^k; M \leftarrow \{0,1\}^m, Y \leftarrow H_K(M);$$
$$M' \leftarrow \mathcal{A}(K, Y) : Y = H_K(M')]$$

Definition 2 ((Post-Quantum) Single-function, Multi-target Preimage Resistance (PQ-S-M-OW))**.** *Given a (quantum) adversary $\mathcal{A}$ who is provided with the hash function key K and p images $(Y_1, Y_2, \ldots, Y_p)$ of messages $M_1, M_2, \ldots, M_p$, $Y_i = H_K(M_i)$, where $1 \leq i \leq p$, the*

success probability that $\mathcal{A}$ finds a preimage of any of the images is given by

$$Succ^{\text{PQ-SM-OW}}_{H_n,p}(\mathcal{A}) = \mathsf{Pr}[K \leftarrow \{0,1\}^k; M_i \leftarrow \{0,1\}^m, Y_i \leftarrow H_K(M_i), 1 \leq i \leq p;$$
$$M' \leftarrow \mathcal{A}(K, (Y_1, Y_2, \ldots, Y_p)) : Y_i = H_K(M')]$$

Definition 3 ((Post-Quantum) Distinct-function, Multi-target Preimage Resistance (PQ-D-M-OW))**.** *Given a (quantum) adversary $\mathcal{A}$ who is provided with p image-key pairs (Y_i, K_i), $Y_i = H_{K_i}(M_i), 1 \leq i \leq p$, the success probability that $\mathcal{A}$ finds a preimage of any pair, j, $1 \leq j \leq p$ using the corresponding hash function key, K_j, is given by*

$$Succ^{\text{PQ-DM-OW}}_{H_n,p}(\mathcal{A}) = \mathsf{Pr}[K_i \leftarrow \{0,1\}^k; M_i \leftarrow \{0,1\}^m, Y_i \leftarrow H_{K_i}(M_i), 1 \leq i \leq p;$$
$$(j, M') \leftarrow \mathcal{A}((K_1, Y_1), (K_2, Y_2), \ldots, (K_p, Y_p)) : Y_j = H_{K_j}(M')]$$

Definition 4 ((Post-Quantum) Second-Preimage Resistance (PQ-SPR))**.** *Given a (quantum) adversary $\mathcal{A}$ who is provided with the hash function key K and a message M, the success probability that $\mathcal{A}$ finds another message M' that has the same image is given by*

$$Succ^{\text{PQ-SPR}}_{H_n}(\mathcal{A}) = \mathsf{Pr}[K \leftarrow \{0,1\}^k; M \leftarrow \{0,1\}^m,$$
$$M' \leftarrow \mathcal{A}(K, M) : M' \neq M \wedge H_K(M) = H_K(M')]$$

Definition 5 ((Post-Quantum) Single-Function, Multi-target Second-Preimage Resistance (PQ-SM-SPR))**.** *Given a (quantum) adversary $\mathcal{A}$ who is provided with the hash function key K and p messages $(M_1, M_2, \ldots, M_p)$, the success probability that $\mathcal{A}$ finds a second*

preimage of any message is given by

$$Succ^{\text{PQ-SM-SPR}}_{H_n,p}(\mathcal{A}) = \mathsf{Pr}[K \leftarrow \{0,1\}^k; M_i \leftarrow \{0,1\}^m, Y_i \leftarrow H_K(M_i), 1 \leq i \leq p;$$
$$M' \leftarrow \mathcal{A}(K, (M_1, M_2, \ldots, M_p)) : M' \neq M_i \wedge H_K(M_i) = H_K(M')]$$

Definition 6 ((Post-Quantum) Distinct-function, Multi-target Second Preimage Resistance (PQ-DM-SPR))**.** *Given a (quantum) adversary $\mathcal{A}$ who is provided with p message-key pairs (M_i, K_i), $1 \leq i \leq p$, the success probability that $\mathcal{A}$ finds a second preimage of any pair (j), $1 \leq j \leq p$ using the corresponding hash function key (K_j) is given by*

$$Succ^{\text{PQ-DM-SPR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \mathsf{Pr}[K_i \leftarrow \{0,1\}^k; M_i \leftarrow \{0,1\}^m, 1 \leq i \leq p;$$
$$(j, M') \leftarrow \mathcal{A}((K_1, M_1), \ldots, (K_p, M_p)) :$$
$$M' \neq M_j \wedge H_{K_j}(M_j) = H_{K_j}(M')]$$

Note that if the keys of the hash function family are chosen randomly, then the above security notion in Definition 6 is referred to as *Multi-Function, Multi-target Second-Preimage Resistance (MM-SPR)).*

Definition 7 ((Post-Quantum) Target Collision Resistance (PQ-TCR))**.** *Given a (quantum) adversary $\mathcal{A}$ who is allowed to choose a target message M, then they are given the hash function key. After that, they are required to find another message M' that has the same image as M under the same hash function key. The success probability of $\mathcal{A}$ is given by*

$$Succ^{TCR}_{H_n}(\mathcal{A}) = \mathsf{Pr}[M \leftarrow \mathcal{A}(1^n), K \leftarrow \{0,1\}^k; (M') \leftarrow \mathcal{A}((K, M) :$$
$$M' \neq M \wedge H_K(M) = H_K(M')]$$

Definition 8 ((Post-Quantum) Extended Target Collision Resistance (PQ-eTCR))**.** *Given a (quantum) adversary $\mathcal{A}$ who is allowed to first choose a target message M and then are provided with a hash function key, K, after which, $\mathcal{A}$ needs to find another message-key pair (possibly the same key) that has the same image, $H_K(M)$. The success probability of $\mathcal{A}$ is given by*

$$Succ_{\mathcal{H}_n}^{PQ\text{-}eTCR}(\mathcal{A}) = \mathsf{Pr}[M \leftarrow \mathcal{A}(1^n), K \leftarrow \{0,1\}^k; (K', M') \leftarrow \mathcal{A}(K, M) : M' \neq M \wedge H_K(M) = H_{K'}(M')]$$

By definition, eTCR refers to a multi-function hash family.

Definition 9 ((Post-Quantum) Multi-target Extended Target Collision Resistance (PQ-M-eTCR))**.** *Given a (quantum) adversary $\mathcal{A}$ who is given a target set of p message-key pairs (M_i, K_i), $1 \leq i \leq p$, and they are required to find a different message-key pair (possibly the same key) whose image collides with any of the pairs in the target set. The success probability of $\mathcal{A}$ is given by*

$$Succ_{\mathcal{H}_n,p}^{PQ\text{-}M\text{-}eTCR}(\mathcal{A}) = \mathsf{Pr}[K_i \leftarrow \{0,1\}^k; M_i \leftarrow \{0,1\}^m, 1 \leq i \leq p; (j, K', M') \leftarrow \mathcal{A}((K_1, M_1), \ldots, (K_p, M_p)) : M' \neq M_j \wedge H_{K_j}(M_j) = H_{K'}(M')]$$

Definition 10 ((Post-Quantum) M-eTCR with Nonce (PQ-NM-eTCR))**.** *Given a (quantum) adversary $\mathcal{A}$ who is given a target set of p key-message-nonce tuples (K_i, M_i, i), $1 \leq i \leq p$, and they are required to find a different key-message-nonce tuple (K', M', j) whose image collides with the j-th tuple in the target set (possibly the same key). The success probability*

of $\mathcal{A}$ is given by

$$\begin{aligned}\mathit{Succ}^{\mathit{PQ\text{-}N M\text{-}eTCR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \mathsf{Pr}[K_i \leftarrow \{0,1\}^k; M_i \leftarrow \{0,1\}^m, 1 \leq i \leq p;\\ (K', M', j) \leftarrow \mathcal{A}((K_1, M_1, 1), \ldots, (K_p, M_p, p)) :\\ M' \neq M_j \wedge H(K_j||j, M_j) = H(K'||j, M')]\end{aligned}$$

Definition 11 ((Post Quantum) Pseudorandom Function (PQ-PRF))**.** *$\mathcal{H}_n$ is called a PRF function family, if it is efficiently computable and for any (quantum) adversary $\mathcal{A}$ who can query a black-box oracle O that is initialized with either $\mathcal{H}_n$ function or a random function $\mathcal{G}$ where $\mathcal{G} : \{0,1\}^m \rightarrow \{0,1\}^n$. $\mathcal{A}$ is required to distinguish the output of O by determining which function it is initialized with. The success probability of $\mathcal{A}$ is given by*

$$\mathit{Succ}^{\mathit{PQ\text{-}PRF}}_{\mathcal{H}_n}(\mathcal{A}) = \mid \mathsf{Pr}[O \leftarrow \mathcal{H}_n : \mathcal{A}^{O(\cdot)} = 1] - \mathsf{Pr}[O \leftarrow \mathcal{G} : \mathcal{A}^{O(\cdot)} = 1] \mid$$

Table 2.1 provides the success probability of both generic classical and quantum adversary $\mathcal{A}$ who makes q_h queries to a hash oracle against the above hash security notion, where p is the number of targets (see [14, 27] for the proofs and more details).

Table 2.1: Hash function security against generic classical and quantum attacks. Entries represent the success probability of $\mathcal{A}$ who makes q_h queries, and p represents the number of targets

	OW,MM-OW, SPR MM-SPR, PRF	SM-OW, SM-SPR TCR	eTCR	M-eTCR	NM-eTCR
Classical $\mathcal{A}$	$\frac{(q_h+1)}{2^n}$	$\frac{(q_h+1)p}{2^n}$	$\frac{(q_h+1)^2}{2^n} + \frac{q_h^2}{2^k}$	$\frac{p(q_h+1)^2}{2^n} + \frac{pq_h^2}{2^k}$	$\frac{(q_h+p)}{2^n} + \frac{pq_h}{2^k}$
Quantum $\mathcal{A}$	$\Theta(\frac{(q_h+1)^2}{2^n})$	$\Theta(\frac{(q_h+1)^2 p}{2^n})$	$\Theta(\frac{(q_h+1)^2}{2^n} + \frac{q_h^2}{2^k})$	$\Theta(\frac{p(q_h+1)^2}{2^n} + \frac{pq_h^2}{2^k})$	$\Theta(\frac{(q_h+p)^2}{2^n} + \frac{pq_h^2}{2^k})$

2.2 Digital Signature Schemes

In the following we give the definition for digital signature scheme and the existential unforgeability under adaptive chosen message attacks (EU-CMA), the standard security notion for digital signature schemes.

2.2.1 Digital Signature Scheme

A digital signature scheme is a tuple of three polynomial time algorithms *DSS = (KGen, Sign, Vf)*

- $KGen(1^n)$: The key generation algorithm takes the security parameter n as input, and outputs a pair of keys: a secret key sk (for signing) and the public key pk (for verification)
- $Sign(sk, M)$: The signing algorithm takes the secret key sk and the message M as inputs, and outputs the signature Σ
- $Vf(pk, M, \Sigma)$: The verification algorithm takes the public key pk, the message M, and the signature Σ as inputs, and outputs 1 iff the signature Σ is valid and 0 otherwise, such that the following correctness condition is achieved

$$\forall(pk, sk) \leftarrow KGen(1^n), \forall(M \in \{0,1\}^*) : Vf(pk, M, Sign(sk, M)) = 1$$

2.2.2 Existential Unforgeability Under Adaptive Chosen Message Attacks

Digital Signature Schemes are analyzed with respect to existential unforgeability under adaptive chosen message attacks (EU-CMA). EU-CMA is usually defined by a security game in which the adversary $\mathcal{A}$ who has access to the scheme's public key is allowed to ask the signing challenger, Chall, for signatures of the messages of their choice. $\mathcal{A}$ wins the game if they are able to return a message and signature pair such that the signature is valid for that message and the message is not one of the queried ones. A digital signature scheme is secure with respect to EU-CMA if the probability of $\mathcal{A}$ winning the game is negligible.

Game: $\mathsf{EU\text{-}CMA}_{DSS}(n)$
$(\mathsf{SK}, \mathsf{PK}) \leftarrow DSS.kGen(1^n)$
while $\sigma_j \leftarrow \mathcal{A}(\text{query}(M_j), \mathsf{PK}, \mathsf{Chall}^{sign(\mathsf{SK},.)})$, j++ **do;**
$(M', \sigma') \leftarrow \mathcal{A}(\text{forge}, \mathsf{PK})$
if $M' \notin \{M_1, M_2, \ldots, M_q\}$ // where $q < j$
Return $DSS.verify(\mathsf{PK}, M', \sigma')$

Figure 2.1: EU-CMA game

The success probability of $\mathcal{A}$ in the above game is given by

$$Succ_{DSS}^{\mathsf{EU\text{-}CMA}}(\mathcal{A}) = \mathsf{Pr}[\textbf{Game}: \mathsf{EU\text{-}CMA}_{DSS} = 1]$$

For a digital signature scheme DSS and a security parameter n, the formal EU-CMA security game is given by

Definition 12 (EU-CMA)**.** *Let $n \in \mathbb{N}$ be the security parameter, DSS a digital signature scheme. DSS is called EU-CMA-secure if for all $q, t = poly(n)$ the maximum success probability $InSec^{EU-CMA}(DSS(1^n); t, q)$ of all possibly quantum PPT adversaries $\mathcal{A}$ running*

in time ≤ t, making at most q queries to Sign Oracle in the above experiment, is negligible in n:

$$\mathsf{InSec}^{\text{PQ-EU-CMA}}(DSS(1^n), t, q) = \max_{\mathcal{A}}\{\mathsf{Succ}^{\text{EU-CMA}}_{DSS(1^n)}(\mathcal{A})\} = negl(n)$$

2.3 One-Time Signature Schemes OTS

In the following we give the first hash-based OTS by Lamport-Diffie and the Winternitz One-Time Signature Scheme and its variants.

2.3.1 The Lamport-Diffie One-Time Signature LD-OTS

The Lamport-Diffie One-Time Signature LD-OTS was introduced in the 1970s [2]. Let n be a positive integer. n is the output length of the following two hash functions:

$$F : \{0,1\}^n \rightarrow \{0,1\}^n,$$

$$G : \{0,1\}^* \rightarrow \{0,1\}^n,$$

LD-OTS key generation: The algorithm samples at random $2n$ secret keys $SK = sk_0[0], sk_0[1], sk_1[0], sk_1[1], \ldots, sk_{n-1}[0], sk_{n-1}[1]$ each of n bits. It applies the F function to the secret key elements and produces the public key $PK = pk_0[0], pk_0[1], pk_1[0], pk_1[1], \ldots, pk_{n-1}[n-1], pk_{n-1}[n-1]$ each of n bits, where $pk_i[j] = F(sk_i[j]),\ 0 \leq i \leq n-1, j = \{0,1\}$

LD-OTS Signing: The algorithm takes the message M and the secret key SK as inputs. It hashes the message using the hash function $G(M) = X = x_0, x_1, \ldots, x_{n-1}$ and outputs the

signature as follows:

$$\Sigma = \sigma_0, \sigma_1, \ldots, \sigma_{n-1} = sk_0[x_0], sk_1[x_1], \ldots sk_{n-1}[x-1]$$

LD-OTS Verification: The algorithm takes the public key PK, the message M, and the signature Σ as inputs. It computes the message digest $G(M) = X = x_0, x_1, \ldots, x_{n-1}$. Then it checks whether

$$F(\sigma_0), F(\sigma_1), \ldots, F(\sigma_{n-1}) \stackrel{?}{=} pk_0[x_0], pk_1[x_1], \ldots pk_{n-1}[x-1]$$

The signing and verification are fast but the signature size (n^2 bits) is large.

2.3.2 The Winternitz One-Time Signature Scheme

In this section we illustrate the WOTS one-time signature scheme [28]. It was proposed to reduce the signature size of LD-OTS at the expense of computational cost. In what follows we explain the key generation, signature and verification algorithms.

The algorithm uses the 2 hash functions F and G defined in LD-OTS. Let $n \in \mathbb{N}$. The Winternitz parameter $w \in \mathbb{N}$, $w \geq 2$, (should be power of 2, the recommended values are $\{4, 16, 256\}$) which identifies the trade-off between the signature size and the signing / verification computational cost. n and w are used to compute the required number of secret keys l as follows

$$l_1 = \lceil \frac{n}{\log_2 w} \rceil, \qquad l_2 = \lfloor \frac{\log_2(l_1(w-1))}{\log_2 w} \rfloor + 1, \qquad l = l_1 + l_2$$

WOTS Key Generation Algorithm: the algorithm takes n and w as inputs. It sam-

ples at random l secret keys each of n-bit length where l is as defined above. $SK = sk_0, sk_1, \ldots, sk_{l-1}$. It applies the F function to each secret key element for $w-1$ iterations and produces the public key $PK = F^{w-1}(sk_0), F^{w-1}(sk_1), \ldots, F^{w-1}(sk_{l-1})$, each of n bits.

WOTS Signing algorithm: it takes as input the l secret keys, the Winternitz parameter w, and the message to be signed. It hashes the message using the hash function $G(M) = X$. These n bits are divided into $l_1 = \lceil \frac{n}{\log_2 w} \rceil$ substrings $(a_1, a_2, \ldots, a_{l_1})$ each of $\log_2 w$ bits. Then it calculates the check sum CS as follows

$$CS = \sum_{i=1}^{l_1} (w - 1 - a_i)$$

This checksum CS, is divided into $l_2 = \lfloor \frac{\log_2(l_1(w-1))}{\log_2 w} \rfloor + 1$ substrings $(CS_1, CS_2, \ldots, CS_{l_2})$ each of w-bit. Let $B = b_1, b_2, \ldots, b_l = a_1, a_2, \ldots, a_{l_1}, CS_1, CS_2, \ldots, CS_{l_2}$. The signature Σ is computed by mapping the secret keys into intermediate values in each chain as follows

$$\Sigma = \sigma_1, \sigma_2, \ldots, \sigma_l = F^{b_1}(sk_1), F^{b_2}(sk_2), \ldots, F^{b_l}(sk_l)$$

The idea of using the checksum is to guarantee that having a message $B = b_1, b_2, \ldots, b_l = a_1, a_2, \ldots, a_{l_1}, CS_1, CS_2, \ldots, CS_{l_2}$ as explained above, it is infeasible to find another message $B' = b'_1, b'_2, \ldots, b'_l = a'_1, a'_2, \ldots, a'_{l_1}, CS'_1, CS'_2, \ldots, CS'_{l_2}$ such that $\forall i, 1 \leq i \leq l, b'_i \geq b_i$

WOTS Verification algorithm: it takes the public key (PK), the signature Σ, and message M as inputs. It computes the integers $B = b_1, b_2, \ldots, b_l$ like the signing algorithm. Then it checks if

$$(F^{w-1-b_0}(\sigma_0), F^{w-1-b_1}(\sigma_1), \ldots, F^{w-1-b_{l-1}}(\sigma_{l-1})) \stackrel{?}{=} (pk_0, pk_1 \ldots, pk_{l-1})$$

If equality holds, the signature is valid, otherwise it is rejected.

Later on several variants of WOTS (WOTS-SPR [3], WOTS$^+$ [4], WOTS-T [14], NOTS [29]) were introduced to improve the security and performance of the algorithm, and make the security depends on (stronger) specific security notion of F.

2.4 Few-Time Signature Schemes (FTS)

Few-time signature schemes were introduced to allow signing few messages, at the range of 10 messages, using the same key and keeping the security of the scheme at certain level. In the following we present an overview on Bins and Balls (BiBa), the first FTS scheme. For consistency we postpone the Hash to Obtain Random Subset (HORS) scheme, the second and most practical FTS, to the next chapter.

2.4.1 Bins and Balls (BiBa)

In [5], Perrig introduced Bins and Balls signing scheme (BiBa). The scheme parameters are the number of bins n and the number of balls t. Let $H_K : \{0,1\}^k \times \{0,1\}^{m_2} \rightarrow [0, n-1]$ represent a keyed hash function H with the key K, F be a PRF where $F : \{0,1\}^{m_2} \times \{0,1\}^{m_1} \rightarrow \{0,1\}^{m_2}$, and G be the message hash function where $G : \{0,1\}^* \rightarrow \{0,1\}^k$.

BiBa Key Generation Algorithm: The algorithm samples at random t secret keys each of length m_2 bits $SK = sk_0, sk_1, \ldots, sk_{t-1}$. The public key is $PK = F_{sk_0}(0), F_{sk_1}(0), \ldots, F_{sk_{t-1}}(0)$, each of length m_2 bits.

BiBa Signing Algorithm: The algorithm takes as inputs the secret key SK, and the message M. The message is hashed using the hash function $G(M) = h$. The value h is used to be the key to the hash function H. The signature $\Sigma = (i, j, sk_i, sk_j)$ where $sk_i \neq sk_j$ and $H_h(sk_i) = H_h(sk_j)$

BiBa Verification Algorithm: The algorithm takes as inputs the public key PK, the signature Σ and the message M. It computes $G(M) = h$ and checks whether $H_h(sk_i) \stackrel{?}{=} H_h(sk_j) \wedge F_{sk_i}(0) \stackrel{?}{=} pk_i \wedge F_{sk_j}(0) \stackrel{?}{=} pk_j$. If the three equalities hold, the signature is valid, otherwise it is rejected.

The algorithm signing failure probability is $\approx exp(-t(t-1)/2n)$. Accordingly, to increase this probability, t should be chosen larger than $\sqrt{n}$. The success probability of an adversary, knowing only the secret values of the received signatures, to generate a valid signature is $1/n$, assuming that it is infeasible to inverse the function F. To increase the security the author proposed to find multi-collision instead of one collision.

2.5 Many-Time Signature Schemes

In 1978 Ralph Merkle introduced Merkle Signature Scheme MSS [10]. Such an algorithm enables signing multiple messages where signatures are verified with one public key (Merkle root). MSS employs several instances of underlying signing schemes (OTS) in a Merkle tree structure. MSS is a stateful signing algorithm where the signer needs to update the secret key state (signing index) after each signature. Later, variants of MSS were introduced to enhance its security and the signature size. Such variants include eXtended Merkle Signature Scheme (XMSS) [11], XMSS+ [12], Multi Tree XMSS (XMSS^{MT}) [13], and XMSS with tightened security (XMSS-T) [14]. On the other side, other variants that employ FTS as their underlying signing scheme were introduced to achieve a stateless construction, e.g., SPHINCS [15], SPHINCS$^+$ [9, 16], and Gravity SPHINCS [17]. Stateless signature algorithms conform to the basic definition of digital signatures where no state updates are required to guarantee security, and only keys are needed to securely generate valid signatures at any time. Section 3.3 provides more details about the construction of the Merkle tree

and Section 4.1 gives the detailed description of SPHINCS$^+$ as an example of the stateless hash-based digital signature scheme.

Chapter 3

A Dynamic FORS with Adaptive Chosen Message Security

In this chapter, we analyze the security of Forest Of Random Subsets (FORS), the underlying FTS scheme in SPHINCS$^+$ digital signature scheme, with respect to adaptive chosen message attacks. We show that in such a setting, the security of FORS decreases significantly with each signed message when compared to its security against non-adaptive chosen message attacks. We propose a chaining mechanism that with slightly more computation, dynamically binds the *Obtain Random Subset* (ORS) generation with signing, hence, eliminating the offline advantage of adaptive chosen message adversaries. We apply our chaining mechanism to FORS and present DFORS whose security against adaptive chosen message attacks is equal to the non-adaptive security of FORS. In a nutshell, using SPHINCS$^+$-128s parameters, FORS provides 75-bit security and DFORS achieves 150-bit security with respect to adaptive chosen message attacks after signing one message. We note that our analysis does not affect the claimed security of SPHINCS$^+$. Nevertheless, this work provides a better understanding of FORS and other HORS variants, and furnishes a

solution if new adaptive cryptanalytic techniques on SPHINCS$^+$ emerge. The contributions of this chapter were published in [22].

3.1 Preliminaries

In what follows, we provide the notation and definitions used throughout the chapter. FORS can be seen as a generalized instance of HORS and it inherits most of the specifications of HORS. Accordingly, for completeness, we provide a brief overview of the HORS signature scheme.

3.1.1 Notation

Let n denote our security parameter. Consider a finite key space $\mathbb{K}$, message space $\mathbb{M}$ of arbitrary length messages, the two hash families H and G where $H = \{H_k : \{0,1\}^* \rightarrow \{0,1\}^{\kappa\tau} | k \in \mathbb{K}\}$, and $G = \{G_k : \{0,1\}^* \rightarrow \{0,1\}^n | k \in \mathbb{K}\}$. H_k (resp. G_k) is a $\kappa\tau$-bit (resp. n-bit) keyed one-way function. Let the $\kappa\tau$-bit message digest of an arbitrary length message $m \in \mathbb{M}$ be divided into κ elements, each of length τ bits, such that the integer representation of a given element belongs to $\{0, 1, \ldots, t-1\}$, where $t = 2^\tau$. We refer to the set $\{0, 1, \ldots, t-1\}$ by T, and the subset of κ-elements of the set T by $S_\kappa(T)$. Let $ORS_\kappa(m)$ denote an *Obtain Random Subset* function which returns a κ element subset from the $\kappa\tau$-bit hash value of a message m, formally defined as follows

$$ORS_\kappa(m) : H_k(m) \rightarrow S_\kappa(T) | k \in \mathbb{K}$$

The notion of ORS functions was introduced by Reyzin and Reyzin when HORS was proposed [6]. It has been shown that the security of the scheme is reduced to the subset resilience problem [6]. More precisely, for a given bit security level, at most r messages can

be signed before re-keying is required, otherwise an adversary can find a message whose ORS is covered by the union of the $ORSs$ of the r messages.

Definition 13. *The messages $(m_1, m_2, \ldots, m_r, m_{r+1})$ are in an r-subset-cover relation, C_κ^r, if the Obtain Random Subset of message m_{r+1} $(ORS_\kappa(m_{r+1}))$ is a subset of the union of all Obtain Random Subsets of the r-messages, $ORS_\kappa(m_1) \cup ORS_\kappa(m_2) \cup \cdots \cup ORS_\kappa(m_r)$, formally*

$$C_\kappa^r(m_1, m_2, \ldots, m_{r+1}) \Leftrightarrow ORS_\kappa(m_{r+1}) \subseteq \bigcup_{i=1}^{r} ORS_\kappa(m_i).$$

If finding the above cover relation for a given ORS function is infeasible, then it is said that such a function is r-subset resilient.

Definition 14. *An ORS function is r-subset-resilient if for any polynomial time adversary $\mathcal{A}^{(1^n,\kappa,t)}$, the probability of finding $(m_1, m_2, \ldots, m_{r+1})$ such that $ORS_\kappa(m_{r+1})$ is a subset of $ORS_\kappa(m_1) \cup ORS_\kappa(m_2) \cup \cdots \cup ORS_\kappa(m_r)$ is negligible, formally*

$$\mathsf{Pr}[(m_1, m_2, \ldots, m_{r+1}) \leftarrow \mathcal{A}^{(1^n,\kappa,t)} : C_\kappa^r(m_1, m_2, \ldots, m_{r+1})] \leq negl(n, t).$$

Definition 15. *An ORS function is r-target-subset-resilient, if for any polynomial time adversary $\mathcal{A}$ who is given the $ORSs$ of r messages $\bigcup_{i=1}^{r} ORS_\kappa(m_i)$, it is infeasible to find a message m_{r+1} such that its κ-element $ORS_\kappa(m_{r+1})$ is a subset of the union of $ORSs$ of the r messages, formally*

$$\mathsf{Pr}[(m_{r+1}) \leftarrow \mathcal{A}^{(1^n,\kappa,t,m_1,m_2,\ldots,m_r)} : C_\kappa^r(m_1, m_2, \ldots, m_{r+1})] \leq negl(n, t)$$

3.1.2 Hash to Obtain Random Subset (HORS) Few-time Digital Signature Scheme

In HORS [6], the signer randomly generates t secret keys each of n-bit length, $(SK = sk_0, sk_1, \ldots, sk_{t-1})$. Using a one-way function $f : \{0,1\}^n \rightarrow \{0,1\}^n$, the signer computes the public key, $PK = (pk_0 = f(sk_0), pk_1 = f(sk_1), \ldots, pk_{t-1} = f(sk_{t-1}))$. For signing an arbitrary length message $m \in \mathbb{M}$, $ORS_\kappa(m) = \{h_0, h_1, \ldots, h_{\kappa-1}\}$ is evaluated by dividing the $\kappa\tau$-bit message digest value of $H_K(m)$ into κ elements, each of length τ bits. Each element is represented by an integer h_i where $0 \leq i \leq \kappa - 1$ and $h_i \in \{0, 1, \ldots, t-1\}$, $t = 2^\tau$. To generate the signature, σ, the signer reveals the secret keys whose indices correspond to the integer representation of the κ elements in the ORS, i.e., $\sigma = (sk_{h_0}, sk_{h_1}, \ldots, sk_{h_{\kappa-1}})$. For verification, the verifier computes $ORS_\kappa(m) = \{h_0, h_1, \ldots, h_{\kappa-1}\}$, then checks if $f(sk_{h_i}) = pk_{h_i}$, otherwise verification fails. The description of HORS is given in Algorithm A.1 in Appendix A.1.

Security. Assuming that f is a one-way function, the security of HORS is reduced to the hardness of the (target) subset-resilience problem [6]. It has been shown that the probability of finding a message (m_{r+1}) such that $ORS_\kappa(m_{r+1})$ is covered by the obtained random subsets of the r previously signed messages is $(r\kappa/t)^\kappa$ which corresponds to the probability of κ randomly chosen elements being a subset of the revealed $r\kappa$ secret keys. The corresponding bit security is then

$$\log_2(t/r\kappa)^\kappa = \kappa(\log_2 t - \log_2 r - \log_2 \kappa).$$

In [8], it was proven that the security of HORS with respect to adaptive chosen message attacks is

$$\frac{\kappa}{r+1}(\log_2 t - \log_2 r - \log_2 \kappa) + \frac{\log_2 r!}{r+1},$$

(see Appendix A.2). A practical example of a weak-message attack was also given where an adaptive adversary finds messages that map to subsets with repeated indices which results in smaller subsets, i.e., number of distinct elements $< \kappa$. Such subsets are easier to cover and consequently, a 7-bit decrease in the expected security of SPHINCS against classical attacks was reported.

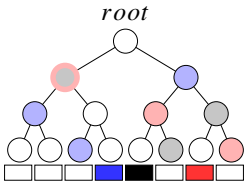
Variants. HORS++ [7] was introduced to provide security against adaptive attacks. A one-to-one mapping function $S(m)$ that belongs to a cover-free family [30] is utilized to ensure that for any $r + 1$ messages $S(m_{r+1}) \nsubseteq \bigcup_{i=1}^{r}(S(m_i)$. Three constructions for $S(m)$ based on polynomials over finite fields, error correcting codes, and algebraic curves over finite fields were presented. Consequently, HORS++ increases the signature size and the size of the secret keys to achieve the same security level of HORS against non-adaptive chosen message attacks. Moreover, the computational efficiency is decreased due to the computation of $S(m)$. Later, PORS was suggested to replace HORS in SPHINCS where the idea of having distinct elements in subsets of weak messages was enforced by use of a pseudorandom bit generator to obtain the subsets [8]. However, although PORS mitigates weak-message attacks, it is still vulnerable to adaptive chosen message attacks under the definition given in Appendix A.2. Lastly, FORS was proposed and used in SPHINCS$^+$ [9], where security against weak-message attacks is achieved by increasing the key size from t values to κt values such that each index out of the κ indices in the ORS reveals a secret key from a different pool of t secret keys.

3.2 FORS Security Analysis

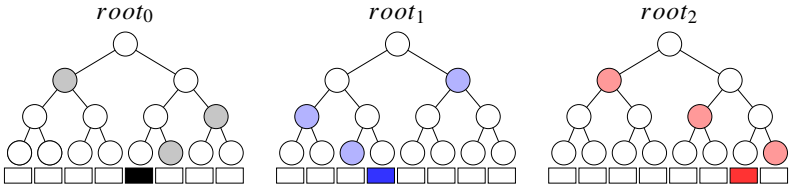
Unlike HORS which generates t secret keys from which the secret keys that are indexed by ORS(m) are released, FORS generates (κt) secret keys and dedicates t secret keys for each index out of the κ indices. By doing so, FORS mitigates weak message attacks because even if two elements in $ORS(m)$ are equal, they index values from different secret key pools. The n-bit public key of FORS is the hash of the concatenation of κ Merkle tree roots. Each root is associated with a binary hash tree whose leaves are the hashes of t secret key elements in a given pool. Accordingly, one FORS instance has κ trees, each of height $\log t = \tau$.

Figure 3.1 depicts the signatures of message 100 011 110 using (a) HORS and (b) FORS, where $\kappa = 3$ and $t = 8$. In FORS, the first 3 bits, i.e., 100, of the message selects sk_4, the secret key corresponding to the 4-th leaf indexed from the left and starting from 0 in the first tree along with its authentication path to $root_0$. Similarly, the second (resp. third) 3 bits of the message selects sk_3 (resp. sk_6) from the second (resp. third) tree with the authentication path to $root_1$ (resp. $root_2$). In HORS, the three 3-bit parts of the message index sk_4, sk_3, and sk_6 from the same tree, and with each selected secret key a 3 node authentication path is selected, hence the overlap in the node (colored in pale red and gray) at the pre-root level. More details about hash trees and authentication path calculations are provided in Section 3.3.

It can be verified from Figure 3.1 that if two 3-bit parts of the message are equal, then the same secret key value is revealed in HORS. This fact is exploited in the weak messages attack where an adversary searches for a message that has as many repeated ORS elements as possible, which lead to an ORS set that contains fewer distinct elements, and thus can be easily covered with the ORSs of the revealed r messages. However, this problem is



(a) HORS signature within a binary tree construction



(b) FORS signature within κ binary trees construction

Figure 3.1: HORS and FORS signatures of the message 100 011 110 where $\kappa = 3$ and $t = 8$. The 8 rectangles under each tree depict the eight secret keys whose hashes are stored in the corresponding leaf nodes.

mitigated in FORS because repeated ORS elements select secret keys from different pools. In what follows, we investigate the security of FORS with respect to non-adaptive chosen message attacks.

3.2.1 FORS in a Non-adaptive Setting

Reyzin and Reyzin introduced clear attack models for analyzing HBFT signature schemes against (non) adaptive chosen message attacks [6]. Such models are used in the analysis of all HORS-variants, i.e., PORS, and FORS. Specifically, in a non-adaptive setting, also referred to by r-target subset resilience problem (see Def. 15), an adversary is required to first choose r messages $m_1, m_2, \ldots, m_r$, after which they are provided with key k of H_k and allowed to select a message m_{r+1} and evaluate $H_k(m_{r+1})$. A successful non-adaptive chosen message attack happens when the adversary is able to find C_κ^r, i.e., find a message m_{r+1} that is in an r-subset cover relation with $m_1, m_2, \ldots, m_r$. This scenario corresponds to an attacker who is trying to forge a signature after observing all r allowed signatures per key, or an adversary who is allowed r queries at a time before being supplied with k to

verify any of the returned signatures. Few-time signature schemes are expected to maintain their security against forgery attacks even after releasing all r signatures.

Finding C_κ^r in FORS. Given an adversary who observed the signatures of r messages, finding a message m_{r+1} that is in an $r-$subset cover relation with the other r messages ($C_\kappa^{r\text{-FORS}}(m_1, m_2, \ldots, m_{r+1})$) has probability of success $(r/t)^\kappa$ [16], which is equal to the probability that each $\log t$-bit element out of the κ elements in $ORS(m_{r+1})$ is covered by an element at the same position of the ORSs of the other r messages, i.e., $h_i(m_{r+1}) \in \bigcup_{j=1}^{r} h_i(m_j)$ for $0 \leq i \leq \kappa - 1$, where $h_i(m_j)$ denotes the i-th ORS element of the j-th message. Accordingly, the corresponding bit security against non-adaptive chosen message attacks is given by

$$\log_2(t/r)^\kappa = \kappa(\log_2 t - \log_2 r).$$

3.2.2 Adaptive Chosen Message Attack Against FORS

In this setting, an adversary is given the hash key k and allowed to evaluate H_k for any messages of their choice before selecting $r + 1$ messages. This attack also indicates the r-subset resilience of the utilized hash function (see Def. 14). The definition of adaptive chosen message attack is given in Appendix A.2. Applying the same analysis to FORS, given the key k of H_k, an adversary $\mathcal{A}$ generates the $ORSs$ of $q > r$ messages offline, where $H_k(m_i) = h_0||h_1||\ldots||h_{\kappa-1}$ and $ORS(m_i) = \{h_0, h_1, \ldots, h_{\kappa-1}\}$, for $0 \leq i \leq q - 1$. $\mathcal{A}$ searches for all possible combinations of $(r+1)$ message sets from the set of q messages. For any given $r + 1$ messages combination, the probability that message m_{r+1} is covered by the remaining r messages (i.e., $C_\kappa^{r\text{-FORS}}(m_1, m_2, \ldots, m_{r+1})$), is $(r/t)^\kappa$. Accordingly, $\mathcal{A}$ obtains $\binom{q}{r+1}$ sets of $r + 1$ messages and each set gives $\binom{r+1}{r}$ possible choices for m_{r+1}.

Therefore, the probability of $\mathcal{A}$ successfully generating $C_\kappa^{r\text{-FORS}}$ is bounded from above by

$$\mathsf{Succ}^{C_\kappa^{r\text{FORS}}}(\mathcal{A}) \leq \binom{q}{r+1}\binom{r+1}{r}(r/t)^\kappa,$$

$$\mathsf{Succ}^{C_\kappa^{r\text{-FORS}}} \leq q\binom{q-1}{r}(r/t)^\kappa,$$

$$\mathsf{Succ}^{C_\kappa^{r\text{-FORS}}}(\mathcal{A}) \leq \frac{q.(q-1)\ldots(q-r)}{r!}(r/t)^\kappa.$$

which can be approximated by

$$\mathsf{Succ}^{C_\kappa^{r\text{-FORS}}}(\mathcal{A}) \leq \frac{q^{r+1}}{r!}(r/t)^\kappa.$$

Assuming a success probability close to 1, the above equation can be expressed as

$$(r+1)\log_2 q - \log_2 r! + \kappa(\log_2 r - \log_2 t) = 0.$$

Then the bit security of FORS with respect to adaptive chosen message attacks is given by

$$\frac{\kappa}{r+1}(\log_2 t - \log_2 r) + \frac{\log_2 r!}{r+1}.$$

One may conclude that due to the offline adversarial advantage given to $\mathcal{A}$ (i.e., knowledge of k implies the feasibility of evaluating ORSs for more than r messages of their choice), FORS bit security against adaptive chosen message attacks decreases by a factor of $(r+1)$ when compared to the non-adaptive setting. Note that, currently there is no attack against SPHINCS$^+$ that can utilize the offline adversarial privileges and produce $r+1$ messages in an r-subset cover relation. This is because SPHINCS$^+$ uses a fixed pseudorandom

generation of the key k to get the obtained random subset $ORS_\kappa(H_k(m))$. We also note that k is message dependent and is sent in the clear with each signature so verification takes place. Accordingly, in the event of attacks on the process by which k is evaluated from m, a dramatic decrease in the security of SPHINCS$^+$ will follow. Consequently, in the following section we present a technique that is robust against adaptive chosen message attacks on FORS. Our mechanism eliminates the adversarial offline advantages associated with knowing the hash key k.

3.3 Dynamic Forest of Random Subsets (DFORS)

In this section we present Dynamic Forest Of Random Subsets DFORS, a new HORS-variant that mitigates the offline advantage of an adversary which leads to the adaptive chosen message attack on FORS (discussed in Section 3.2). The main feature of DFORS is that the generation of the ORS is performed concurrently with signing such that each signature element is utilized to generate the next element of the ORS. In other words, signing and ORS generation are bound together using a chaining mechanism that utilizes the revealed secret keys. This procedure ensures that given a message, only the signer is able to efficiently generate an ORS. By doing so, even if an adversary has knowledge of k, they are not able to compute ORSs of a given message of their choice unless they have some secret key knowledge. In what follows we give a detailed specification of DFORS.

3.3.1 DFORS Parameters

DFORS uses the following parameters.

n : The security parameter and the bit-length of (i) the secret seed $SK.seed$, (ii) secret keys $sk_{i,j}$ ($0 \leq i \leq t-1, 0 \leq j \leq \kappa-1$), (iii) public key $PK.root$, and (iv) the output

of the used one-way function F, and hash function G.

κ : The number of (i) sub-strings of the input message, (ii) secret key pools where each contains t secret keys, and (iii) hash trees.

τ : The bit length of a sub-string of the input message and the hash tree height.

t : the number of secret keys per pool and the number of leaves in each hash tree, $t = 2^{\tau}$.

The input message for DFORS is of length $\kappa \log t = \kappa\tau$ bits. To achieve n-bit security when signing r messages, we have $\kappa\tau > n$ (see Section 3.4.1).

3.3.2 Key Generation

In what follows, we give the specifications of the secret and public key generation procedures. Moreover, DFORS is described in Algorithm 3.2.

Secret key generation. Let $SK.seed$ denote an n-bit secret seed that is sampled at random. Given a pseudorandom function, $PRF : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$, the n-bit κt secret key values $sk_{i,j}$, $0 \leq i \leq t-1$, $0 \leq j \leq \kappa - 1$ are generated by

$$sk_{i,j} = PRF(SK.seed, i + jt),$$

where each set of t secret keys belong to one of the κ pools.

Hash trees and public key generation. Using the one-way function $F : \{0,1\}^n \rightarrow \{0,1\}^n$ on the secret keys $sk_{i,j}$, $0 \leq i \leq t-1$, $0 \leq j \leq \kappa - 1$, the leaf nodes of the κ hash trees are generated, $L_{i,j} = F(sk_{i,j})$. Every t leaves, $L_{*,j}$, are combined together in a Merkle tree construction to form the j-th (out of κ) tree. Then, the roots of these κ trees,

$root_0, root_1, \ldots, root_{\kappa-1}$, are concatenated to form an input to the hash function to get the n-bit public key expressed as

$$PK.root = G_k(root_0||root_1||\ldots||root_{\kappa-1}).$$

Binary Hash Tree. DFORS uses the XMSS binary Merkle tree construction [11]. The height of the binary hash tree is τ. It has $\tau + 1$ levels, $t = 2^\tau$ leaf nodes (each of size n bits) on level 0, i.e., $L_i, 0 \leq i \leq t-1$, and an n-bit root node on level τ. We denote the nodes in level j by $N_{i,j}$ where $0 \leq i < 2^{\tau-j}$, $0 \leq j \leq \tau$ and $N_{i,0} = L_i$. To construct the tree, the hash function G and a $2n$-bit mask, q, per hash evaluation are used. These bit masks are introduced to provide second-preimage resistance. The rationale for using different bit masks for each hash evaluation is to mitigate multi-target attacks [14]. For details on generating the hash keys $K_{i,j}$ and bit masks $q_{i,j}$, the reader is referred to [9, 14]. Formally, for $0 < j \leq \tau$, a node $N_{i,j}$ is given by

$$N_{i,j} = G_{k_{i,j}}((N_{2i,j-1}||N_{2i+1,j-1}) \oplus q_{i,j}).$$

Figure 3.2 shows a simplified example of one of the κ trees in DFORS with $t = 8$. Assuming it is the j-th tree, it depicts the nodes in the authentication path (colored in gray) associated with revealing $sk_{3,j}$.

3.3.3 Signing and ORS Generation

We denote by $Z(h)$ a function that takes as input $\kappa\tau$ bits, h, and outputs the j-th τ bits of h, where $j = h \bmod \kappa$. Formally, $Z : \{0,1\}^{\kappa\tau} \rightarrow \{0,1\}^\tau$, and letting $h = h_0||h_1||\ldots||h_{\kappa-1}$,

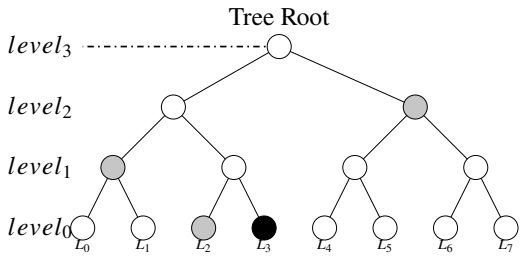


Figure 3.2: A binary hash tree with the nodes in the authentication path (colored in gray) for leaf node L_3 (colored in black)

for $0 \leq j \leq \kappa - 1$

$$Z(h) : h_j \leftarrow \{h_0||h_1|| \ldots ||h_{\kappa-1}\}, j = h \bmod \kappa.$$

The signing algorithm takes as input the message m, the secret seed $SK.seed$, and the hash key k. It constructs the κ trees as explained above in Section 3.3.2. To compute the κ random subset $ORS_\kappa(m) = (b_0, b_1, \ldots, b_{\kappa-1})$, the algorithm first evaluates $H_k(m) = h^0$, then computes $Z(h^0) = b_0$. The first element in the signature, sig_0, is comprised of i) the secret key of index b_0 in the first pool, $\sigma_0 = sk_{b_0,0}$, and ii) the corresponding authentication path $Auth_0$, thus $sig_0 = \sigma_0, Auth_0$. Next, h^0 and $sk_{b_0,0}$ are used to choose the second random element, $Z(h^1) = b_1$, where $h^1 = H_{sk_{b_0,0}}(h^0||h^0)$. The second signature element, sig_1, is the secret key of index b_1 in the second pool, $\sigma_1 = sk_{b_1,1}$, and its corresponding authentication path $Auth_1$, $sig_1 = \sigma_1, Auth_1$. In general, the i-th element of the $ORS_\kappa(m)$ is given by $Z(h^i) = b_i$ where $h^i = H_{sk_{b_{i-1},i-1}}(h^0||h^{i-1})$. The i-th signature element, sig_i, is the secret key value of index b_i in the i-th pool and its corresponding authentication path $Auth_i$, $sig_i = \sigma_i, Auth_i$, where $\sigma_i = sk_{b_i,i}$. The above process is repeated until κ elements are generated $(b_0, b_1, \ldots, b_{\kappa-1})$. Finally, the signature is given by

$$\Sigma = (sig_0, sig_1, \ldots, sig_{\kappa-1}) = (sk_{b_0}, Auth_0, sk_{b_1}, Auth_1, \ldots, sk_{b_{\kappa-1}}, Auth_{\kappa-1})$$

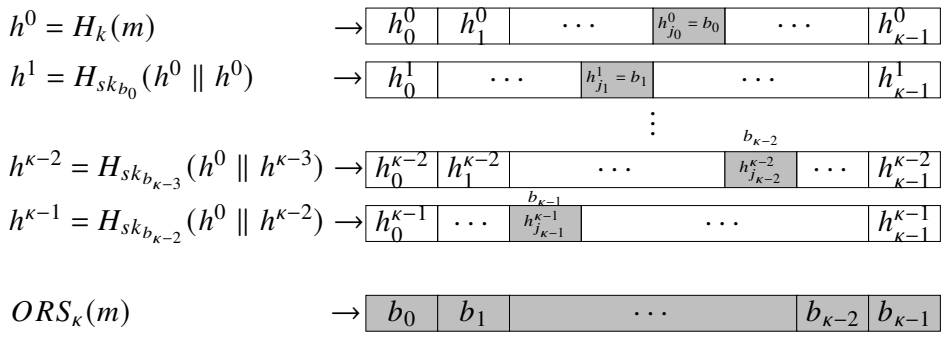


Figure 3.3: The DFORS procedure to compute $ORS_\kappa(m)$, where $j_i = h^i \bmod \kappa$, $b_i = h^i_{j_i}$, and sk_{b_i} is the b_i-th secret key in the i-th secret key pool.

$$= (\sigma_0, Auth_0, \sigma_1, Auth_1, \ldots, \sigma_{\kappa-1}, Auth_{\kappa-1}).$$

The ORS generation and signing process is illustrated in Figure 3.3. The authentication path of a leaf L_i contains all the sibling nodes of the nodes in the path from the leaf L_i to the tree root. It is required so that the verifier can successfully generate the root in order to verify the signature element σ_i related to the leaf node L_i. Figure 3.2 shows a simple hash tree with the authentication path for leaf L_3 colored in black and the authentication path nodes colored in gray, $Auth_i = (L_2, N_{0,1}, N_{1,2})$.

3.3.4 Signature Verification

The verification algorithm takes as input the message m, the public key $PK.root$, the hash key K, and the signature $\Sigma = (\sigma_0, Auth_0, \sigma_1, Auth_1, \ldots, \sigma_{\kappa-1}, Auth_{\kappa-1})$. It computes $H_k(m) = h^0$, then $Z(h^0) = b_0$ to get the leaf index of the first hash tree. Then, it applies the one-way function F to the signature element σ_0 of the signature Σ to get the leaf node L_{b_0} in the first tree. The authentication path $Auth_0$ and the leaf L_{b_0} are used to compute the root of the first tree. The leaf index b_0 is required so that the verifier knows which node is concatenated on the right and on the left. The tree root calculation procedure is described in Algorithm 3.1. Generally, the verification algorithm computes the i-th tree

root by applying Algorithm 3.1 on σ_i, $Auth_i$, and the leaf index b_i where $b_i = Z(h^i)$, and $h^i = H_{\sigma_{i-1}}(h^0||h^{i-1})$. This process is repeated until κ tree roots are computed which are then concatenated to form an input to the hash function G. If the output of G is equal to $PK.root$, the signature is valid, otherwise verification fails.

Algorithm 3.1 Tree Root Computation

Input: Leaf node L_i, Leaf index i, Auth. Path = $(A_0, A_1, \ldots, A_{\tau-1})$.
Output: The Tree Root N_τ.

Set $N_0 \leftarrow L_i$
for $1 \leq j \leq \tau$ **do**
  if $\lfloor i/2^{j-1} \rfloor \equiv 0 \bmod 2$ **then**
    $N_j = G_{k_{i,j}}(N_{j-1}||A_{j-1} \oplus q_{i,j})$
  else
    $N_j = G_{k_{i,j}}(A_{j-1}||N_{j-1} \oplus q_{i,j})$
  end if
end for
Return (N_τ)

Algorithm 3.2 DFORS Algorithm

procedure KEY GENERATION(t, κ)
  $SK.seed \xleftarrow{R} \{0,1\}^n$
  for $0 \leq j \leq \kappa - 1$ **do**
    for $0 \leq i \leq t - 1$ **do**
      $sk_{i,j} \leftarrow PRF(SK.seed, i + jt)$
      $L_{i,j} \leftarrow F(sk_{i,j})$
    end for
  end for
  Compute the roots of the κ tree as described in section 3.3.2
  $PK.root \leftarrow G(root_0||root_1||\ldots||root_{\kappa-1})$
  Output $(SK.seed, PK.root)$
end procedure

---

procedure SIGNING(m, $SK.seed$,k, κ, t)
  Generate the κ binary hash trees as in key generation procedure
  $h^0 \leftarrow H_k(m)$, $h^0 = h^0_0||h^0_1||\ldots||h^0_{\kappa-1}$
  $b_0 \leftarrow Z(h^0) = h^0_{j_0}$, $j_0 = h^0 \ mod \ \kappa$
  $sig_0 \leftarrow (\sigma_0, Auth_0)$, Where $\sigma_0 = sk_{b_0,0}$
  for $1 \leq i \leq \kappa - 1$ **do**
    $h^i \leftarrow H_{sk_{b_{i-1},i-1}}(h^0||h^{i-1})$, $h^i = h^i_0||h^i_1||\ldots||h^i_{\kappa-1}$
    $b_i \leftarrow Z(h^i) = h^i_{j_i}$, $j_i = h^i \ mod \ \kappa$
    $sig_i \leftarrow (\sigma_i, Auth_i)$, where $\sigma_i = sk_{b_i,i}$
  end for
  $\Sigma \leftarrow (\sigma_0, Auth_0, \sigma_1, Auth_1, \ldots, \sigma_{\kappa-1}, Auth_{\kappa-1})$
  Output (Σ,m)
end procedure

---

procedure VERIFICATION($m, PK.root, k, \Sigma = (\sigma_0, Auth_0, \sigma_1, Auth_1, \ldots, \sigma_{\kappa-1}, Auth_{\kappa-1})$)
  $h^0 \leftarrow H_k(m)$, $h^0 = h^0_0||h^0_1||\ldots||h^0_{\kappa-1}$
  $b_0 \leftarrow Z(h^0) = h^0_{j_0}$, $j_0 = h^0 \ mod \ \kappa$
  $L_{b_0} \leftarrow F(\sigma_0)$
  $root_0 \leftarrow$ Algorithm 3.1 $(L_{b_0,0}, b_0, Auth_0)$
  for $1 \leq i \leq \kappa - 1$ **do**
    $h^i \leftarrow H_{\sigma_{i-1}}(h^0||h^{i-1})$, $h^i = h^i_0||h^i_1||\ldots||h^i_{\kappa-1}$
    $b_i \leftarrow Z(h^i) = h^i_{j_i}$, $j_i = h^i \ mod \ \kappa$
    $L_{b_i} \leftarrow F(\sigma_i)$
    $root_i \leftarrow$ Algorithm 3.1 $(L_{b_i,i}, b_i, Auth_i)$
  end for
  if $G(root_0||root_1||\ldots||root_{\kappa-1}) = PK.root$ **then**
    $out = 1$
  else
    $out = 0$
  end if
  Output (out)
end procedure

3.4 DFORS Security and Efficiency

In what follows, we analyze the security of DFORS and demonstrate the effect of the dynamic chaining on the security of FORS. Afterwards, the computational cost of the DFORS key generation, signing, and verification algorithms are presented. The bit size of the signature and keys are also given.

3.4.1 DFORS Security Analysis

In this section, we present a detailed analysis of DFORS with respect to weak-message attacks and r-target subset resilience adversaries. More precisely, since the proposed chaining technique does not allow an adaptive adversary who has knowledge of k to compute the ORSs of any message of their choice before asking the signing oracle for its signature, DFORS is essentially r-subset resilient. Hence, our analysis focuses on its security when an adversary is given the signatures of r messages.

Weak-message attacks. DFORS inherits FORS mitigation to weak-message attacks [16] because it specifies an independent key pool for each index in the ORS. Consequently, even if an ORS element is repeated, the corresponding revealed secret keys will be different.

r-target subset resilience. According to Definition 15, we assume an adversary $\mathcal{A}$ when given the ORSs of r messages will return m_{r+1} where $C^r_\kappa(m_1, m_2, \ldots, m_{r+1})$. In what follows, we show that the success probability of $\mathcal{A}$ is bounded from above by $(r/t)^\kappa$. Note that since ORS generation is secret key dependent, the ORS function of DFORS is intrinsically r-subset resilient. In other words, the value of any random ORS element, b_i, depends on the previously revealed signature element $\sigma_{i-1} = sk_{b_{i-1}}$ and the original message m. Accordingly, without any oracle queries, $\mathcal{A}$ has no feasible function to evaluate ORSs

of messages of their choice. On the other hand, if $\mathcal{A}$ is given the signatures of r messages or they queried r messages of their choice, they need to find a message m_{r+1} such that each element in its obtained random subset, $ORS_{\kappa}^{\mathsf{DFORS}}(m_{r+1}) = (b_0, b_1, \ldots, b_{\kappa-1})$, is covered by the elements at the same corresponding positions in the ORSs of the other r messages

$$C_{\kappa}^{r}(m_1, m_2, \ldots, m_{r+1}) \Leftrightarrow b_i(m_{r+1}) \in \bigcup_{j=1}^{r} b_i(m_j), 0 \leq i \leq \kappa - 1.$$

Due to the chaining process in generating $b_0, b_1, \ldots, b_{\kappa-1}$, $\mathcal{A}$ generates the ORSs sequentially. At any position i, if $b_i(m_{r+1}) \notin \cup_{j=1}^{r} b_i(m_j)$, then $\mathcal{A}$ fails. In addition, they cannot evaluate $b_{i+1} = Z(H_{sk_{b_i}}(h^0||h^i))$ when sk_{b_i} is not revealed by any of signatures of the r messages, Generally, for the i-th position in $ORS_{\kappa}^{\mathsf{DFORS}}(m_{r+1})$

$$b_i(m_{r+1}) \notin \bigcup_{j=1}^{r} b_i(m_j) \Rightarrow sk_{b_i} \notin \bigcup_{j=1}^{r} \sigma_i(m_j),$$

where $\sigma_i(m_j)$ and $b_i(m_j)$ denote the i-th signature element and i-th ORS element of the j-th message, respectively. Thus, the probability that $\mathcal{A}$ finds $C_{\kappa}^{r}(m_1, m_2, \ldots, m_{r+1})$ successfully is equal to their probability of finding a message m_{r+1} such that $\forall i \in \{0, 1, \ldots, \kappa - 1\}$, each of the $\log t$-bit $b_i(m_{r+1}) \in \{b_i(m_1), b_i(m_2), \ldots, b_i(m_r)\}$. Since $\mathcal{A}$ is given r messages, the probability of finding a cover for one $b_i(m_{r+1})$ is $(r/t)^{i+1}$ because this implies that $\forall j < i; b_j(m_{r+1}) \in \{b_j(m_1), b_j(m_2), \ldots, b_j(m_r)\}$. Thus, the probability of finding a cover for all the κ elements in $ORS_{\kappa}^{\mathsf{DFORS}}$ is equal to the probability of finding a cover for the last element, $b_{\kappa-1}(m_{r+1})$, which is $(r/t)^{\kappa}$. Therefore

$$\mathsf{Succ}^{C_{\kappa}^{r}\text{-}\mathsf{DFORS}}(\mathcal{A}) \leq (r/t)^{\kappa},$$

so the corresponding DFORS bit security against adaptive chosen message attacks is

$$\log_2(t/r)^\kappa = \kappa(\log_2 t - \log_2 r).$$

Compared to the adaptive chosen message attack security of FORS (See Section 3.2), the bit security of DFORS is higher by a factor of $(r+1)$. The extra cost is performing $\kappa - 1$ more calls to the hash function. Unlike FORS, the signing procedure cannot be parallelized because of the chaining mechanism.

3.4.2 Theoretical Efficiency

Key generation. This procedure requires κt PRF function computations to generate the t secret values for κ pools, κt one-way function F computations to compute the leaf nodes of the hash trees, and $\kappa(t-1)+1$ hash function G evaluations to evaluate the κ hash trees and get the public key $PK.root$.

Signing. This procedure requires κt PRF function computations, κt one-way function F computations, κt hash function (H and G) to compute the κ hash trees ($\kappa(t-1)$ hash G calls), and κ hash H calls to get $ORS_\kappa(m)$. Note that the whole tree structure is computed with each signature, otherwise, the scheme storage requirements will be huge.

Verification. This procedure requires κ one-way function F computations that compute the trees leaves, $\kappa(\tau+1)$ hash function (H and G) evaluations to reconstruct the κ trees roots from the revealed secret values and the authentication paths ($\kappa\tau$ calls to G), and κ calls H to get $ORS_\kappa(m)$.

Signature size. The signature contains κ secret key elements and $\kappa\tau$ tree node for the associated authentication paths. Thus, the signature size is $\kappa n(\tau+1)$ bits, where n is the

bit size of each secret keys and hash tree node.

Length of keys. The size of the secret key, $SK.root$, is equal to that of the public key, $PK.root$, and it is n bits.

The computational complexities of the above procedures are given in Table 3.2.

3.4.3 Comparison with HORS Variants

DFORS inherits all the advantageous security properties of FORS. Additionally, it is secure against adaptive chosen message attacks. In fact, for the same parameters the bit security of DFORS with respect to adaptive chosen message adversaries is equal to that of FORS under non-adaptive chosen message attacks. Table 3.1 gives a comparison between the bit security level of FORS and DFORS in an adaptive adversarial setting. We use the recommended parameters (i.e., n, τ, and κ) for all six instances of SPHINCS$^+$.

Table 3.1: DFORS and FORS security levels for an adaptive chosen message attack using the SPHINCS$^+$parameters for different numbers of signed messages

SPHINCS$^+$ instance	τ	κ	FORS				DFORS			
			$r=1$	$r=2$	$r=4$	$r=8$	$r=1$	$r=2$	$r=4$	$r=8$
SPHINCS$^+$-128s	15	10	75	47	27	15	150	140	130	120
SPHINCS$^+$-128f	9	30	135	80	43	22	270	240	210	180
SPHINCS$^+$-192s	16	14	112	70	40	22	224	210	196	182
SPHINCS$^+$-192f	8	33	132	77	41	20	264	231	198	165
SPHINCS$^+$-256s	14	22	154	95	54	29	308	286	264	242
SPHINCS$^+$-256f	10	30	150	90	49	25	300	270	240	210

Table 3.1 shows the significant effect of increasing the number of signed messages, r, on the bit security of FORS. On the other hand, this effect is very reasonable with DFORS. For instance, when $r = 1$, an adaptive attack on FORS is equivalent to a collision attack on the underlying $\kappa\tau$-bit hash function H which has a complexity of $2^{\kappa\tau/2}$ evaluations. However, due to the r-subset resilience of DFORS where finding a covered ORS requires successive

dependency on the signature elements, an adversary must find a second preimage of the ORS in the revealed secret keys, hence the complexity is $2^{\kappa\tau}$ evaluations.

Table 3.2 presents a comparison between DFORSand other HORS variants with respect to their computational efficiency, signature and key sizes, and security against adaptive chosen message attacks.

Table 3.2: Comparison between HORS, PORS, FORS, and DFORS

Algorithm	**KGen (# OWF)**†	**Signing cost**	**Verification cost**	**Signature size** ‡	**SK/PK size**‡	**Adaptive security**
HORST	t PRF t OWF $t-1$ Hash	t PRF t OWF t Hash	κ OWF $\kappa(\log t - x) + 2^{x\dagger\dagger}$ Hash	$\kappa(\log t - x + 1) + 2^{x\dagger\dagger}$	1	NO
PORS‡‡	t PRF t OWF $t-1$ Hash	$t+\kappa$ PRF t OWF t Hash	κ OWF $\kappa(\log t - x - 1) + 2^{x\dagger\dagger}$ Hash	$\kappa(\log t - \lfloor \log \kappa \rfloor + 1)$	1	NO
FORS	κt PRF κt OWF $\kappa(t-1)+1$ Hash	κt PRF κt OWF $\kappa(t-1)+1$ Hash	κ OWF $\kappa \log t + 1$ Hash	$\kappa(\log t + 1)$	1	NO
DFORS	κt PRF κt OWF $\kappa(t-1)+1$ Hash	κt PRF κt OWF κt Hash	κ OWF $\kappa(\log t + 1)$ Hash	$\kappa(\log t + 1)$	1	YES

† OWF denotes one-way function.
‡ Size is given as a factor of n bits.
†† $x = \lceil \log \kappa \rceil$ for optimal signature size in case of HORST and for the upper bound on the signature size in PORS.
‡‡ Verification cost and signature size are the upper bound values.

3.5 Conclusion

In this cahpter we analyzed the security of FORS, the underlying hash-based few-time signing scheme of SPHINCS$^+$ with respect to adaptive chosen message attacks. We showed that as the number of signed messages, r, increases, its bit security with respect to adaptive chosen message adversaries decreases significantly compared to its non-adaptive counterpart. As a solution, we proposed DFORS, which builds on FORS but utilizes a secret key dependent ORS function. Such a function binds the process of generating the ORS with signing which makes it feasible only for the signer. Accordingly, we showed that the bit

security of DFORS against adaptive chosen message attacks is more than that of FORS by a factor of $r + 1$. Note that our analysis does not affect the claimed security of SPHINCS$^+$ but rather provides a better understanding of the security of its underlying signing scheme and offers a mechanism that can be adopted by most HORS variants to provide security against adaptive chosen message attacks.

Chapter 4

Verifiable ORS Generation: Improving The Performance of SPHINCS$^+$

In this chapter, we propose a *Verifiable* Obtain Random Subsets (v-ORS) generation mechanism which with one extra hash computation binds the message with the signing FORS instance (the underlying few-time signature algorithm). This enables SPHINCS$^+$ to offer more security against generic attacks because the proposed modification restricts the ORS generation to use a hash key from the utilized signing FORS instance. Consequently, such a modification enables the exploration of different parameter sets for FORS to achieve better performance at the same security level. For instance, when using v-ORS, one parameter set for SPHINCS$^+$-256s provides 82.9% reduction in the computation cost of FORS which leads to around 27% reduction in the number of hash calls of the signing procedure. Given that NIST has identified the performance of SPHINCS$^+$ as its main drawback, these results are a step forward in the path to standardization. The contributions of this chapter were published in [23].

4.1 Specifications of SPHINCS$^+$

In this section, we give a brief description of SPHINCS$^+$ which consists of the following three types of trees. (i) The hyper-tree is the main tree for the whole construction. It has height h and contains d layers of subtrees, numbered 0 to $d-1$, where each subtree is of height h/d. The root of the top layer subtree (layer $d-1$) is part of the SPHINCS$^+$ public key. (ii) The subtrees are the Merkle trees that build the hyper-tree. These subtrees adopt the XMSS-T construction [14]. Their leaf nodes are the public keys of WOTS$^+$. The corresponding secret keys of each leaf node are used to sign the root of the subtree at the lower layer. Note that since these roots are fixed, a given WOTS$^+$ leaf node always signs the same value. In any layer, j, there are $2^{(d-1-j)(h/d)}$ subtrees where $0 \leq j \leq d-1$. (iii) FORS instances correspond to the 2^h leaf nodes of the hyper-tree. Each FORS instance contains κ trees, each of τ levels and 2^τ leaves which contain secret keys that are used to sign the message. A FORS instance root is the hash of the concatenation of its κ trees Merkle roots, and is signed by a WOTS$^+$ leaf from the corresponding subtree at layer 0. Figure 4.1 gives a simplified depiction of the SPHINCS$^+$ construction where the FORS trees and subtrees have 3 levels. In this figure, the message digest is signed by a FORS instance at the bottom layer whose root is coloured in red. Such a root is in turn signed using the WOTS$^+$ leaf node, coloured green, in the corresponding subtree at layer 0. The authentication paths are coloured in gray and the roots of the used subtrees are coloured in yellow, which are similarly iteratively signed by intermediate WOTS$^+$ nodes until the root of the top subtree is reached. The top subtree root is the public key of SPHINCS$^+$.

4.1.1 SPHINCS$^+$ Parameters

SPHINCS$^+$ has the following parameters:

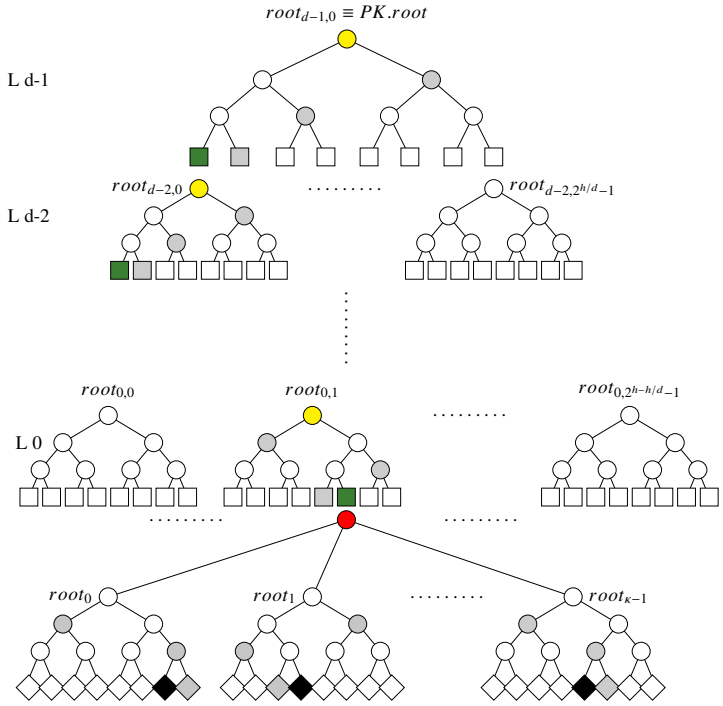


Figure 4.1: Simplified SPHINCS$^+$ depiction where the FORS trees and subtrees are 3 levels high. The diamond, circle, and square nodes denote FORS leaves, intermediate hash nodes, and WOTS$^+$ leaves, respectively.

- h is the total height of the SPHINCS$^+$ hyper-tree and the bit-length of the FORS instance index.

- d is the number of tree layers.

- κ is the number of (i) sub-strings, correspondingly, the number of the ORS elements, in the message digest and (ii) hash trees in a FORS instance where each tree has t secret keys.

- τ is the bit length of a sub-string of the message digest and the FORS hash tree height.

- t is the number of secret keys corresponding to the leaves in each tree in a FORS instance, $t = 2^{\tau}$.

- w is the Winternitz parameter of WOTS$^+$.

- n is the security parameter and it is the bit-length of (i) the secret seed, $SK.seed$, and the secret pseudorandom number *SK.prf*, (ii) FORS secret keys, $SK_{i,j,z}$ ($0 \leq i \leq 2^h - 1$, $0 \leq j \leq \kappa - 1, 0 \leq z \leq t - 1$), (iii) the public key, $PK.root$, and the public seed *PK.seed*, (iv) the output of the one-way function, F, hash function, H, and tweakable hash Th, and (v) the hash randomizer, R.

4.1.2 Tweakable Hash Function (*Th*)

The notion of tweakable hash functions was introduced in [9,16]. It takes as inputs a public parameter $PK.seed$, a tweakable (variable) value, $ADRS$, of length T bits, and the input message. Note that in SPHINCS$^+$ the tweak value is the index of a specific hash call in the algorithm which is calculated by an addressing scheme, hence making the hash function calls at different positions in the algorithm independent (for the details of SPHINCS$^+$ Hash addressing scheme, see Appendix A.3). Formally, Th is defined as follows

$$Th(Pk.seed, ADRS, M) : \{0,1\}^n \times \{0,1\}^T \times \{0,1\}^* \rightarrow \{0,1\}^m$$

A tweakable hash maps an arbitrary length message M to a message digest of m bits using a public parameter $PK.seed$ of n bits and the hash address ADRS of T bits. For the coherence with previous works on hash-based digital signature schemes, the function F (resp. H) is denoted by Th_1 (resp. Th_2), formally,

$$Th_1 \stackrel{\text{def}}{=} F : \{0,1\}^n \times \{0,1\}^{256} \times \{0,1\}^n \rightarrow \{0,1\}^n$$

$$Th_2 \stackrel{\text{def}}{=} H : \{0,1\}^n \times \{0,1\}^{256} \times \{0,1\}^{2n} \rightarrow \{0,1\}^n$$

Note that the hash address length in SPHINCS$^+$ is 256 bits, and F (resp. H) maps an n-bit (resp. $2n$-bit) input to an n-bit output. The hash function H_{msg} which is used to calculate the message digest using an n-bit hash randomizer R is given by

$$Th_{msg} \stackrel{\text{def}}{=} H_{msg} : \{0,1\}^n \times \{0,1\}^{2n} \times \{0,1\}^* \rightarrow \{0,1\}^m$$

4.1.3 Generation of Keys

In what follows, we give the specifications of the secret and public keys generation procedures. Let *SK.seed, SK.prf, PK.seed* be sampled at random, where $SK.seed$ is used to generate the secret keys of both FORS and WOTS$^+$, and $PK.seed$ is used in the addressing scheme. Both seeds are used to construct all the trees and evaluate the root of the top layer, $PK.root$. *SK.prf* is used to compute the hash randomizer during signing. The SPHINCS$^+$ secret key is given by *SK=(SK.seed, SK.prf, PK.seed, PK.root)* and the corresponding public key is given by *PK= (PK.seed, PK.root)*.

4.1.4 Signing Algorithm

The signing algorithm defines the ORS generation and the message signing procedures.

ORS generation. This procedure takes a message M, *SK.prf*, and PK as inputs, and outputs the index of the FORS instance that will be used in the signing procedure and the indexes of the secret keys (ORS elements) which are revealed from that instance in the signature. More precisely, using a pseudorandom key generation function PRF, the hash randomizer R is calculated as

$$R = PRF_{msg}(SK.prf, OptRand, M) \tag{4.1}$$

where OptRand is a 256 bit value which by default is set to 0 and can be any random value

to prevent deterministic signing. An h-bit $indx$ of the FORS instance that is used to sign the message, and a $\kappa\tau$-bit message digest, $md = b_0||b_1||\ldots||b_{\kappa-1}$ are evaluated using H_{msg} with R as a hash randomizer as follows

$$md||indx = H_{msg}(R, PK, M), \tag{4.2}$$

The ORS is the set of κ substrings $(b_0, b_1, \ldots, b_{\kappa-1})$, each of length τ bits.

Message signing. The FORS signature contains the set of σ_i which is the b_i-th secret key leaf from the i-th FORS tree of the indexed I-th FORS instance, i.e., SK_{I,i,b_i}, and its corresponding authentication path $Auth_i$, $0 \leq i \leq \kappa - 1$

$$SIG_{FORS} = (\sigma_0, Auth_0), (\sigma_1, Auth_1), \ldots (\sigma_{\kappa-1}, Auth_{\kappa-1}) \tag{4.3}$$

The κ roots of the trees in the FORS instance are concatenated and hashed to get an n-bit FORS root

$$FORS.root = Th(PK.seed||ADRS_I||root_o||root_1||\ldots||root_{\kappa-1}) \tag{4.4}$$

$FORS.root$ is then signed using the WOTS$^+$ of the corresponding leaf node in the corresponding subtree at layer 0 to get the WOTS$^+$ signature (σ_{W_0}), and its authentication path $Auth_{W_0}$ of h/d hash nodes, i.e., $SIG(FORS.root) = \sigma_{W_0}, Auth_{W_0}$. Then the root of this subtree at layer 0 is signed using the WOTS$^+$ at the corresponding subtree at layer 1. This process is iterated until the top layer is reached, i.e., for $0 \leq i \leq d - 1$, $SIG(tree.root_{i-1}) = \sigma_{W_i}, Auth_{W_i}$.

The signature, Σ, contains the randomizer R, the FORS signature, and d WOTS$^+$

signatures with their authentication paths

$$\Sigma = (R, SIG_{FORS}, (\sigma_{W_0}, Auth_{W_0}), \ldots, (\sigma_{W_{d-1}}, Auth_{W_{d-1}})) \tag{4.5}$$

4.1.5 Signature Verification

The verification algorithm operates on the message, M, the signature, Σ, and the public key, $PK = (PK.root, PK.seed)$. It starts by computing $H_{msg}(R, PK, M) = indx||md$ to index the FORS instance and the d indices of WOTS$^+$ leaf nodes that are used to sign the subtree roots. The message digest, $md = b_0, b_1, \ldots, b_{\kappa-1}$, defines the indices of the revealed FORS secret keys from each FORS tree. Then the FORS signature, SIG_{FORS} is used to compute the FORS root as follows. The one-way function F is applied to the revealed secret keys, SK_{I,i,b_i}, $0 \leq i \leq \kappa - 1$, to get the leaf nodes L_{I,i,b_i} which along with the corresponding authentication path $Auth_i$ are used to compute the i-th FORS tree root. The κ FORS tree roots are concatenated and hashed to get the FORS root as defined in Equation 4.4. Finally, each subtree/FORS root with the corresponding signature is used to calculate the root of the subtree until the root of the top layer is reached and compared to $PK.root$ to determine the validity of the signature.

4.2 SPHINCS$^+$ with Verifiable ORS

We observe that the randomizer R is sent as part of the signature to be used by the verifier to compute the ORS elements without a means of verifying its correct computation. In other words, consider a forging adversary who is allowed to query the signing oracle with messages of their choice (see Section 2.2 for EU-CMA security). Such an adversary is always free to choose a randomizer that generates ORS elements which collide with the

ORS sets revealed in the previous (queried) signatures without any restriction on the signing FORS instance, i.e., the message digest md and FORS index $indx$ in Equation 4.2 are not bound together. Such a security notion in SPHINCS$^+$ is captured by its ORS function Interleaved Target Subset Resilience (ITSR) (See Definition 16) which requires specific parameterization in terms of the number and height of the FORS trees to reach the claimed bit security. In what follows, we propose a modification to the ORS generation in the SPHINCS$^+$ signing algorithm that binds the message digest md, correspondingly the ORS, with the FORS instance that is used for signing. Our modification restricts the freedom of the adversary when attempting the previous attack steps, hence, increasing the ITSR bit security of the modified ORS function. Consequently, we are able to offer efficient parameter sets for the underlying FORS scheme to enhance the performance of SPHINCS$^+$.

Verifiable ORS (v-ORS) Generation. The signer first generates a hash randomizer, R, as given in Equation 4.1. Then R is used as a hash randomizer to calculate the index of the FORS instance used for signing and a secret key index within that same FORS instance. Formally, given $H_1 : \{0,1\}^n \times \{0,1\}^{2n} \times \{0,1\}^* \rightarrow \{0,1\}^n$, we obtain

$$h_{msg} = H_1(R, PK, M), \tag{4.6}$$

Let the first $h + \lceil \log_2 \kappa \rceil + \tau$ bits of h_{msg} an index for a secret key in a FORS tree within a FORS instance. Specifically, the first h bits denote the I-th index for a FORS instance, the following $\lceil \log_2 \kappa \rceil$ bits denotes the J-th index of a FORS tree within the I-th FORS instance, and $0 \leq J \leq \kappa - 1$, and the last τ bits specify the Z-th index of a secret key, $(SK_{I,J,Z})$, within the J-th FORS tree. Note that the bit length of J is $\lceil \log_2 \kappa \rceil$, so if κ is not a power of 2, J is reduced to $J \mod \kappa$. $SK_{I,J,Z}$ is then used as a hash key to compute the message

digest md. Formally, consider $H_2 : \{0,1\}^n \times \{0,1\}^{2n} \times \{0,1\}^{2n} \rightarrow \{0,1\}^{\kappa\tau}$, then

$$md = b_0||b_1||\ldots||b_{\kappa-1} = H_2(SK_{I,J,Z}, PK, R||h_{msg}), \tag{4.7}$$

where b_j indexes a FORS signature secret key from the j-th FORS tree in the I-th FORS instance. Hence, the ORS is given by the set of indexes $\{b_0, b_1, \ldots, b_{\kappa-1}\}$. Note that such an ORS is valid if it can be generated using the hash randomizer, $(SK_{I,J,Z})$, which is sent as part of the signature to the verifier. Hence, the reason for naming the modified ORS generation v-ORS, is that only a legitimate signer can efficiently generate it and this fact is verifiable. We refer to a SPHINCS$^+$ using v-ORS by vSPHINCS$^+$.

Signing and verification in vSPHINCS$^+$. The FORS signature, SIG_{FORS}, is evaluated as in SPHINCS$^+$, see Equation 4.3. However, a vSPHINCS$^+$ signature includes $(SK_{I,J,Z})$ along with its authentication path

$$\Sigma = (R, (\sigma', Auth'), SIG_{FORS}, (\sigma_{W_0}, Auth_{W_0}), \ldots, (\sigma_{W_{d-1}}, Auth_{W_{d-1}})),$$

where σ' is the secret key $SK_{I,J,Z}$ and $Auth'$ is its corresponding authentication path. Note that since the same FORS instance is used in signing, $Auth'$ is generated when the J-th FORS tree is built to evaluate $(\sigma_J, Auth_J)$. In the verification procedure, the signature verifier uses R as a hash randomizer to calculate the FORS index I, FORS tree index J, and the key index Z, from the selected tree from the FORS instance, see Equation 4.6.

During verification, the received signature element σ' is used to generate the message digest md (respectively the ORS), as shown in Equation 4.7. After that, (σ' and $Auth'$) are used to calculate the root of the FORS tree J, and compare it with the root obtained from the FORS signature elements $(\sigma_J, Auth_J)$. If they are different, the signature is invalid, otherwise, the

FORS root is calculated and the same verification process as in SPHINCS$^+$ is performed.

4.2.1 Rationale of Design Choices

Binding the ORS generation with the signing FORS instance restrains the adversary freedom to generate an ORS set which also has to be a valid subset of the ORSs of the queried messages. Precisely, Equation 4.6 in v-ORS restricts choosing the hash randomizer that generates the ORS in Equation 4.7 to a specific FORS secret key, which is infeasible for the adversary to guess unless it was revealed through the queried messages (this event occurs with low probability as given in Equation 4.8).

For evaluating the ORS, i.e., md in Equation 4.7, we initially planned to hash the message itself by applying $H_2(SK_{I,J,Z}, PK, M)$ but we realized that such a decision worsens the signing performance if the message size is large. Specifically, the message is going to be hashed twice; once to generate, h_{msg} in Equation 4.6, which provides the FORS secret key that is used as a hash randomizer. The second time is during the ORS evaluation using H_2. Accordingly, we decided on hashing the message hash output, h_{msg}, in Equation 4.7 by applying $H_2(SK_{I,J,Z}, PK, h_{msg})$. Nevertheless, we found that for a valid forgery, an adversary needs to find a message-randomizer pair (M', R') which outputs $h_{msg} = H_1(R', PK, M')$ where h_{msg} is a second preimage of any of the queried messages. Such an attack is equivalent to breaking the security of multi-target extended target collision resistance M-eTCR of the hash function H_1 of vSPHINCS$^+$ as given in Definition 9. An M-eTCR attack has a success probability of $\frac{qs\cdot(q+1)}{2^n} + \frac{q\cdot qs}{2^n}$ [14], where qs is the number of targets, i.e., the queried messages and q is the computational cost that the adversary needs to query the hashing oracle. In case of an M-eTCR attack on H_1, a forgery is certain because h_{msg} leads to the same $SK_{I,J,Z}$ and consequently same ORS as $ORS = H_2(SK_{I,J,Z}, PK, h_{msg})$.

Consequently, we decided to included the hash randomizer R with h_{msg} as an input to the second hash call $H_2(SK_{I,J,Z}, PK, R||h_{msg})$. In such a case, a valid forgery requires the adversary to find a message M' that outputs $h_{msg} = H_1(R_j, PK, M') = H_1(R_j, PK, M_j)$ were R_j is the hash randomizer used with a message M_j out of the queried messages and $0 \leq j < qs$. Such an attack is equivalent to breaking the security of multi-function multi-target second preimage resistance (MM-SPR) of the hash function H_1 of vSPHINCS$^+$ (see Definition 6) which has a success probability of $\frac{q+1}{2^n}$ [14], where q is the computational cost that the adversary needs to query the hashing oracle. Note that getting an MM-SPR of H_1 leads to $SK_{I,J,Z}$ and an ORS where $ORS = H_2(SK_{I,J,Z}, PK, R||h_{msg})$.

4.2.2 Performance Implications

Compared to SPHINCS$^+$, the signature size is increased by $(\tau + 1) \times n$ bits because $SK_{I,J,Z}$ and its authentication path are included in vSPHINCS$^+$ signatures. Note that different key sets are used for each ORS element to mitigate the weak-message attack [8], which means that the ORS elements are not distinct. Hence, it is not necessary to dedicate an extra FORS tree to choose the key ($SK_{I,J,Z}$) from because it is a single value and even if it has the same index value, Z, as one of the ORS elements, they might come from a different key sets (tree). To counter the effect of increasing the signature size, one can leverage the increase in the security due to the restrictions imposed by ORS generation using v-ORS (See Section 4.3) to explore more efficient parameters for FORS. More precisely, if we can decrease the number of ORS elements by one, then the number of FORS trees is decreased by one, so the signature size is the same as in SPHINCS$^+$. Accordingly, we achieve a better performance by saving the computations required to generate a FORS tree. Various FORS parameter sets are explored in Section 4.5, with some achieving around 27% reduction in the number of hash calls to generate a signature. On top of that, the majority of SPHINCS$^+$

instances when using v-ORS maintain the same signature size while offering reduction in signing computation. For some instances, we obtain better performance and smaller signatures, e.g., for SPHINCS$^+$-192s, v-ORS achieves around 11% reduction in the signing computation with 0.44% decrease in the signature size when compared to SPHINCS$^+$-192s. In what follows, we analyze the interleaved target subset resilience of v-ORS.

4.3 Interleaved Target Subset Resilience of v-ORS

The notion of target subset resilience (TSR) of ORS functions has been used to evaluate the security of HORS and other few-time hash-based signature schemes against (non) adaptive chosen message attacks [6]. For such schemes, an adversary is successful in forging signatures if they are successful in generating a valid ORS for a message when given the ORSs of previously queried messages (See Definition 15). Similarly in SPHINCS$^+$where its security with respect to forgery attacks is reduced to the TSR security of the ORS function of FORS. Following the analysis in [16], to map such a security notion to FORS, which may be viewed as a huge HORS instance with interleaved key sets, we analyze its interleaved target subset resilience. In vSPHINCS$^+$, we may view all the FORS instances as one large FORS instance that consists of 2^h key pools, and each pool contains κ sets of t n-bit keys. The two successive calls to H_1 and H_2 in Equations 4.6 and 4.7 bind and map the message to a specific key pool and generates a set of values, $\{b_j\}_{j=0}^{\kappa-1}$, such that each FORS signature secret element is the b_j-th value in the j-th key set. We define our v-ORS function by

$$H_2 \circ H_1 \stackrel{\text{def}}{=} H_2(SK_{I,J,Z}, PK, R||H_1(R, PK, M)),$$

where each of H_1 and H_2 can be viewed as a composition of a keyed hash function and a mapping function. Formally, let H_1 and H_2 denote two keyed hash functions where

$H_1 : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^n$ and $H_2 : \{0,1\}^k \times \{0,1\}^{2n} \rightarrow \{0,1\}^{md}$. Consider the following two mapping functions, MAP_1 and MAP_2, where $MAP_1 : \{0,1\}^n \rightarrow \{0,1\}^h \times [0, \kappa - 1] \times [0, t - 1]$, and $MAP_2 : \{0,1\}^{md} \rightarrow [0, t - 1]^\kappa$. For the parameters h, κ, t, let $G_1 = MAP_1 \circ H_1$ map a message of arbitrary length to the Z-th secret key within the J-th tree of the I-th FORS instance. Such a key is then used for keying H_2. Moreover, let $G_2 = MAP_2 \circ H_2$ map $2n$-bit message (the concatenation of the hash key, R, of H_1 and the hash output of H_1) to a set of κ indices within the I-th FORS instance, $((I, 0, b_0), (I, 1, b_1), \ldots, (I, \kappa-1, b_{\kappa-1}))$. To this end, our v-ORS function is represented by $G = G_2 \circ G_1$. In what follows, we give a formal definition of the (post-quantum) interleaved target subset resilience ((PQ)-ITSR) of v-ORS.

Definition 16 ((PQ)-ITSR)**.** Let $\mathcal{A}$ denote a (quantum) adversary who has access to the signing oracle which on input of an m-bit message M_i, samples a key K_i at random and returns K_i, $K_{G_1} \leftarrow G_1(K_i, M_i)$, and $G_2(K_{G_1}, K_i || H_1(K_i, M_i))$. $\mathcal{A}$ is allowed to query qs messages of their choice. The success probability of (PQ)-ITSR adversary on v-ORS is given by

$$\mathsf{Succ}^{\text{(PQ)-ITSR}}_{H_2 \circ H_1, qs}(\mathcal{A}) = \mathsf{Pr}[(K', M') \leftarrow \mathcal{A}(1^n) \text{ s.t. } G(K', M') \subseteq \bigcup_{i=1}^{qs} G(K_i, M_i) \wedge M' \notin \{M_i\}_{i=1}^{qs}],$$

The (PQ)-ITSR insecurity of keyed hash functions H_1 and H_2 against any (quantum) adversary $\mathcal{A}$ who runs in time $\leq \xi$ and makes no more than qs-queries is given by

$$\mathsf{InSec}^{\text{PQ-ITSR}}(H_2 \circ H_1; \xi; qs) = \max_{\mathcal{A}} \mathsf{Succ}^{\text{(PQ)-ITSR}}_{\mathcal{H}_2 \circ \mathcal{H}_1, qs}(\mathcal{A}).$$

Note that for the target subset resilience problem used in SPHINCS [15], the adversary $\mathcal{A}$ was able to freely choose the HORST index I in the multi-target setting, while in SPHINCS$^+$, the FORS instance I is verifiable by applying the hash on the message to be signed. Moreover, $\mathcal{A}$ was also able to freely generate an ORS by freely choosing a hash randomizer R, but in v-ORS the generation of an ORS is restricted by using a secret key from the FORS instance used as the hash randomizer, which should be verified at the verification process. In what follows we analyze the complexity of a generic attack on the interleaved target subset resilience of v-ORS.

ITSR security of v-ORS. A PQ-ITSR adversary wants to find a message with ORS elements which are revealed in the ORSs of the queried qs messages. The adversary considers the following part of the signature

$$(R, (\sigma', Auth'), SIG_{FORS}) = R, (SK_{I,J,Z}, Auth_J), (SK_{I,0,b_0}, Auth_0), \ldots, (SK_{I,\kappa-1,b_{\kappa-1}}, Auth_{\kappa-1}),$$

where R is the randomizer that chooses the hash function which evaluates the FORS instance index I and secret key index, (J, Z). The secret key, $SK_{I,J,Z}$ is used as a new verifiable randomizer that generates the $ORS = b_0||b_1||\ldots||b_{\kappa-1}$. First the forger needs to find a message-randomizer pair (R', M') such that the obtained FORS secret key, $SK_{I,J,Z} \leftarrow H_1(R', PK, M')$, is revealed in the qs queries. Assuming that, the I-th FORS instance is used r times out of the qs queries, and the secret key $SK_{I,J,Z}$ is revealed in those r signatures ($\kappa + 1$ FORS secret keys are revealed in each signature), then the probability of getting an

$SK_{I,J,Z}$ that is also a previously revealed FORS secret key is given by

$$\begin{aligned}\mathsf{Pr}(SK_{I^r,J,Z}) &= \mathsf{Pr}(I^r) \times \mathsf{Pr}(SK_{I,J,Z}|I^r) \\ &= \binom{qs}{r}\left(1-\frac{1}{2^h}\right)^{qs-r}\frac{1}{2^{hr}} \times \left(1-\left(1-\frac{\kappa+1}{\kappa 2^\tau}\right)^r\right),\end{aligned} \tag{4.8}$$

where $\mathsf{Pr}(I^r)$ denotes the probability of hitting a FORS instance I such that I was used to sign r messages out of the qs queries. $\mathsf{Pr}(I^r)$ is given by the binomial probability formula $\binom{qs}{r}\left(1-\frac{1}{2^h}\right)^{qs-r}\frac{1}{2^{hr}}$ where $\binom{qs}{r}$ is the number of outcomes we want, i.e., the targeted FORS instance I is used r times out of qs. $(1-\frac{1}{2^h})^{qs-r}\frac{1}{2^{hr}}$ is the probability of each outcome, where $\frac{1}{2^{hr}}$ is the probability of targeting the I-th (out of 2^h) FORS instance for r times, and $(1-\frac{1}{2^h})^{qs-r}$ is the probability of not targeting the I-th FORS instance for the remaining $qs-r$ times. $\mathsf{Pr}(sk_{I,J,Z}|I^r)$ denotes the probability that the secret key $SK_{I,J,Z}$ is revealed in the queries where the I-th FORS instance is used r times and it is given by $\left(1-\left(1-\frac{\kappa+1}{\kappa 2^\tau}\right)^r\right)$. Note that each query reveals $(\kappa+1)$ secret keys from the same FORS instance, i.e., κ secret keys from the FORS trees corresponding to the ORS elements and one secret key that is used as the verifiable ORS randomizer.

To this end, the forger uses $(SK_{I,J,Z})$ as a new verifiable hash randomizer to generate the message digest md and correspondingly a valid ORS. Note that $(SK_{I,J,Z})$ could be any secret key that was previously revealed, whether as a hash randomizer, σ', which is the output of G_1, or as a FORS signature element, σ_i, which is an output of G_2.

For successful forgery, the elements of the generated ORS should be previously seen in the r queries for that I-th FORS instance. Recall that in each query, there are $\kappa+1$ revealed n-bit secret key elements. Let $\mathsf{P}(\text{r-TSR})$ denote the success probability of breaking the r-target

subset resilience of v-ORS which is the probability that all the generated ORS κ elements by an adversary are revealed in the r queries that are signed by the I-th FORS instance. Such a probability is given by

$$\mathsf{P(r\text{-}TSR)} = \left(1 - \left(1 - \frac{\kappa+1}{\kappa 2^{\tau}}\right)^{r}\right)^{\kappa},$$

Let Pr(ITSR) denote the success probability of a classical adversary in breaking the interleaved target subset resilience vSPHINCS^{+}. Specifically, it denotes the probability of an adversary that is successful in finding an (R', M') pair such that $SK_{I,J,Z} \leftarrow H_1(R', PK, M')$ where $SK_{I,J,Z}$ is revealed in r signatures and that when such an $SK_{I,J,Z}$ is used to evaluate md, the resulting ORS elements are revealed in the r messages signed using the I-th instance. Formally, Pr(ITSR) is the combination of $\mathsf{Pr}(SK_{I^r,J,Z})$ and P(r-TSR) over all r possible values and is given by

$$\begin{aligned}\mathsf{Pr(ITSR)} &= \sum_{r} \mathsf{Pr}(SK_{I^r,J,Z}) \times \mathsf{Pr(r\text{-}TSR)} \\ &= \sum_{r} \binom{qs}{r}\left(1 - \frac{1}{2^h}\right)^{qs-r} \frac{1}{2^{hr}} \times \left(1 - \left(1 - \frac{\kappa+1}{\kappa 2^{\tau}}\right)^{r}\right)^{\kappa+1} \end{aligned} \tag{4.9}$$

Therefore, a classical adversary that makes q_h queries to $H_2 \circ H_1$ has success probability

$$(q_h+1)\sum_{r} \binom{qs}{r}\left(1 - \frac{1}{2^h}\right)^{qs-r} \frac{1}{2^{hr}} \times \left(1 - \left(1 - \frac{\kappa+1}{\kappa 2^{\tau}}\right)^{r}\right)^{\kappa+1}$$

A quantum adversary that is running a second preimage Grover search for the hash functions

H_1 and H_2 has a success probability

$$O\left((q_h+1)^2 \sum_r \binom{qs}{r}\left(1-\frac{1}{2^h}\right)^{qs-r} \frac{1}{2^{hr}} \times \left(1-\left(1-\frac{\kappa+1}{\kappa 2^\tau}\right)^r\right)^{\kappa+1}\right)$$

4.4 vSPHINCS$^+$ Security Reduction

The security of SPHINCS$^+$ is evaluated with respect to existential unforgeability under adaptive chosen message attack (PQ)-EU-CMA, see Section 2.2.2 for definition. It has been shown that the insecurity function of SPHINCS$^+$ with respect to (PQ)-EU-CMA is bounded by the summation of the insecurity functions of the underlying hash and PRF functions with respect to specific security notions [9]. We follow similar strategy to evaluate the insecurity function of vSPHINCS$^+$ with respect to PQ-EU-CMA. However in vSPHINCS$^+$, an adversary that is successful in breaking either the $ITSR$ of v-ORS or the MM-SPR of H_1 is also successful in forging signatures. In what follows, we present the insecurity function of vSPHINCS$^+$.

Theorem 1. *For security parameter $n \in \mathbb{N}$, parameters $w, h, d, m, t, \kappa, \tau$ as described above, vSPHINCS$^+$ is existentially unforgeable under post-quantum adaptive chosen message attacks if*

- *F, H, and Th are post-quantum distinct-function multi-target second-preimage resistance hash function families,*
- *PRF, PRF_{msg} are post-quantum pseudorandom function families,*
- *$H_2 \circ H_1$ is post-quantum interleaved target subset resilience hash function families.*

- *H_1 is a post-quantum multi-function multi-target second-preimage resistance hash functions family.*

The insecurity function, $\mathsf{InSec}^{\mathsf{PQ\text{-}EU\text{-}CMA}}(vSPHINCS^+, \xi, 2^h)$, that describe the maximum success probability over all adversaries running in time $\leq \xi$ against the PQ-EU-CMA security of vSPHINCS$^+$ and making a maximum of $qs = 2^h$ queries is bounded by

$$\begin{aligned}
&\mathsf{InSec}^{\mathsf{PQ\text{-}EU\text{-}CMA}}(vSPHINCS^+, \xi, 2^h) \leq \frac{1}{2^n} + \mathsf{InSec}^{\mathsf{PQ\text{-}PRF}}(PRF, \xi) \\
&+ \mathsf{InSec}^{\mathsf{PQ\text{-}PRF}}(PRF_{msg}, \xi) + \mathsf{InSec}^{\mathsf{PQ\text{-}MM\text{-}SPR}}(H_1, \xi) + \mathsf{InSec}^{\mathsf{PQ\text{-}ITSR}}(H_2 \circ H_1, \xi) \\
&+ \mathsf{InSec}^{\mathsf{PQ\text{-}DM\text{-}SPR}}(H, \xi) + \mathsf{InSec}^{\mathsf{PQ\text{-}DM\text{-}SPR}}(Th, \xi) + \mathsf{InSec}^{\mathsf{PQ\text{-}DM\text{-}SPR}}(F, \xi)
\end{aligned}$$

Proof. The proof is based on the approach of the proof given in [14, 16]. In what follows, let the original PQ-EU-CMA game denote the game in Section 2.2.2 where $\mathcal{A}$ is allowed to make qs queries to a signing oracle running vSPHINCS$^+$. $\mathcal{A}$ wins the game if they find a valid forgery (M', Σ') where the message M' is not in the queried set of qs messages. The success probability of $\mathcal{A}$ is reduced to the probability of winning any of the following games.

- $GAME_0$ is the original PQ-EU-CMA game.

- $GAME_1$ is $GAME_0$ except the outputs of the PRF functions are replaced by values generated by a truly random generator. The difference in the success probabilities between $GAME_1$ and $GAME_0$ is bounded by $\mathsf{InSec}^{\mathsf{PRF}}(\text{PRF})$. Otherwise, $\mathcal{A}$ can be used to distinguish the PRF function from a truly random generator which contradicts the assumption of the used PRF functions.

- $GAME_2$ is similar to $GAME_1$ except that the hash randomizer R is generated using

truly number generator instead of the PRF_{msg} function. Following the same reasons as $GAME_1$, the difference in the success probability between the two games is bounded by the insecurity function of the used PRF_{msg} function ($\mathsf{InSec}^{\mathsf{PRF}}(\mathrm{PRF_{msg}})$).

- $GAME_3$ is similar to $GAME_2$ except that the game is considered lost if the resulting valid forgery (M', Σ') satisfies either of the following three cases.

  – Case 1: In such a case, the adversary $\mathcal{A}$ could find M' such that $H_1(R_j, PK, M') = H_1(R_j, PK, M_j) = h_{msg}$ where M_j is in the queried messages. In other words, $\mathcal{A}$ finds a second preimage M', for any message of the qs queried messages, (w.l.o.g., M_j) using the j-th hash randomizer R_j. Accordingly, the output of G_1 is the same FORS secret key index, $SK_{I,J,Z}$, thus, the ORS of M' is the same as that of M_j, i.e., $H_2(SK_{I,J,Z}, PK, R_j||H_1(R_j, PK, M')) = H_2(SK_{I,J,Z}, PK, R_j||H_1(R_j, PK, M_j))$. Consequently, the rest of the signature will be the same. This case describes an adversary $\mathcal{A}$ that is able to break the multi-target multi-function second preimage resistance of the hash function H_1 (PQ-MM-SPR for the H_1 function), this happens with success probability equals $\frac{q+1}{2^n}$, where q is the number of queries to the hash function H_1 (see [14] for the proof of success probability of MM-SPR).

  – Case 2: In this case, the adversary could find a message-randomizer pair (M', R') where both of the following condition hold.

    - $G_1 = MAP_1 \circ H_1(R', PK, M')$ function maps to an index of a previously revealed FORS secret key, $SK_{I,J,Z}$, i.e., it is one from those keys that were revealed through the qs queried messages.
    - $G_2 = MAP_2 \circ H_2(SK_{I,J,Z}, PK, R'||H_1(R', PK, M'))$ function maps to in-

dexes of previously revealed FORS secret keys, SK_{I,j,b_j} for $0 \leq j \leq \kappa - 1$.

In this case, the adversary can break the security of post-quantum interleaved target subset resilience of $H_2 \circ H_1$, PQ-ITSR($H_2 \circ H_1$), which has the success probability that is given in Equation 4.9.

– Case 3: In the case where the adversary does not find a message-randomizer pair (M', R') that satisfies Case 2, then there is at least one signature element (except the randomizer R) of the message signature Σ, that was not revealed through the qs signatures i.e. there is at least one element (FORS secret key) of the FORS signature that is not revealed previously. Accordingly, the forged signature must result in a second preimage of a revealed node of any of the following

  - A FORS tree node in which the secret key corresponding to ORS element is not previously revealed: the adversary is required to find a value (the corresponding secret key that supposed to be revealed) along with an authentication path in which there is a node that is a second preimage of any node of the revealed authentication paths for the same FORS tree. Accordingly from that colliding node and up, the authentication path will be the same as in the previous revealed signature. Hence, the adversary needs to break the PQ-DM-SPR security of the H function,
  - The FORS instance root, i.e., the adversary is required to find a value (the corresponding secret key that is supposed to be revealed) along with an authentication path that results in a FORS tree root such that when concatenated with the other FORS tree roots of the FORS instance, collides with the revealed FORS instance root. Hence, the adversary needs to break the PQ-DM-SPR security of the Th function

- A WOTS$^+$ node from the d leaf nodes that sign the root of the down layer tree. Hence, the adversary needs to break the PQ-DM-SPR for the F function or the Th function that evaluates WOTS$^+$.PK,
- Any node of the d subtrees except the leaf nodes (breaking the PQ-DM-SPR of the H function)

The difference in the success probability between $GAME_3$ and $GAME_2$ is bounded by $\mathsf{InSec}^{\text{PQ-MM-SPR}}(H_1)+\mathsf{InSec}^{\text{PQ-ITSR}}(H_2 \circ H_1)+2^{-n}+\mathsf{InSec}^{\text{PQ-DM-SPR}}(H)+\mathsf{InSec}^{\text{PQ-DM-SPR}}(Th)+\mathsf{InSec}^{\text{PQ-DM-SPR}}(F)$, otherwise, the adversary could break the security of the post-quantum multi-function multi-target second-preimage resistance of H_1 hash function, or the security of the post-quantum interleaved target subset resilience of $H_2 \circ H_1$, or the security of the post-quantum distinct-function multi-target second-preimage resistance of F, H, or Th. Combining all the games together gives the bound of the insecurity function of vSPHINCS$^+$ with respect to EU-CMA.

vSPHINCS$^+$ bit security. The EU-CMA bit security of vSPHINCS$^+$ is calculated by $-\log_2$ of the $\mathsf{InSec}^{\text{EU-CMA}}(\text{vSPHINCS}^+)$ which is bounded by combining the success probabilities of the ITSR of the hash functions $H_1 \circ H_2$ introduced in Section 4.3 and those security notions in Theorem 1, where the classical adversary makes q_h queries to the hash function. Note that in such a case, the PRF, MM-SPR, and DM-SPR success probabilities are given

by $\frac{q_h+1}{2^n}$, and consequently the $\mathsf{InSec}^{\text{EU-CMA}}(\text{vSPHINCS}^+)$ is bounded by

$$\begin{aligned}
&\mathsf{InSec}^{\text{EU-CMA}}(\text{vSPHINCS}^+, q_h) \leq \frac{q_h+1}{2^n} + \frac{q_h+1}{2^n} + \frac{q_h+1}{2^n} \\
&+ \frac{q_h+1}{2^n} + \mathsf{InSec}^{\text{ITSR}}(H_2 \circ H_1, \xi) + \frac{q_h+1}{2^n} + \frac{q_h+1}{2^n} + \frac{q_h+1}{2^n} \\
&\leq 7 \cdot \frac{q_h+1}{2^n} + (q_h+1) \sum_r \binom{2^h}{r} \left(1 - \frac{1}{2^h}\right)^{2^h-r} \frac{1}{2^{hr}} \left(1 - \left(1 - \frac{\kappa+1}{\kappa 2^\tau}\right)^r\right)^{\kappa+1} \\
&\leq O\left(\frac{q_h+1}{2^n} + (q_h+1) \sum_r \binom{2^h}{r} \left(1 - \frac{1}{2^h}\right)^{2^h-r} \frac{1}{2^{hr}} \left(1 - \left(1 - \frac{\kappa+1}{\kappa 2^\tau}\right)^r\right)^{\kappa+1}\right),
\end{aligned}$$

The classical bit security of vSPHINCS$^+$ is given by

$$b = -\log_2 \left(\frac{1}{2^n} + \sum_r \binom{2^h}{r} \left(1 - \frac{1}{2^h}\right)^{2^h-r} \frac{1}{2^{hr}} \left(1 - \left(1 - \frac{\kappa+1}{\kappa 2^\tau}\right)^r\right)^{\kappa+1}\right) \tag{4.10}$$

The quantum bit security is given by

$$b = -\frac{1}{2}\log_2 \left(\frac{1}{2^n} + \sum_r \binom{2^h}{r} \left(1 - \frac{1}{2^h}\right)^{2^h-r} \frac{1}{2^{hr}} \left(1 - \left(1 - \frac{\kappa+1}{\kappa 2^\tau}\right)^r\right)^{\kappa+1}\right)$$

4.5 vSPHINCS$^+$ Comparison and New Parameters

The success probability of an ITSR adversary on vSPHINCS$^+$is provided in Equation 4.9, the corresponding success probability for SPHINCS$^+$ is given by

$$\sum_r \binom{2^h}{r} \left(1 - \frac{1}{2^h}\right)^{2^h-r} \frac{1}{2^{hr}} \left(1 - \left(1 - \frac{1}{2^\tau}\right)^r\right)^{\kappa}$$

It can be seen that our modification enhances the security of SPHINCS$^+$ because the power of the last term is greater than the corresponding one in SPHINCS$^+$. Note that we can approximate $\frac{\kappa+1}{\kappa 2^\tau}$ by $\frac{1}{2^\tau}$ for $2^\tau >> \kappa$, but this is not considered in the results presented in this section.

In Table 4.1, we provide the ITSR bit security, signature size, and the signing computational cost (i.e., the number of hash calls required to generate a signature, where the inputs to all of these hash calls have the same length) for both SPHINCS$^+$ and vSPHINCS$^+$ using the original parameters of different versions of SPHINCS$^+$. The signature size for SPHINCS$^+$ is calculated by

$$(h + \kappa(\tau + 1) + d.l + 1)n \text{ bits.}$$

For vSPHINCS$^+$, this signature size is given by

$$(h + (\kappa + 1)(\tau + 1) + d.l + 1)n \text{ bits.}$$

For both SPHINCS$^+$ and vSPHINCS$^+$, the number of hash calls required for signing is given by

$$2(d(l \cdot 2^w \cdot 2^{h/d} + 2^{h/d} - 1) + 2 \cdot \kappa \cdot 2^\tau + \kappa(2^\tau - 1)).$$

SPHINCS$^+$ provides two instantiations, simple and robust. The former istantiation does not require the use of bismasks, hence, provides faster signing. Our calculations in this work considers the instances of the simple instantiation. Nevertheless, for robust instantiations, vSPHINCS$^+$ attains the same performance ratios when compared to SPHINCS$^+$ as it does with the simple instantiations. In both instantiations, SPHINCS$^+$ offers 6 instances with different parameters at different security levels. Specifically, for each n-bit security level, SPHINCS$^+$ offers one parameter set for fast computation, denoted by SPHINCS$^+$-nf and

another for small signature size, denoted by SPHINCS$^+$-ns.

Table 4.1: Interleaved target subset resilience bit security, signature size, and number of hash calls for SPHINCS$^+$ and vSPHINCS$^+$ with the original recommended SPHINCS$^+$ round-three parameters

SPHINCS$^+$ instance	h	d	τ	κ	**SPHINCS$^+$**			**vSPHINCS$^+$**		
					bitSec	size	Hash calls	bitSec	size	Hash calls
SPHINCS$^+$-128s	63	7	12	14	133	7856	4372438	141	8064	4372438
SPHINCS$^+$-128f	66	22	6	33	128	17088	210386	132	17200	210386
SPHINCS$^+$-192s	63	7	14	17	193	16224	7534544	203	16584	7534544
SPHINCS$^+$-192f	66	22	8	33	194	35664	338514	198	35880	338514
SPHINCS$^+$-256s	64	8	14	22	255	29792	6561732	265	30272	6561732
SPHINCS$^+$-256f	68	17	9	35	255	49856	691672	260	50176	691672

As depicted in Table 4.1, vSPHINCS$^+$ provides higher bit security than SPHINCS$^+$. Note that, SPHINCS$^+$ parameters were chosen to achieve a certain n-bit security, hence, using the same parameters, vSPHINCS$^+$ achieves higher than n bits of security. On the other hand, the corresponding signature size of vSPHINCS$^+$ is slightly increased by $(\tau + 1)n$ bits. For instance, for SPHINCS$^+$-128s (128 bit security is required), SPHINCS$^+$ achieves 133 bit security while vSPHINCS$^+$ achieves 141 bit security. Since the recommended parameters for SPHINCS$^+$-128s enable vSPHINCS$^+$ to offer 13 bits more than the required 128-bit security, we can search for different parameters for the FORS scheme to improve the performance of vSPHINCS$^+$.

4.5.1 Efficient Parameter Sets

Our initial goal was to have the same signature size as SPHINCS$^+$while providing a bit security equal to or greater than that required. Accordingly, we chose to decrease the value of κ by one which means a FORS instance in vSPHINCS$^+$ has one less FORS tree than in SPHINCS$^+$. This enables vSPHINCS$^+$ to have the same signature size as SPHINCS$^+$ while maintaining an ITSR bit security that is higher than that required. Note that we

are comparing the ITSR bit security of the two schemes because if the chosen parameters enable an ITSR-bit security more than the targeted n bits, then an adversarial forgery is more efficient through a generic SPR attack on one of the used hash functions.

Table 4.2 presents the security level, signature size, computational cost, the percentage difference in signature size and hash calls when vSPHINCS$^+$ with newly explored parameters is compared to the original SPHINCS$^+$ instances. A red $+x$ (resp. green $-y$) denotes an increase (resp. decrease) by x% (resp. y%) relative to that of an SPHINCS$^+$instance.

Table 4.2: Interleaved target subset resilience bit security, signature size, and number of hash calls for vSPHINCS$^+$ with the new FORS parameters. Green "-" (resp. red "+") denotes a percentage reduction (resp. increase) in the number of hash calls or the signature size relative to SPHINCS$^+$

SPHINCS$^+$ instance	h	d	τ	κ	**vSPHINCS$^+$**				
					bitSec	size	Hash calls	% size	% calls
SPHINCS$^+$-128s	63	7	12	13	132	7856	4347864	0	-0.56
SPHINCS$^+$-128s	63	7	10	17	131	8112	4132816	+3.25	-5.48
SPHINCS$^+$-128f	66	22	6	32	129	16976	210004	0	-0.18
SPHINCS$^+$-192s	63	7	14	16	192	16224	7436242	0	-1.3
SPHINCS$^+$-192s	63	7	13	17	192	16152	6698960	-0.44	-11
SPHINCS$^+$-192f	66	22	8	32	193	35664	336980	0	-0.45
SPHINCS$^+$-256s	64	8	14	21	254	29792	6463430	0	-1.5
SPHINCS$^+$-256s	64	8	11	30	256	31136	4767668	+4.5	-27
SPHINCS$^+$-256f	68	17	8	41	255	50752	647116	+1.8	-6.4

The small instances, e.g., SPHINCS$^+$-128s, have fewer tree layers and FORS trees than the fast instances, e.g., SPHINCS$^+$-128f, which results in a smaller signature size but more computations, i.e., more hash calls for signing as the tree has more leaves than the fast instance. Accordingly, by decreasing the value of κ in vSPHINCS$^+$, we are removing a FORS tree from the original instance which maintains the same signature size as in SPHINCS$^+$. As the number of FORS trees within a FORS instance in the fast construction is larger and the FORS tree itself is smaller than those in the small construction, removing a FORS

tree results in a lesser effect (i.e., reduction in signature size and saving more hash calls) than deleting a FORS tree in the small construction. Note that the computation savings is a percentage of all SPHINCS$^+$ hash calls, including the hash calls for the subtrees. As a result, the percentages in Table 4.2 for instances with just one FORS tree deleted (corresponding to 0 % for the size change) are not huge.

We have looked for other parameters that achieve better computational cost. For each instance, we were able to find around two parameter sets that lead to computation saving and either no or slight increase in the signature size. For instance, we found parametrizations that attain computational savings of around 27% in vSPHINCS$^+$-256s (resp. 5.5% for vSPHINCS$^+$-128s) with a very small increase in the signature size, 4.5% (resp. 3.25%). Note that the signature size increase in the case of the vSPHINCS$^+$-256s instance is slightly higher than the other instance because these new parameters enable vSPHINCS$^+$ to achieve the required 256-bit security while SPHINCS$^+$ attains 255 bits of security. For vSPHINCS$^+$-192s, we achieve computational saving of 11% and a signature size saving of 0.44% relative to SPHINCS$^+$-192s with the original parameters.

4.5.2 SPHINCS$^+$ Re-parameterization in Round Three Submission

On October 23, 2020, 4 instances of SPHINCS$^+$ had their parameters modified in the round three submission to the NIST PQC. The numbers in Table 4.1 reflect the performance of vSPHINCS$^+$ when compared to SPHINCS$^+$ with the new round 3 parameters. For SPHINCS$^+$-128f and SPHINCS$^+$-256f the parameter change improved the computational cost by 22.6% and 9.9%, and increased the signature sizes by 0.66% and 1.3%, respectively. For SPHINCS$^+$-128s, the new parameters resulted in an increase of 2.4% in the computation cost and decrease of 2.8% in the signature size. Table 4.3 depicts the new round 3 parameters

for SPHINCS$^+$ instances and the percentage change relative to round 2 parameters. As shown in Table 4.2, even with the new round 3 parameters, v-ORS betters the computational cost of all SPHINCS$^+$ instances, with one instance, i.e., SPHINCS$^+$-256s, attaining around 27% decrease in the signing computation.

Table 4.3: Interleaved target subset resilience bit security, signature size, number of hash calls for SPHINCS$^+$ round 2 and round 3 parameters, along with the relative percentage change in the signature size and number hash calls

SPHINCS$^+$ instance	**SPHINCS$^+$ R3**			**SPHINCS$^+$ R2**			**% change**	
	bitSec	size	Hash calls	bitSec	size	Hash calls	% size	% H calls
SPHINCS$^+$-128s	133	7856	4372438	133	8080	4267996	-2.8	+2.4
SPHINCS$^+$-128f	128	17088	210386	128	16976	271900	+0.66	-22.6
SPHINCS$^+$-192s	193	16224	7534544	196	17064	8855508	-4.9	-14.9
SPHINCS$^+$-192f	194	35664	338514	194	35664	338514	0	0
SPHINCS$^+$-256s	255	29792	6561732	255	29792	6561732	0	0
SPHINCS$^+$-256f	255	49856	691672	254	49216	768482	+1.3	-9.9

Note on the small instances. We observed that in the re-parameterized small instances, SPHINCS$^+$-128s and SPHINCS$^+$-192s, the hyper-tree height h and the number of layers d are decreased from 64 to 63 and from 8 to 7, respectively. We can tweak this strategy for vSPHINCS$^+$ to achieve more computational saving. Concretely, for vSPHINCS$^+$-128s, we can choose the number of layers, d, to be 9 instead of 7 with $\tau = 12$, and $\kappa = 13$, which leads to 63.08% saving in the hash calls, while increasing the signature size by 14.25% when compared to SPHINCS$^+$-128s with round 3 parameters.

4.6 Conclusion

In this chapter we proposed v-ORS, a new ORS generation mechanism that enables SPHINCS$^+$ to provide better performance at the same security level. Using v-ORS, a signed message is bound with the signing FORS instance which restricts a forging adversary to searching among those queries that use that specific FORS instance. The increased

restrictions allow some freedom in exploring efficient parameters for the underlying FORS scheme, which in turn enables SPHINCS$^+$ using v-ORS to achieve better performance. More precisely, v-ORS allows some versions of SPHINCS$^+$ to offer around 27% savings in the signing computational cost with minimal effect on the signature size. Given that the high computational cost is the main reason for selecting SPHINCS$^+$ as an alternate candidate in Round 3 of the NIST post-quantum cryptography competition, the results presented here are a positive step towards making its practical adoption widely accepted.

Chapter 5

Security Analysis Of DGM and GM Group Signature Schemes Instantiated With XMSS-T

In this chapter, we provide a security analysis for Group Merkle (GM) (PQCrypto 2018) and Dynamic Group Merkle (DGM) (ESORICS 2019) which are the recent proposals for post-quantum hash-based group signature schemes. GM and DGM are designed as generic constructions that employ any stateful Merkle hash-based signature scheme. XMSS-T (PKC 2016, RFC8391) is the latest stateful Markle hash-based signature scheme where (almost) optimal parameters are provided [27]. In this chapter, we show that the setup phase of both GM and DGM does not enable their drop-in instantiation by XMSS-T, thus limiting both designs to employing earlier XMSS versions with sub-optimal parameters which negatively affects the performance of these schemes. We provide simple changes to the setup phase of GM and DGM to address this limitation and enable the adoption of XMSS-T. Moreover, we analyze the bit security of DGM when instantiated with XMSS-T and show that it

is susceptible to multi-target attacks because of its parallel Signing Merkle Trees (SMT) approach. More precisely, when DGM is used to sign 2^{64} messages, its bit security is 44 bits less than that of XMSS-T. Finally, we provide a DGM variant that mitigates multi-target attacks and show that it attains the same bit security as XMSS-T. The contributions of this chapter were published in [24].

5.1 Introduction

A Group Signature Scheme (GSS) incorporates N members into a signing scheme with a single public key. This allows any group member to sign anonymously on behalf of the whole group [31]. A group manager is assigned to perform the system setup, reveal the identity of a given signer when needed, add new members, and revoke memberships as required. Remote attestation protocols, e-commerce, e-voting, traffic management, and privacy preserving techniques in blockchain [32, 33, 34] are some of the applications that utilize group signature schemes. There have been several proposals for group signature schemes [32, 35, 36, 37, 38, 39]. However, the majority of them rely on number theoretic assumptions that are not secure against post-quantum attacks.

Currently, there is an imperative need to supersede the current public key infrastructure by quantum-secure algorithms. This need is evidenced by the current post-quantum cryptography standardization competition (PQC) run by NIST [40]. GSS is one of the public key infrastructure primitives which researchers are investigating to provide quantum security. The first post-quantum lattice-based group signature scheme was introduced in 2010 [41]. Subsequently, a number of lattice-based structures were introduced [42, 43, 44, 45, 46, 47]. Nevertheless, unlike the lattice-based signature schemes finalist candidates in PQC, their group signature structures are not as efficient [48]. Code-based group signature schemes

were developed to provide another alternative for quantum secure GSSs [49, 50, 51], but they have very large signature sizes on the order of megabytes [52]. Nevertheless, the construction of the Zero-Knowledge proof [53] ZKBoo [54], constructed only with symmetric primitives, shows that symmetric primitives can also be used to design post-quantum signature and group signature schemes. ZKBoo uses the concept presented by Ishai *et al.* namely MPC in the head [55]. From this, one can achieve new optimized Zero-Knowledge proofs such as [56, 57, 58] [2,9] or [19]. The state-of-the-art post-quantum signatures based on symmetric primitives are constructed by Chase et al., who built digital signature schemes by designing ZKB++ [56][9], an optimization of ZKBoo. Alongside ZKB++, the Fiat-Shamir Transform [59][12] and Unruh Transform [60][30] were employed to construct a Non-Interactive Zero-Knowledge proof (NIZK) [61][3].

In 2018, El Bansarkhani and Misoczki introduced Group Merkle (GM), the first post-quantum stateful hash-based group signature scheme [20]. The following year, Dynamic Group Merkle (DGM), the latest hash-based group signature scheme, was introduced to solve some of the limitations of GM [21]. GM and DGM provide quantum security with reasonable signature sizes on the order of kilobytes and both are general constructions that can be instantiated with any stateful Merkle hash-based signature scheme. The security analysis of both schemes included standard security notions of group signature schemes (anonymity and full-traceability) [62, 63], but no bit security analysis was provided. Shafieinejad *et al*. proposed a grouped one-time signature scheme (GOT) which when deployed in a Merkle tree construction allows any member of the group to sign with any GOT leaf of the group tree [64]. This design increases the number of OTS secret keys, e.g., l secret keys in the Winternitz One-Time Signature (WOTS) scheme, by a factor of $\sqrt{N}$ where N is the number of group members. The GOT secret keys are shared between all users using a traversal

design scheme where each user has l-secret keys such that no other member gets the same key set. The limitations of GOT include the requirement that a common updated state should be shared between the group members to prevent signing by the same index more than once. More importantly, a group of colluding members can generate a valid signature that reveals the identity of the signer to a member outside the colluding group. This attack is applicable because the security analysis was presented under the assumption of honest group members.

XMSS$^+$ [12], XMSS^{MT} [13], and XMSS-T [14] are stateful hash-based signature schemes that were proposed to tackle the performance drawbacks of the Merkle Signature Scheme (MSS) [10]. The last version of XMSS-T as described in Internet Engineering Task Force (IETF) RFC8391 [65] provides (almost) optimal parameters and mitigates multi-target attacks.

5.2 Specification of Related Schemes

In this section, we provide a brief description of XMSS-T, GM, and DGM, the related signing schemes used throughout this chapter. We only provide details of the procedures that are relevant to our analysis. For more details, the reader is referred to [14, 20, 21, 65].

5.2.1 Extended Merkle Signature Scheme-Tightened (XMSS-T)

XMSS-T is a multi Merkle tree construction where the leaf nodes of the trees are the public keys of the Winternitz One-Time Signature scheme with Tightened security (WOTS-T) [3]. In what follows, we consider the specification of one tree instance of XMSS-T, because this is how it is used in GM and DGM. XMSS-T has a public addressing mapping scheme, $ADRS$, that maps a public seed, $pk.seed$, a leaf/internal node index, i, and a level, j, to generate a (distinct) new hash randomizer, r, and bit-mask, q, for each hash

call in the scheme (hashing in the WOTS-T scheme and the Merkle tree hashing). Such a distinct randomizer enables the scheme to mitigate multi-target attacks. Precisely, a Merkle tree of height h, has 2^h leaf nodes (WOTS-T.pk), and the i-th node at level j is denoted by $X_{i,j}$ where $0 \leq i < 2^{h-j}, 0 \leq j \leq h$. The internal nodes are generated by $X_{i,j} = H(r_{i,j}, (X_{2i,j-1}||X_{2i+1,j-1}) \oplus q_{i,j})$ where $r_{i,j}$ and $q_{i,j}$ are the hash randomizer used and the bit-mask generated by the addressing scheme $(r_{i,j}, q_{i,j}) \leftarrow ADRS(pk.seed, i, j)$. The XMSS-T addressing scheme (see Appendix A.5 for details) takes the leaf index, i, and calculates j according to the hashing sub-structure, i.e. OTS hash chains, L-tree hashing, or Merkle tree hashing. Then, it generates the required hash randomizer and bit mask. For simplicity, we let $ADRS$ take the node level, j, as input.

The nodes at level 0, $X_{i,0}$, are the leaf nodes, and they are the public keys of WOTS-T which also utilizes the addressing scheme, $ADRS$, to evaluate the required hash randomizers and bit masks for hashing. For details of the WOTS-T signing scheme and the addressing schemes, the reader is referred to [14] and Appendix A.5, respectively. Figure 5.1 depicts a simplified example of XMSS-T with one tree having 8 signing leaves $L_0, \cdots, L_7$. A signature using leaf L_2 (colored in black) has all the gray nodes in its authentication path.

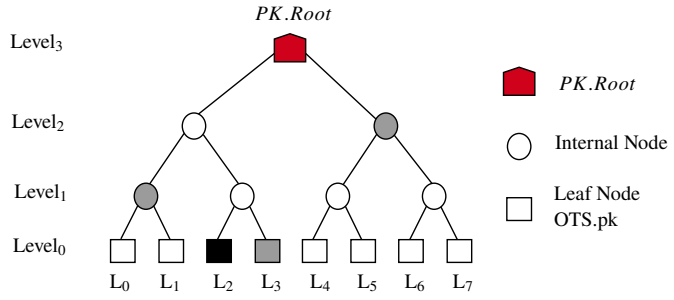


Figure 5.1: A single layer XMSS-T where the leaf nodes are the WOTS-T public keys. The nodes colored in gray are the authentication path for signing with the leaf node L_2.

5.2.2 Group Merkle (GM)

Group Merkle (GM) is the first post-quantum hash-based group signature scheme. It is a single Merkle tree construction that can be instantiated by any stateful one-tree Merkle hash-based signature scheme that employs a One-Time Signing (OTS) scheme as the underlying signing algorithm. The group manager in GM is responsible for the setup phase for N group members. In this phase, a member j, $1 \leq j \leq N$, generates B OTS keys and sends the corresponding public keys $(OTS.pk_{(j-1)B+1}, OTS.pk_{(j-1)B+2}, \ldots, OTS.pk_{jB})$ to the group manager who labels all NB keys received from the N members by $(1, 2, \cdots, NB)$, where the set of consecutive labels $(j-1)B+1, (j-1)B+2, \ldots, jB$ belongs to the j-th member.

To ensure signer anonymity, the OTS public keys are shuffled by encrypting the corresponding labels using a symmetric encryption algorithm, $pos_i = Enc(i, sk_{gm})$, where sk_{gm} is the group manager's secret key and $1 \leq i \leq NB$. Thus the group manager has the pairs $(OTS.pk_1, pos_1), \cdots, (OTS.pk_{N \cdot B}, pos_{N \cdot B})$. These pairs are reordered in ascending order of the encrypted positions to perform the pair permutation. Then the GM tree is constructed where the leaf node denoted by $L_i = X_{i,0}$ contains the pair $(OTS.pk_j, pos_j)$ and i is the new permuted position of $OTS.pk_j$. Accordingly the p-th node at level 1 is calculated using $X_{p,1} = H(X_{2p,0}||X_{2p+1,0}) = H(OTS.pk_x, pos_x||OTS.pk_z, pos_z)$ for $0 \leq p \leq \frac{NB}{2} - 1$, i.e. $L_{2p} = X_{2p,0} = (OTS.pk_x, pos_x)$ and $L_{2p+1} = X_{2p+1,0} = (OTS.pk_z, pos_z)$, because after the permutation, position x is mapped to $2p$ and position z is mapped to $2p+1$. Hashing neighboring nodes continues up the levels until the tree root is evaluated which is the group's public key $GM.gpk$. Note that the encrypted position is included in the signature, and is used by the group manager to reveal the identity of the signer. Figure 5.2 shows a simplified example of a GM tree with two members colored in red and blue, each having 2 OTS key pairs.

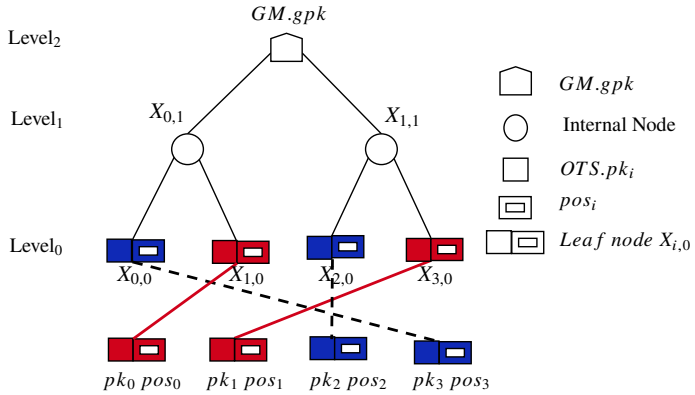


Figure 5.2: GM with two members colored in red and blue, each having two signing leaves. The leaf permutation is done by sorting the encrypted positions.

5.2.3 Dynamic Group Merkle (DGM)

DGM [21] combines two types of Merkle trees, an Initial Merkle Tree (IMT) and multiple Signing Merkle Trees (SMTs). The IMT has a height of 20 which is the maximum height that can be used in practice and have acceptable performance. The IMT leaves are random values and they are not signing leaves. They are used to build the IMT tree whose root is the group public key $DGM.gpk$. SMTs have variable height and their leaves are OTSs which are used by group members to sign messages. Initially, a group member asks the group manager for B OTS signing keys and the group manager randomly choose B internal nodes from the IMT, i.e. nodes at levels $1, 2, \ldots, 19$, and assigns an OTS from each SMT that is linked to one of these internal nodes. If all the OTSs of an existing SMT are assigned or an IMT internal node does not have an SMT yet, then a new SMT is generated. The height of an SMT is equal to the level of the internal node that it is linked to.

SMT generation: An SMT is constructed in the same manner as a GM tree. However, in DGM, the OTS secret and public key pairs are generated by the group manager and the SMT is built without input from group members. Let $OTS.pk_i$ denote the i-th OTS

public key, $0 \leq i \leq z$, where z denotes the total number of signatures supported by the scheme. Let $i = (v, u)$ where u is the OTS number within the v-th SMT. The OTS public keys are generated and indexed by $DGM.i = (v, u)$. These indexes are then encrypted with a symmetric encryption algorithm to generate $DGM.pos_i = Enc(DGM.i, sk_{gm})$ where sk_{gm} is the group manger secret key. The OTS public keys are then shuffled by sorting the encrypted positions $DGM.pos$. Afterwards, the SMT leaves are generated, precisely, the j-th SMT leaf node is the hashing of the concatenation of the i-th OTS public key and their encrypted position, $L_j = H(OTS.pk_i||DGM.pos_i)$, where j is the new position of the i-th OTS after the permutation. These leaves are used to build the SMT and evaluate its root r_{SMT} which is then linked to an IMT internal node called the fallback node, Fn, using a symmetric encryption algorithm. More precisely, r_{SMT}, is linked to Fn by evaluating the fallback key as $Fk = Dec(Fn, r_{SMT})$ which is included in the signature. Note that the verifier has to communicate with the group manager to check the validity of the received Fk, and then calculate $Fn = Enc(Fk, r_{SMT})$ to complete the verification process. After all the leaves of an SMT have been used, a new SMT is generated and linked to the same Fn. Note that different SMTs linked to the same fallback node Fn, have different fallback keys.

Figure 5.3 depicts a simplified example of a DGM where the IMT, colored in blue, has height 4. The SMT colored in red is linked to the first IMT internal node, $Fn = X_{0,3}$, at level 3. When L_2 is used to sign a message M, the resulting signature is given by $\Sigma = (indx, OTS.\sigma_{indx}, DGM.pos_i, Auth)$, where $indx = 2$ is the signing leaf index with respect to the IMT to enable determining which node is concatenated on its right and left in both the SMT and IMT from the authentication path in the verification process, $OTS.\sigma_{indx}$ denotes the OTS signature by the leaf index $indx$, and $Auth = Auth_{SMT}, Fk, Auth_{IMT}$, where $Auth_{SMT} = L_3, SMT.X_{0,1}, SMT.X_{1,2}$ is the SMT authentication path (colored in

pink), and $Auth_{IMT} = IMT.X_{1,3}$, colored in light blue, is the IMT authentication path for the fallback node Fn.

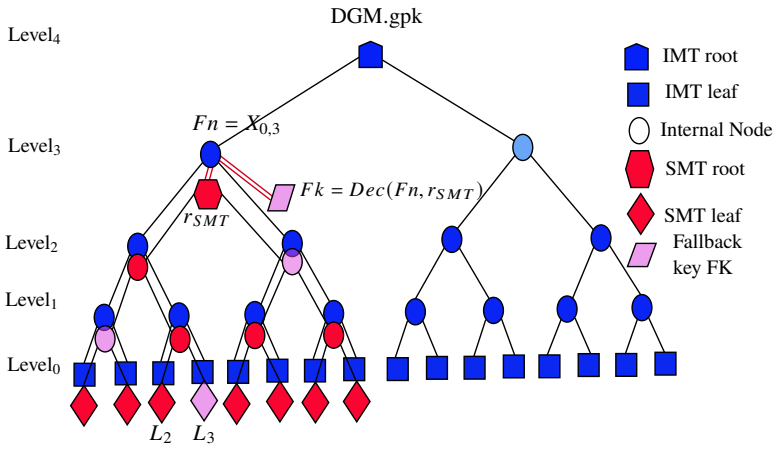


Figure 5.3: DGM.

5.3 Instantiating GM And DGM with XMSS-T

In both GM and DGM, the signing leaves which contain the public keys of the OTS used by the group members, must first be generated and then permuted. Afterwards, the Merkle tree (SMT in DGM), is built using the permuted leaves. In GM, the group members generate their OTS keys and send the corresponding OTS public keys to the group manager who permutes them and then builds the GM tree. Finally, the group manager distributes the GM tree signing leaves to the members. On the other hand, in DGM, the group manager generates the OTSs on behalf of the group members, evaluates the signing leaves and permutes them, then constructs the SMTs and assigns at random OTSs from randomly chosen SMTs.

XMSS-T is the latest stateful MSS variant and has (almost) optimal parameters when compared to the other MSS variants which translates into smaller signature. For instance, with the same parameters (SHA-256 hash function, Winternitz parameter w=16, tree height

=20), the bit security of XMSS-T (resp. XMSS [11]) is 256 (resp. 196). With 196 bit security, XMSS-T (resp. XMSS) has a signature size of 14,328 (resp. 22,296) bits. XMSS-T uses WOTS-T as the underlying OTS signing scheme which requires the signing leaf index, i, within the Merkle tree to generate the OTS public keys. More precisely, XMSS-T uses an addressing scheme that utilizes the signing leaf index within the Merkle tree as input to generate a distinct hash randomizer and bit mask for each hash call in all the hash chains of WOTS-T [14] (see Appendix A.5). These hash randomizers and bit masks are used to evaluate the WOTS-T public keys which represent the signing leaves (see Section 5.2.1).

Instantiating GM and DGM using XMSS-T is not directly achievable because in the specification of both schemes, a signing leaf index, i, is known only after its corresponding OTS public key has been generated and the associated leaf permuted, while in XMSS-T, WOTS-T requires the leaf index, i, to evaluate the OTS public key and generate the corresponding leaf. One solution is to employ an earlier XMSS version with OTS variant that does not require the position of the leaf within the Merkle tree to evaluate the OTS public keys. This results in using OTS with larger parameters than WOTS-T which negatively affects the performance of the group signature scheme.

GM and DGM with XMSS-T. We provide simple changes in the setup phase of both GM and DGM which enables their instantiation with XMSS-T. In GM, the setup phase is interactive so we add an extra communication step between the group manager and the group members where the permuted indexes are first sent to the members who can then generate their WOTS-T public keys. More precisely, the permutation in GM is done by encrypting a given position that is associated with an OTS public key, but the encryption itself is independent of the value of the public key, i.e. $pos_i = Enc(i, sk_{gm})$. Accordingly, the group manger can initially permute the indexes of the leaves for all group members

before the OTS keys are generated. Afterwards, the permuted indexes are assigned to group members in a manner similar to the original setup phase (see Section 5.2.2). Each group member uses the assigned indexes within the tree as an input to the WOTS-T addressing scheme, $ADRS$, to generate the required hash randomizers and bitmasks which are required to generate their WOTS-T public keys. Finally, the WOTS-T public keys are sent back to the group manager who constructs the GM tree using XMSS-T.

In DGM, no extra communication is needed because the group manger generates the OTS signing keys and corresponding public keys for the group members. Accordingly, the manager may permute the indexes using a symmetric encryption algorithm and then generate the OTS public keys using the permuted indexes. In other words, the specification of the setup phase stays the same with only the permutation and OTS key generation steps swapped.

5.4 DGM with XMSS-T Security Analysis

In [21] DGM was analyzed with respect to the security notions of group digital signature schemes, i.e. anonymity and traceability. However, since DGM was not instantiated with a specific Merkle signing scheme, no bit security analysis for its unforgeability was provided. In this section, we analyze the bit security of DGM unforgeability when it is instantiated with XMSS-T. We note that the same analysis is valid if DGM is instantiated with earlier XMSS versions. Henceforth, we refer to DGM when instantiated with XMSS-T as simply DGM.

Unforgeability in group signature schemes. A basic security notion of (group) digital signature scheme is that signatures cannot be forged. More precisely, it is computationally infeasible for an adversary $\mathcal{A}$ who does not know the secret key and is allowed unrestricted

queries to the signing oracle to generate a message signature pair (M', Σ') that passes as valid by the verification algorithm. In what follows, we give the definition of the unforgeability game $\mathsf{EXP}^{forge}_{\mathcal{GS},\mathcal{A}}(n, N)$ for a group signature scheme, $\mathcal{GS}$ with N members and security parameter n. Such a game is described by Bellare *et al.* in [62] as an adaptation from the traceability game where $\mathcal{A}$ is not allowed to corrupt members. Intuitively, $\mathcal{A}$ is successful in winning $\mathsf{EXP}^{forge}_{\mathcal{GS},\mathcal{A}}$ if the forged message is either traced to a group member or cannot be traced to any member.

Definition 17 (Unforgeability)**.** *A group signature scheme $\mathcal{GS}$ is unforgeable if for any PPT adversary $\mathcal{A}$ that is given unrestricted access to the signing and opening oracles, $\mathcal{A}$ is not able to generate a valid signature for a message that was not queried before. $\mathcal{A}$ has a negligible advantage in the experiment $Exp^{forge}_{\mathcal{GS},\mathcal{A}}$ as defined in Figure 5.4*

$$Adv^{forge}_{\mathcal{GS},\mathcal{A}}(n, N) = |\ Pr[Exp^{forg}_{\mathcal{GS},\mathcal{A}}(n, N) = 1]\ | \leq negl(n)$$

$Exp^{forge}_{\mathcal{GS},\mathcal{A}}(n, N)$
- $(GPK, sk^*) \leftarrow G.KGen(1^n, 1^N)$
- Unrestricted queries:
  * $Sign(\cdot, M)$
  * $G.Open(M, \Sigma)$
- Generate (M', Σ')
- **If** $G.Verify(\Sigma', M', gpk) == 1$ **Return** 1
  Else Return 0

Figure 5.4: Unforgeability experiment

5.4.1 Multi-Target Attacks and XMSS-T

If an n-bit hash function is used once in a cryptographic primitive with security parameter λ whose security is dependent on the second preimage resistance of the hash function, then finding a second preimage of the generated digest requires 2^n computations, so it suffices

that $n = \lambda$. However, if the same hash function is used t times in the cryptographic primitive, i.e. an adversary has access to t digests generated with the same hash function, then a second preimage may be obtained on any of these t targets with $2^n/t$ computations. Assuming that $n = \lambda$, the security of the scheme is reduced from n to $n - \log t$. A naive remedy to obtain n-bit security is to use a message digest of length $n + \log t$. Alternatively, one may require that each hash application is different such that each digest for the t targets is evaluated by a different hash function. Thus finding a second preimage for any function, i.e. using the same hash key, requires 2^n computations.

In XMSS-T, the addressing scheme, $ADRS$, generates a hash randomizer and bit mask for each hash function call depending on the hash node index in the tree or WOTS-T chain iteration. For a tree of height h, the i-th node at level j is denoted by $X_{i,j}$, where $0 \leq i \leq 2^{h-j}$, $0 \leq j \leq h$. $ADRS$ is given by $(r_{i,j}, q_{i,j}) \leftarrow ADRS(pk.seed, i, j)$ where $r_{i,j}$ and $q_{i,j}$ are the hash randomizer and bit-mask used. The internal nodes are generated as $X_{i,j} = H(r_{i,j}, (X_{2i,j-1}||X_{2i+1,j-1}) \oplus q_{i,j})$, i.e. $H_{r_{i,j}}$ is unique for $X_{i,j}$. Accordingly, if an adversary collects all the signatures supported by the scheme, each element in the WOTS-T signatures and each node in any the authentication path is generated by a different hash function. Consequently, creating a forgery requires finding a second preimage of a given node using the corresponding hash function where other nodes are no longer applicable targets.

5.4.2 Multi-Target Attacks on DGM

DGM allows multiple SMT trees to branch out of any IMT internal node, i.e. the fallback node. Accordingly, one may regard DGM as several overlapping parallel trees with heights ranging from 1 to 20. The IMT tree is the only one with height 20 and the SMTs have heights ranging from 1 to 19.

To visualize the structure, Figure 5.5 depicts a reduced DGM instance with an IMT, colored in blue, of height 4 and 42 SMTs, colored red, yellow and green. We assume a uniform distribution in the selection of the IMT internal nodes and a key is assigned from the linked SMT. Hence, each IMT internal node has the same number of assigned OTS keys, i.e. leaf nodes, and the number of SMTs per node in level j is double the number of SMTs per node in level $(j + 1)$. There are 4, 2, and 1 SMTs branching from internal IMT nodes at level 1, 2, and 3, respectively, and their respective index colors are green, yellow, and red. This simplified example has 112 signing leaves which can be used to sign 112 messages under the same public key (IMT root). Note that there is no maximum number of SMTs so if more signing leaves are needed, new SMTs are constructed and linked to a random internal node.

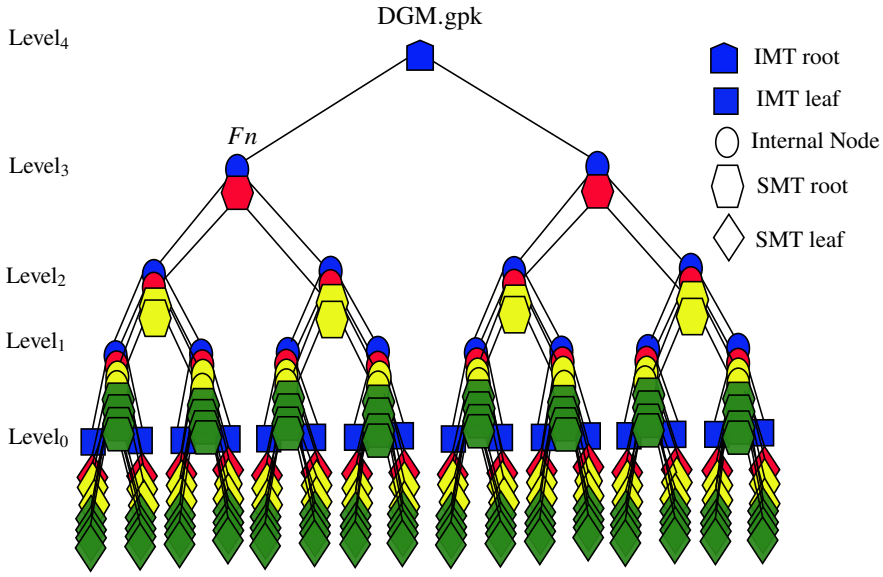


Figure 5.5: A simplified example of a DGM of height 4 with 42 SMTs, 112 signing leaves, and fallback nodes uniformly distributed across the internal IMT nodes.

Following the NIST PQC recommendation, a signature scheme should be secure to sign up to 2^{64} messages under the same public key [40]. Thus, in what follows we assume that DGM is used to sign 2^{64} messages. According to the design specifications, when a group

member needs B signing keys (leaves), the group manager randomly selects B IMT internal nodes and assigns to that member the next unassigned OTS of the SMTs linked to these internal nodes. The total number of internal nodes excluding the root in an IMT of height 20 is $2^{19}+2^{18}+\cdots+4+2 = 2^{20}-2$. Recall that if the SMT OTS leaves linked to a randomly chosen internal node are used up, then a new SMT tree is generated, linked to that fallback node and one of its leaves is assigned. Accordingly, assuming a uniform distribution in the random fallback node selection, to assign 2^{64} OTS to the group members, each IMT internal node is chosen $2^{64}/(2^{20}-2) > 2^{44}$ times. This means that each IMT internal node at level j, $1 \leq j \leq 19$, has $2^{44-(j-1)-1} = 2^{44-j}$ SMT trees each of height j, i.e. 2^{43} SMTs of height 1 for each IMT internal node at level 1, 2^{42} SMTs of height 2 for each IMT internal node at level 2, and up to 2^{25} SMTs of height 19 for each IMT internal node at level 19.

When DGM is instantiated with XMSS-T, to enable verification of a given signature, a DGM instance is seen as one tree of height 20 which means that regardless of the signing SMT location with respect to the IMT, the leaf indexing is in the set $\{0, 1, 2^{20}-1\}$, i.e. leaf indexing is considered relevant to the IMT where the signing SMT is considered a part of the IMT. This indexing is required to enable the verifier to evaluate the position of the nodes in the authentication path of the IMT up to its root (the pale blue nodes in Figure 5.3), which is essential in determining which nodes are concatenated on its right and left. Consequently, different SMTs that are linked to the same IMT internal node have the same indexing, and accordingly their parallel nodes at the same position are evaluated with the same hash function, i.e. the same hash randomizers and bit masks.

For instance, in Figure 5.5 any 4 green SMT roots branching from the same level 1 IMT blue node are evaluated with the same hash function as they share the same index within the IMT. Moreover, there are SMTs nodes that share the same indexes and nodes of SMTs

that are connected to upper IMT internal nodes, for example, any 4 green SMT roots in the figure from a level 1 IMT node share the same indexes with 2 yellow SMT internal nodes and one red SMT internal node. Therefore, even though XMSS-T is secure against multi-target attacks, employing several parallel instances of it with the same indexing in the form of SMTs makes DGM vulnerable to multi-target attacks. Intuitively, an adversary who collects a set of message-signature pairs can group these in t-target sets that share common indexes. Then they can find another message whose digest collides with one of the message digests in a set. Note that a set has t messages with authentication paths that share nodes with the same IMT indexes, so the forgery complexity is $2^n/t$.

5.4.3 DGM Bit Security

Consider that DGM is used to sign 2^y messages where $y > 20$. Accordingly, each internal IMT node is chosen $2^y/(2^{20}-2)$ times by the group manager to assign the next available OTS from the linked SMT. Assume an adversary $\mathcal{A}$ is able to collect all 2^y signatures generated by the scheme. The signature given by $\Sigma = (R, indx, OTS.\sigma_{indx}, DGM.pos_i, Auth)$ is signed with the i-th OTS key pair and has index $indx$ relative to its IMT position, i.e. $indx \in 0, 1, \ldots, 2^{20} - 1$ (see Section 5.2.3). $\mathcal{A}$ can then group the signatures along with their corresponding messages in sets that share the same signing index, $indx$, and each set is expected to have t target message-signatures pairs, so a given target set can be denoted by $ts = \{(M_0, \Sigma_0), (M_1, \Sigma_1), \ldots, (M_{t-1}, \Sigma_{t-1})\}$. Assuming a uniform distribution in selecting IMT Fn positions, the number of targets t per set is given by

$$t = \sum_{j=1}^{j=19} \frac{2^y/(2^{20}-2)}{2^j} < 2^{y-20} \tag{5.1}$$

We assume a fully filled tree similar to the example in Figure 5.5 where all IMT internal nodes have an equal number of assigned leaves, e.g. $2^y/(2^4-2) = 112/14 = 8$. Otherwise, the index that has the maximum signatures is considered. The maximum number of overlapping SMT nodes is then given by $\frac{8}{2} + \frac{8}{2^2} + \frac{8}{2^3} = 7$, hence $t = 7$.

To sign a message M in XMSS-T, its message digest md is initially calculated as $md = H_{msg}(R||DGM.root||indx, M)$ where $H_{msg} : \{H(K, M) : \{0,1\}^m \times \{0,1\}^* \rightarrow \{0,1\}^n$, R is the hash randomizer chosen by the signer, and $index$ is the leaf index relative to IMT. Since ts = has t (M, Σ) pairs all with the same $indx$, then $\mathcal{A}$ can search for a pair (M', R') such that $M' \notin ts$, and the corresponding md' collides with a message digest of one of the messages in ts. Specifically, $\mathcal{A}$ finds (M', R') such that

$$H_{msg}(R'||DGM.root||indx, M') \in \{(H_{msg}(R_0||DGM.root||indx, M_0)), \ldots, (H_{msg}(R_{t-1}||DGM.root||indx, M_{t-1})).\}$$

Thus, $\mathcal{A}$ can successfully find a forgery for (M', R') with probability $2^{-n+\log_2 t}$. Similar multi-target attacks can be applied on the OTS public keys or authentication paths in ts. In what follows, we give the reduction in DGM security when it is used to sign 2^y messages and $y > 20$. For completeness and consistency with the XMSS-T notation [14], the hash functions used in different contexts within the signature scheme are defined as follows.

- $F : \{F(K, M) : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$, used in OTS hash chains.
- $H : \{H(K, M) : \{0,1\}^n \times \{0,1\}^{2n} \rightarrow \{0,1\}^n$, used to calculate the Merkle tree hash nodes.
- $H_{msg} : \{H(K, M) : \{0,1\}^m \times \{0,1\}^* \rightarrow \{0,1\}^n$, used to calculate the message digest.

- $\mathcal{F}_n$ (resp. $\mathcal{F}_m$), a pseudorandom function family that takes a secret seed as input and outputs the OTS secret keys (resp. the message digest hash randomizer R) each of n bits (resp. m bits ($m = n + y$)).

Theorem 2. *For security parameter $n \in \mathbb{N}$ and parameters y, t as explained above, DGM is unforgeable against an adaptive chosen message attack if*

- *F, H, and Th are PQ-DM-SPR hash function families,*

- *$\mathcal{F}_n$ and $\mathcal{F}_m$ are post-quantum pseudorandom function families, and*

- *H_{msg} is PQ-NM-eTCR hash function family.*

The insecurity function, $\mathsf{InSec}^{\mathsf{PQ\text{-}forge}}(DGM, \xi, 2^y)$, that describes the maximum success probability over all adversaries running in time $\leq \xi$ against the PQ-forge security of DGM and making a maximum of $qs = 2^y$ queries is bounded by

$$\mathsf{InSec}^{\mathsf{PQ\text{-}forge}}(DGM, \xi, 2^y) \leq \mathsf{InSec}^{\mathsf{PQ\text{-}PRF}}(\mathcal{F}_n, \xi) + \mathsf{InSec}^{\mathsf{PQ\text{-}PRF}}(\mathcal{F}_m, \xi) +$$
$$max[t \cdot (\mathsf{InSec}^{\mathsf{PQ\text{-}DM\text{-}SPR}}(H, \xi) + \mathsf{InSec}^{\mathsf{PQ\text{-}DM\text{-}SPR}}(F, \xi) + \mathsf{InSec}^{\mathsf{PQ\text{-}NM\text{-}eTCR}}(H_{msg}, \xi))]$$

Proof. The proof is based on the approach of the proof given in [14]. Note that we do not include the proof of $\mathcal{F}_n$ and $\mathcal{F}_m$ with respect to PQ-PRF because they are not affected by instantiating DGM with XMSS-T, hence the proof is similar to that of XMSS-T in [14]. Assume an adversary $\mathcal{A}$ is allowed to make 2^y queries to a signing oracle running DGM with XMSS-T. $\mathcal{A}$ wins the game $\mathsf{EXP}^{forge}_{\mathcal{GS},\mathcal{A}}$ if they find a valid forgery (M', Σ') where the message M' is not in the queried set of 2^y messages. $\mathcal{A}$ initially groups the signatures that share a given *indx* in a set *ts*. Forgery occurs in the following three mutually exclusive cases.

- The message digest of M' performed under $indx$ results in M' being a second preimage of one of the message digests of the messages in ts. More precisely,

$$md = H_{msg}(R'||DGM.root||indx, M') = H_{msg}(R_j||DGM.root||indx, M_j)$$

where $M_j \in ts$. This occurs with success probability $t \times \mathsf{InSec}^{\text{PQ-NM-eTCR}}(H_{msg})$ (see Definition 10), i.e., $\mathcal{A}$ is able to break the security of NM-eTCR of the used hash function.

- The OTS public key of the forged signature, $OTS.pk'$, exists in the set of OTS public keys of the signatures in ts, i.e. $OTS.pk' \in \{OTS.pk_0, \cdots, OTS.pk_{t-1}\}$. This occurs with success probability $t \times \mathsf{InSec}^{\text{PQ-DM-SPR}}(\mathrm{F})$ (see Definition 6), i.e. $\mathcal{A}$ is able to break the security of DM-SPR of the used hash function F.

- The forged signature contains a node in the authentication path ($X'_{i,j}$ the i-th node in level j) that collides with a node at the same position in the set of authentication paths in ts ($X_{i,j}$ the i-th node in level j), e.g. $H(r_{i,j}, (X'_{2i,j-1}||X'_{2i+1,j-1}) \oplus q_{i,j}) = H(r_{i,j}, (X_{2i,j-1}||X_{2i+1,j-1}) \oplus q_{i,j})$ where the nodes $(X'_{2i,j-1}, X'_{2i+1,j-1})$ are from the forged signature authentication path, the nodes $(X_{2i,j-1}, X_{2i+1,j-1})$ are from an authentication path of a signature in ts, and $r_{i,j}, q_{i,j}$ are the hash randomizer and bit mask used for hashing. This occurs with success probability $t \times \mathsf{InSec}^{\text{PQ-DM-SPR}}(\mathrm{H})$ (see Definition 6). Thus, $\mathcal{A}$ is able to break the security of the second preimage resistance of the hash function H.

The above results show that if DGM is instantiated with the XMSS-T parameters in RFC 8391, i.e. the message digest length equals the security parameter n, then DGM does not achieve the same bit security as XMSS-T. In particular, the bit security of DGM decreases

by $\log_2 t$ bits compared to XMSS-T. For example, if XMSS-T has security parameter $n = 256$ and DGM is used to sign 2^{64} messages, then the DGM bit security decreases by $\log_2(\sum_{j=0}^{j=18} 2^{43-j}) = 44$ bits, i.e. DGM has only 212 bits of security. Therefore, if DGM is required to have bit security n, then XMSS-T should use a hash function with output size $n + \log_2 t$ which decreases the signing performance and increases the signature size. In the next section, we propose a solution that allows DGM to be optimal as with XMSS-T.

5.5 DGM$^+$ With Optimal Parameters

In this section we propose DGM$^+$, a DGM-XMSS-T variant that mitigates multi-target attacks (per index) as discussed in Section 5.4. We modify the addressing scheme such that it outputs a different hash randomizer and bit mask for the same hash call location in different SMTs branching from the same IMT internal node, and for overlapped SMTs that share the same indexing for some leaves.

The DGM public parameters are the IMT root (group public key) $DGM.root$ and the public key seed $DGM.seed$ that is used in the XMSS-T addressing scheme to generate the hash randomizers r_i and bit masks q_i for each hash call at address ad_i in the IMT, i.e. $(r_i, q_i) \leftarrow ADRS(DGM.seed, ad_i)$. To enable opening, each SMT leaf has a (v, u) index which is encrypted to generate $DGM.pos$ where v refers to the SMT and u refers to the leaf index within the SMT. Note that both u and v are secret but $DGM.pos$ is not because it is sent in the signature. If we assume that the bit size of v is equal to the block length, b, of the encryption algorithm used, then ev can be obtained as the first b bits of $DGM.pos$, where ev denotes the encrypted v. Accordingly, we propose the following.

- IMT uses $DGM.seed$ directly, as the seed to generate the hash randomizers and bit masks for each hash call within the IMT.

- Each SMT utilizes a (publicly calculated) different seed SMT.$seed_v$ for its hash randomizer and bit mask generation. SMT.$seed_v$ is unique for the v-th SMT and is calculated as SMT.$seed_v = PRF(\text{DGM}.seed, ev)$

For all SMTs that share indexing, we utilize different seed values with each SMT and keep the XMSS-T addressing scheme unchanged [65] (see Section A.5). Accordingly, different hash randomizers and bit masks are used at the same IMT location for different SMTs. Note that for signing, the IMT utilizes DGM.$seed$ in its construction, while the v-th SMT utilizes SMT.$seed_v = PRF(DGM.seed, ev)$ in its construction. Let $SMT.root.level$ denote the level of the fallback node for a given signing SMT. The signature authentication path, $Auth$, contains the entire SMT authentication path, $Auth.SMT_v$, and the top $20 - SMT.root.level$ nodes from the IMT. The latter authentication path starts from the neighboring node of the fallback node linked to the SMT root and up to DGM.$root$.

For verification, the verifier uses two seeds, $DGM.seed$ for hash evaluations of the authentication path from the fallback node and up, and STM.$seed_v = PRF(DGM.seed, ev)$ which is used in the WOTS-T hash iterations and the SMT authentication path, $Auth.SMT_v$, hash evaluations.

5.5.1 Message Hashing with DM-SPR

It was shown in Section 5.4 that DGM security depends on the nM-eTCR of the hash function where the number of targets, t, is considered per index. We modify the message hashing such that DGM security depends on the DM-SPR of the hash function (Definition 6), to prevent multi-targets attacks. This is achieved by using the message hash randomizer $R = F^{w-1}(sk_1)$ as follows

$$md = H_{msg}(R||DGM.root||idx, M) = H_{msg}(F^{w-1}(sk_1)||DGM.root||indx, M)$$

where $F^{w-1}(sk_1)$ is the last iteration, $w-1$, of the first secret key of WOTS-T (see [14] for the details of WOTS-T).

Rationale for message hashing modification The elements $(R||DGM.root||idx)$ serve as the hash key where R is chosen at random for each new message hashing, and $DGM.root$ is fixed. What if an adversary $\mathcal{A}$ who has access to the signing oracle is able to get the hash randomizer R before querying the signing oracle? Then, $\mathcal{A}$ can search to find two messages that have the same image using the same hash randomizer, R, i.e. $\mathcal{A}$ looks for a collision. Thus, $\mathcal{A}$ can query the signing oracle with one message and the other message will have the same signature. Nevertheless, as R is chosen randomly and it is known to the adversary just after querying the signing oracle, $\mathcal{A}$ can attempt to find a second preimage of any of the queried messages using a hash randomizer R', i.e. for a valid forgery the adversary needs to break the NM-eTCR security of the hash function used.

If we replace the hash randomizer R with the last iteration of the first secret key of the OTS $pk_1 = F^{w-1}(sk_1)$ (see [14] for details), which is known to the public only after signing, then R is not chosen at random but will be known to the public only after signing. Accordingly, for valid forgery, the adversary is restricted to use the same message hash randomizer, $R = F^{w-1}(sk_1)$, that is sent in the signature to find a message digest collision with the queried set. Hence, they are required to break the security of MM-SPR of the hash function which has a lower probability of success than breaking the NM-eTCR security of the hash function. Note that the last chain iteration of the first OTS secret key, $F^{w-1}(sk_1)$, is not a public parameter and is known only after signing with the corresponding leaf node, i.e. it is different than the OTS public key which is the root of L-tree, where the L-tree is a Merkle tree with l leaf nodes (last chain iterations of l OTS secret key, $F^{w-1}(sk_i)$ for $0 \leq i \leq l-1$), in which if the last node in the level does not have a sister node, it is lifted to the upper

level. As a result of the introduced modification of the message hashing, the message hash randomizer size is n bits which achieve the optimal parameter (each signature element is n bits) where the message hashing function is required to be PQ-MM-SPR (Definition 6). While XMSS-T achieves (Almost) optimal pentameter because its security depends on the PQ-NM-eTCR security (Definition 10) of the utilized message hashing function [27], in which it is still not optimal with regard to the required length of the hash randomizer, R, which has to be chosen as $R \geq n + \log_2 p$ bits, where p is the number of signatures that can be signed by the scheme.

In the verification procedure, the verifier checks if $pk_1 = F^{w-a_1-1}(\sigma_1)] \stackrel{?}{=} R$, where σ_1 is the first element in the OTS signature, otherwise it returns invalid signature. Accordingly, for a valid forgery the adversary is required to find a second preimage using the same hash key $F^{w-1}(sk_1)||DGM.root||idx$, so they are required to break the MM-SPR of the hash function (see Definition 6).

Note that using the given message hashing to generate R from the OTS public keys may enable XMSS-T [27] to attain optimal parameters. Specifically, when R is bound to a specific signing leaf, a length of n bits is sufficient to provide n bit security.

5.5.2 DGM and DGM$^+$ Comparison

This section provides a comparison between DGM and DGM$^+$ when both are instantiated with XMSS-T to provide n bit security and support 2^y messages where $y \geq 20$ and the IMT height is 20.

Secret and public keys sizes For DGM to achieve n bit security, the hash output size should be $n + \log_2 t$ bits where t is given by (5.1), so its tree nodes and secret keys will also have length $n + \log_2 t$ bits. A DGM public key is a pair $(pk.seed, IMT.root)$ each of length

$n + \log_2 t$ bits, and the secret key contains $sk.prf$ that is used to generate the message hash randomizer and $sk.seed$ that is used to generate the WOTS-T secret keys. Accordingly, the total secret key size is $2(n + \log_2 t)$ bits.

For DGM$^+$, the size of the tree nodes and secret keys is n bits. The DGM$^+$ public key size is $2n$ bits, i.e. $(pk.seed, IMT.root)$ each of n bits. The secret key contains only $sk.seed$ of length n bits because it does not require $sk.prf$ as the message hash randomizer is the last hash iteration of the first secret key WOTS-T.

Signature size The DGM signature contains the message hash randomizer, R, the leaf index, the encrypted position, the WOTS-T signature, the authentication path, and the fallback key. The signature element size in DGM$^+$ is n bits while in DGM it is $n + \log_2 t$ bits. Another effect is that the message digest size is increased by the number of WOTS-T elements, l, which increases both the signature size and the computational cost.

Table 5.1 provides the key and signature sizes for both DGM$^+$and DGM with 128, 192, and 256 bit security when they are used to support up to 2^{64} signatures where the signature size is $(22 + l)n + 4$ bytes and l is the number of elements in the OTS signature. The index is 4 bytes and we consider the encrypted position and the message hash randomizer, R, to have the same size as a node.

Table 5.1: DGM and DGM$^+$ keys and signature sizes (bytes) for 128, 192, and 256 bit security and 2^{64} signatures.

Algorithm	bit security	node size	pk	sk	l	signature size
DGM	128	22	44	44	47	1522
	192	30	60	60	63	2554
	256	38	76	76	79	3842
DGM$^+$	128	16	32	16	35	916
	192	24	48	24	51	1756
	256	32	64	32	67	2852

5.6 Conclusion

In this chapter, we discussed the challenges of instantiating GM and DGM using XMSS-T and provided a modification in the setup phases of both GM and DGM to overcome these challenges. Moreover, the bit security of DGM when instantiated with XMSS-T was analyzed. It was shown that the parallel multiple XMSS-T instances construction makes DGM vulnerable to multi-target attacks that may enable forgery with 44 bits less effort than multi-tree XMSS-T when the scheme is used to sign 2^{64} messages. Finally, a solution was proposed that mitigates multi-target attacks and a new message hashing mechanism was presented that reduces the associated signature and secret key sizes.

Chapter 6

$\mathsf{GM}^{\mathrm{MT}}$: A Revocable Group Merkle Multi-Tree Signature Scheme

In this chapter, we propose $\mathsf{GM}^{\mathrm{MT}}$, a revocable hash-based group signature scheme that solves some of the limitations of the current GM and DGM schemes. For GM, it is a one-layer Merkle tree construction which limits the maximum achievable tree height and thus restricts the maximum number of signatures that can be issued by the group under one public key, to 2^{20}. In [20], it was claimed that multi-tree approaches are not applicable for group hash-based schemes without justification and stated that the required storage for each member is a limiting factor. In addition, challenges to the practical adoption of DGM such as the fact that a verifier needs to interact with the group manager to ensure the validity of the signature were discussed [21]. Moreover, the revocation mechanism utilizes a puncturable encryption algorithm [66] for membership verification with a computational cost that is linear in the number of revoked signatures of the members. $\mathsf{GM}^{\mathrm{MT}}$ builds on GM and adopts a multi-tree construction that builds new GM trees to assign new signing leaves while keeping the group public key unchanged, thus growing the multi-tree structure adaptively

to support 2^{64} signatures. Moreover, $\mathsf{GM}^{\mathrm{MT}}$ has a revocation mechanism with linkable anonymity of revoked signatures and logarithmic verification computational complexity compared to the linear complexity of DGM. For opening and revocation, the $\mathsf{GM}^{\mathrm{MT}}$ group manager requires storage that is linear in the number of members while the corresponding storage in DGM is linear in the number of signing leaves supported by the system.

6.1 Preliminaries

A Group Signature Scheme (GSS) is a tuple of five polynomial-time algorithms $\mathcal{GS} = (GKGen, GSign, GVerify, GRevoke, GOpen)$. The specifications of these procedures are as follows.

- $GKGen(1^n, 1^N)$: The Group key generation algorithm takes as input the security parameter n and the number of the group members N. It outputs the group public key GPK, the group members secret keys sk_i for $1 \leq i \leq N$, and the group manager secret key sk_{gm} that is used to reveal the signer identity.

- $GSign(M, sk_i)$: The group signing algorithm takes as inputs a message M, the group member secret key sk_i. The algorithm outputs the signature σ of the input message.

- $GVerify(\Sigma, M, GPK, RevList)$: The group verification algorithm, it is a deterministic algorithm that takes as input the signature Σ along with the corresponding message M, the group public key GPK, and the revocation list $RevList$. It outputs 1 for valid signature and 0 otherwise.

- $GRevoke$: The revocation algorithm updates the revocation list based on the revoked members (signatures) to revoke their abilities to generate valid signatures.

- $GOpen(\Sigma, sk_{gm})$: The open algorithm takes as input the signature Σ, and the group manager secret key sk_{gm}, and outputs the identity of the signer.

In what follows, we provide definitions of the standard security notions for analyzing group signature schemes.

Definition 18. *(Correctness). A group signature scheme $\mathcal{GS}$ with a group public key GPK achieves correctness if for an honest signer with a secret key sk_i*

$$\Pr[GVerify(GSign(M, sk_i), M, GPK) = 0] < negl(n)$$

Other notions that capture the required GSS security include unforgeability, anonymity, unlinkability, collusion resistance, exculpability, and framing resistance. It was shown in [62] that full-anonymity and full-traceability ensures that a given GSS achieves all the aforementioned security requirements. The notion of full-anonymity [62] is very strong as it assumes that an adversary has access to the secret keys of all members and the group manager. Camenisch and Groth introduced a relaxed type of anonymity in which an adversary cannot corrupt the group manger and at least two group members, i.e., challenge identities in the anonymity experiment in Appendix A.6. In our scheme, we follow the anonymity notion introduced by Camenisch and Groth [63] because in our scheme, only secret keys of the group manager are used to reveal signer identities, and knowledge of the signing keys along with the associated signatures also uncovers the corresponding identities. Such a security notion is formally defined in $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}(n, N)$ in Appendix A.6. Hence in our analysis, we focus on the following two security definitions.

Definition 19. *(Anonymity [63]). A group signature scheme $\mathcal{GS}$ achieves anonymity if a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ who is not the group manager but has*

access to the signing and opening oracles and is able to corrupt all but two group members i_0 and i_1, is not able to reveal the identity of the signer when challenged with a signature of a message that is signed by either i_0 or i_1. $\mathcal{A}$ has a negligible advantage in the experiment $Exp_{\mathcal{GS}}^{Anon-b}$ (see Appendix A.6), where $b = \{0, 1\}$ denotes the index of the identity of the signer

$$Adv_{\mathcal{GS},\mathcal{A}}^{Anon-b}(n, N) = |\mathsf{Pr}[Exp_{\mathcal{GS},\mathcal{A}}^{Anon-0}(n, N) = 1] - \mathsf{Pr}[Exp_{\mathcal{GS},\mathcal{A}}^{Anon-1}(n, N) = 1]\ | \leq negl(n)$$

Definition 20. *(Full-traceability [62]). A group signature scheme $\mathcal{GS}$ satisfies full-traceability if a PPT adversary $\mathcal{A}$ that is given unrestricted access to the signing and opening oracles and is able to corrupt some of the group members is not able to generate a valid signature which cannot be opened or traced back by the group manager to an uncorrupted member. $\mathcal{A}$ has a negligible advantage in the experiment $Exp_{\mathcal{GS},\mathcal{A}}^{Full-Trace}$ as defined in Appendix A.6*

$$Adv_{\mathcal{GS},\mathcal{A}}^{Full-Trace}(n, N) = |\ \mathsf{Pr}[Exp_{\mathcal{GS},\mathcal{A}}^{Full-Trace}(n, N) = 1]\ | \leq negl(n)$$

6.2 $\mathsf{GM}^{\mathrm{MT}}$ Hash-Based Group Signature Scheme

$\mathsf{GM}^{\mathrm{MT}}$ is a revocable hash-based group signature scheme that is constructed using a multi-tree approach and utilizes a One-Time Signing Scheme (OTS) as the underlying signing scheme. It is designed as a generic construction such that any stateful hash-based Merkle

signing scheme with an OTS leaves can be used. $\mathsf{GM}^{\mathrm{MT}}$ provides a flexible setup phase where the group manager generates the group public key independent of the parameters of the group members (OTS public keys and their indexes). Figure 6.1 shows that $\mathsf{GM}^{\mathrm{MT}}$ can be regarded as a hybrid construction that encompasses several Group Merkle (GM) signature trees (denoted by clusters) at layer 0, and one stateful hash-based signature scheme consuming all higher layers, i.e., layers 1 to $d-1$. Each GM tree at layer 0 contains a subset of the OTSs of all group members while the multi-tree stateful hash-based signature scheme is used by the group manager to sign the roots of the GM trees at layer 0. The group public key, GPK, is the root of the top layer tree which is generated using the group manager's secret key. Such a construction allows layer 0 GM trees to be constructed incrementally as the signing leaves are used up. Specifically, all group members signing leaves are clustered into GM trees where each GM tree has a subset of the signing leaves of all members. This allows the group manager to manage leaf assignment for all members in a clustered manner. Hence, the scheme enables a practical setup phase with less storage requirements for each group member compared to GM [20] because not all the signing leaves for each group members have to be assigned, and a member can reuse the storage that was allocated to their used leaves. Table 6.1 gives the parameters and notation used in the specification of $\mathsf{GM}^{\mathrm{MT}}$.

In the following, we give detailed specifications of the setup, signing ,verifying, membership revocation, and opening procedures in $\mathsf{GM}^{\mathrm{MT}}$. An algorithmic description of these procedures is provided in Algorithm 6.1. Without loss of generality, in the specification of the scheme procedures, we assume that WOTS^{+} (See Appendix A.7 for details), is the OTS scheme.

Table 6.1: $\mathsf{GM}^{\mathrm{MT}}$parameters and notation.

n	security parameter
N	initial number of group members
B	initial number of signing indexes for each group user
BC_{max}	maximum number of signing leaves for a member in a GM tree (cluster)
Bu_{max}	maximum number of signing leaves for a member in the scheme
d	number of tree layers
h	maximum tree height
h_c	GM/cluster tree height
h_{gm}	group manager tree height, $h_{gm} = h - h_c$
w	Winternitz WOTSOTS parameter
l	number of elements, each of length n bits, in the Winternitz WOTS signature
GPK	group public key which is the root of the top layer tree

6.2.1 Setup Phase and Key Generation

The setup phase is an interactive procedure that involves communication between the group members and group manager for signing leaves assignment. However, since $\mathsf{GM}^{\mathrm{MT}}$ is a multi-tree structure, the group public key is computed by the group manager independent of the inputs from members. Hence, the setup phase is divided into two procedures, group public key generation and signing leaves assignment. The former is performed once during initial group setup while the latter is repeated periodically with the addition of new cluster trees at layer 0.

Key Generation Algorithm. The algorithm randomly samples the secret keys $SK = (sk.enc_{gm}, sk.seed_{gm}) \in \{0,1\}^n \times \{0,1\}^n$, where $sk.enc_{gm}$ is the group manager encryption secret key that is used to reveal the signer identity and $sk.seed_{gm}$ is used to generate the trees of the multi-tree signing scheme, e.g., XMSS scheme [14], at layers 1 to $d-1$ (the top layer). Each tree has height $h_{gm}/(d-1)$. In an actual instantiation, $sk.seed_{gm}$ may be used in a manner similar to the random secret seed in [14]. The root of the top layer tree is the group public key GPK.

Signing leaves assignment. This procedure adds a new GM cluster tree containing a subset of the signing leaves of all N members to the construction. The trees at layer 0 are GM trees, each of height $h_c = h - h_{gm}$, and the first tree (cluster 0), contains B signing leaves of each group members so there are $NB = 2^{h_c}$ signing leaves in total. Note that each cluster tree contains an equal number of signing leaves for each member. However, $\mathsf{GM}^{\mathrm{MT}}$allows revocation and h_c is a constant, so the number of leaves assigned per member in the i-th cluster, $0 < i < 2^{h_{gm}}$, B, may increase because N may decrease. The assignment procedure is the interactive part of the setup phase and involves the following three steps.

- *Label Assignment*. The group manager sets the maximum number of leaves that can be assigned to a member for the lifetime of the scheme, and assigns to each member a sequence of numbers corresponding to their identity, denoted as labels. Specifically, let $BC_{max} > B$ be the maximum number of leaves that can be assigned to a group member in a cluster, so the maximum number of signatures that a group member can sign is $Bu_{max} = 2^{h_{gm}} \times BC_{max}$. Consequently, the i-th group member is assigned Bu_{max} labels denoted by $b_{0,i}, b_{1,i}, \ldots, b_{Bu_{max}-1,i} = iBu_{max}, iBu_{max} + 1, \ldots, iBu_{max} + Bu_{max} - 1$ where $0 \leq i \leq N-1$. Since $\mathsf{GM}^{\mathrm{MT}}$ provides member revocation, BC_{max} is chosen to be greater than B to simplify label assignment, so all labels dedicated to a member may not be assigned. Hence, we use the term label to differentiate from a cluster signing leaf index because unlike indexes, not all labels may be assigned. However, each cluster leaf signing index assigned to a member is associated with a label in the dedicated range. Finally, the group manager stores the last assigned label for each group member in the users list, $UList$. Henceforth, the last assigned label of the i-th member is denoted by $la = UList[i]$ and it is used to evaluate their identity by $\lfloor UList[i]/Bu_{max} \rfloor = i$. When a new cluster is being generated, the group manager retrieves the last assigned label, $la = UList[i]$, for each group member,

i, and a new range of labels, B, is dedicated to their new cluster signing leaves starting from the next value from the last stored label. More precisely, for a new cluster, the i-th member is given B labels $b_{0,i}, b_{1,i}, \ldots, b_{B-1,i} = UList[i] + 1, UList[i] + 2, \ldots, UList[i] + B$. The group manager then updates the stored label in $UList$ with the last label in the new range, i.e., $UList[i] = UList[i] + B$.

- *Signing keys generation*. Each group member, i, generates B WOTS public keys $(pk_{0,i}, pk_{1,i}, \ldots, pk_{B-1,i})$ using their own secret key sk_i, and sends them to the group manager, where $pk_{j,i}$ denotes the j-th public key of the i-th group member within a cluster for $0 \leq i \leq N-1$ and $0 \leq j \leq B-1$.

- *Shuffling and clustering*. The group manager retrieves the last assigned label for each group member and assigns the next set of labels to their public keys (the cluster leaves), in ascending order i.e. $(pk_{0,0}, b_{0,0}), (pk_{1,0}, b_{1,0}), \ldots, (pk_{B-1,0}, b_{B-1,0}), \ldots\ldots, (pk_{0,N-1}, b_{0,N-1}), (pk_{1,N-1}, b_{1,N-1}), \ldots, (pk_{B-1,N-1}, b_{B-1,N-1})$, where $pk_{j,i}$ is the j-th public key of group member i, and $b_{j,i}$ is the corresponding label for $0 \leq i \leq N-1$ and $0 \leq j \leq B-1$. The group manager then updates the last assigned label for each member.

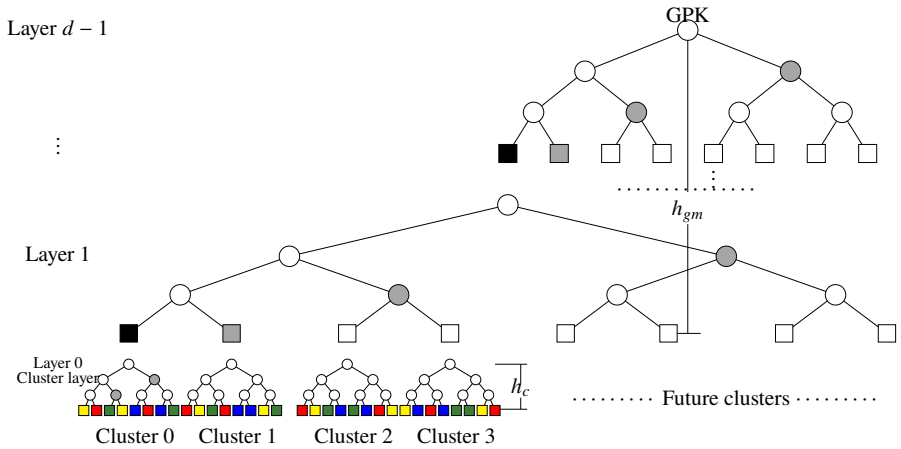


Figure 6.1: A simplified example of the $\mathsf{GM}^{\mathrm{MT}}$ initial setup phase. The gray nodes and the first red node in cluster 0 are the authentication path for signing with the first yellow leaf in cluster 0, while the black leaves are the group manager signing leaves.

Let $E(k, M)$ denote a symmetric encryption of a plaintext M using the key k. The group manager encrypts the labels assigned to the members by $sk.enc_{gm}$ and generates the corresponding encrypted labels $(Eb_{0,0}, \ldots, Eb_{B-1,0}), \ldots\ldots, (Eb_{0,N-1}, \ldots, Eb_{B-1,N-1})$, where $Eb_{j,i} = E(sk.enc_{gm}, b_{j,i})$. The group manager then generates the cluster leaves, $L_{0,0}, L_{1,0}, \ldots, L_{B-1,0}, \ldots\ldots, L_{0,N-1}, L_{1,N-1}, \ldots, L_{B-1,N-1}$, by hashing the concatenation of each group member public key and its corresponding encrypted label, i.e. $L_{j,i} = H(pk_{j,i}||Eb_{j,i})$ is the j-th leaf node of group member i. Next, the group manager permutes the group members leaves by reordering their encrypted labels in ascending order. Then, the group manager builds the cluster tree, and signs its root, $root_c$, by the corresponding upper tree leaf node and this continues until the top layer. Finally, the group manager broadcasts to the group members 2^{h_c} tuples of the encrypted labels, cluster tree leaves, and the corresponding signature of its root. Each member identifies their leaf nodes using their public keys and the corresponding encrypted labels.

After a specific time determined by the group manager, by which the group members are expected to have used up almost all their current cluster leaves, the signing leaves assignment procedure is repeated and a new cluster is generated. This is continued until the last cluster is constructed. Figure 6.1 depicts a simplified example of the initial setup phase. It shows a d layer $\mathsf{GM}^{\mathrm{MT}}$ with 4 clusters in the bottom layer. It is assumed that the group has $N = 4$ members and each member has two signing leaves colored blue, green, yellow, and red in each cluster. $Cluster_1$ is generated after some time period (when the $Cluster_0$ leaves are almost all used up), to provide new signing leaves to the members.

6.2.2 Signing Algorithm

The signing algorithm takes as input a message M of arbitrary length, the signer's secret key (sk_i), which contains the state of the signer (i-th member), which is the signing index t. The algorithm outputs the signature Σ that contains the WOTS signature $\sigma_{w,0}$ of the message M (see Appendix A.7 for the WOTS signing procedure), and the corresponding authentication path $Auth_0 = (Eb, A_{0,0}, A_{0,1}, \ldots, A_{0,h_c-1})$ from the cluster tree in layer 0. Moreover, Σ contains the signature of the group manager on the cluster root $\sigma_{w,j}, Auth_j$ for $1 \leq j \leq d-1$ where $\sigma_{w,j}$ is the WOTS signature of the lower layer tree root, $j-1$, and $Auth_j$ is the corresponding authentication path $Auth_j = (A_{j,0}, A_{j,1}, \ldots, A_{j,\frac{hgm}{d-1}-1})$. The $\mathsf{GM}^{\mathrm{MT}}$ signature is then given by $\Sigma = \sigma_{w,0}, Auth_0, \ldots, \sigma_{w,d-1}, Auth_{d-1}$.

6.2.3 Verification Algorithm

The verification algorithm takes as input the message M, the signature Σ, the public key GPK, and the revocation list $RevList$. It first checks if the received signature has been revoked (see Sec. 6.2.4). If the signature has not been revoked, the algorithm continues with verification by calculating the WOTS public key, pk', from the message digest and the signature element $\sigma_{w,0}$ (see Appendix A.7 for WOTS details). Next, the leaf node is calculated by hashing the concatenation of this WOTS public key and the signature element Eb of $Auth_0$, i.e., $L' = H(pk'||Eb)$. Then, the leaf node and $(A_{0,0}, A_{0,1}, \ldots, A_{0,h_c-1})$ from $Auth_0$ are used to calculate the cluster root that is used with $\sigma_{w,1}$ to get the WOTS public key at layer 1. Next, this public key along with the authentication path $Auth_1$ are used to calculate the tree root at layer 1. This procedure is repeated until the top layer tree root is calculated, GPK'. If it is equal to the public root, $GPK' = GPK$, the algorithm outputs 1 for a valid signature, and 0 otherwise.

6.2.4 Revocation Algorithm

The group manager retrieves the last assigned label of the revoked i-th member, $la = UList[i]$, and then regenerates all the encrypted labels which were assigned to that member, i.e., for the i-th member, the manager generates $E(sk.enc_{gm}, iBu_{max})$, $E(sk.enc_{gm}, iBu_{max}+1), \ldots, E(sk.enc_{gm}, l_a)$. The generated encrypted labels are added to the revocation list, $RevList$, which is then permuted using a Quicksort algorithm so that successive entries in the revocation list are not grouped by members.

Revocation Check: The verifier checks if the received signature is revoked or not by first extracting the encrypted label, Eb, from the signature and checking if it exists in the revocation list, $RevList$. If $Eb \in RevList$, then the received signature has been revoked, otherwise the verifier continues the verification procedure.

6.2.5 Opening Algorithm

The opening algorithm takes as input a message M, a signature Σ, and the group manager secret key $sk.enc_{gm}$, and outputs the identity of the signer i. The algorithm first decrypts the signature element Eb to recover the label $b = D(sk.enc_{gm}, Eb)$. Next, the manager calculates the member's identity $i = \lfloor b/Bu_{max} \rfloor$ and checks that b is less than the last assigned label to the i-th group member, $b \leq Ulist[i]$, if not it aborts.

Algorithm 6.1 $\mathsf{GM}^{\mathrm{MT}}$ Algorithm

Setup Phase

Input: $n, N, d, h_{gm}, h_c, BC_{max}$

$sk.seed_{gm} \xleftarrow{R} \{0,1\}^n$

$sk.enc_{gm} \xleftarrow{R} \{0,1\}^n$

Generate the top layer tree (using $sk.seed_{gm}$) where its root is the group public key GPK

$Bu_{max} = h_{gm} \times BC_{max}$

for $0 \le i \le N-1$ **do**

$UList[i] = i \times Bu_{max}$

end for

Each user $id_i \in \{0, 1, \ldots, N-1\}$ generate their key sk_i

$(pk_{i,0}, \ldots, pk_{i,B-1}) \leftarrow id_i(sk_i)$

return: $sk.seed_{gm}, sk.enc_{gm}, GPK, UList$

Cluster Generation

Input: $N, h_c, UList, sk.enc_{gm}, Bu_{max}$, and the NB WOTS public keys of the N members

for $0 \le i \le N-1$ **do**

$b = UList[i] + 1$

for $0 \le j \le B-1$ **do**

$Eb_{i,j} \leftarrow E(sk.enc_{gm}, b+j)$

$list[iB+j] \leftarrow H(pk_{i,j} || Eb_{i,j})$

end for

$UList[i] = b + B - 1$

end for

$Cluster \leftarrow Quicksort(list)$

Build the cluster tree by $Cluster$ and get its root $root_c$

$\sigma_{root_c} \leftarrow Sign(root_c, sk.seed_{gm})$,

Each group member, i, get their $param_i$ (encrypted labels, leaves, Auth. paths and σ_{root_c}).

Signing Algorithm

Input: $M, param_i, sk_i, state_i$

$\sigma_{w,0} \leftarrow WOTS.Sign(M, sk_i, state_i)$

$(Eb, Auth_0) \leftarrow (param_i, state_i)$

$\mathsf{GM}^{\mathrm{MT}}.\Sigma = M, indx, \sigma_{w,0}, Eb, Auth_0, \sigma_{Root_c}$

$state_i = state_i + 1$

return: Σ

Verification Algorithm

Input: $M, GPK, \Sigma, RevList$

if $Eb \in RevList$ **then**

Output 0

else

calculate the cluster root $root'_c$ using $indx, \sigma_{w,0}, Eb, Auth_0$

calculate the top layer tree root GPK' using $root'_c$, and σ_{root_c}

if $GPK' = GPK$ **then**

return 1

else

return 0

end if

end if

Revocation Algorithm

Input: $UList, Bu_{max}, RevList, sk.enc_{gm}, i$

$j = UList[i]$

while $j \ge i \times Bu_{max}$ **do**

Add $E(sk.enc_{gm}, j)$ to $RevList$

j^{--}

end while

$RevList \leftarrow Quicksort(RevList)$

return $RevList$

Opening Algorithm

Input: $\Sigma, sk.enc_{gm}, Bu_{max}, N, UList$

$b' \leftarrow D(sk.enc_{gm}, Eb)$

if $b' \ge N \cdot Bu_{max} \vee b' > Ulist[\lfloor b'/Bu_{max} \rfloor]$ **then**

return $\perp$

else

return $\lfloor b'/Bu_{max} \rfloor$

end if

6.2.6 Recommended Parameters

$\mathsf{GM}^{\mathrm{MT}}$ parameterization follows the NIST PQC requirements which state that a given signing key pair should produce up to 2^{64} signatures while maintaining the claimed security [67]. Thus, we recommend that $\mathsf{GM}^{\mathrm{MT}}$ be a four layer ($d = 4$) tree construction where the tree height in the bottom layer (clusters), h_c, has three possible values, $h_c = \{16, 18, 20\}$, depending on the number of group members and their signing requirements and storage capabilities. The height of the group manager trees in layers 1 to 3 is 16. The $\mathsf{GM}^{\mathrm{MT}}$ signature size depends on the required security level. More precisely, the $\mathsf{GM}^{\mathrm{MT}}$ signature size is $d \times l + h + 2$ elements, each of length n bits, where n is the security parameter, $n = \{128, 192, 256\}$, and $l = \{35, 51, 67\}$, respectively, is the number of WOTS signature elements (see Appendix A.7 for the WOTS signing algorithm). Table 6.2 gives our recommended parameters for $\mathsf{GM}^{\mathrm{MT}}$ such that it supports at least 2^{64} signatures under the same group public key and the corresponding signature size in bytes (B).

Table 6.2: $\mathsf{GM}^{\mathrm{MT}}$ recommended parameters and signature sizes.

Instance	bit security	d	h	h_c	h_{gm}	N	B	l	w	Signature (B)
$\mathsf{GM}^{\mathrm{MT}}$-128a	128	4	64	16	48	$2 < N \leq 2^6$	$2^{10} < B \leq 2^{15}$	35	16	3296
$\mathsf{GM}^{\mathrm{MT}}$-128b	128	4	66	18	48	$2^6 < N \leq 2^{10}$	$2^8 < B \leq 2^{12}$	35	16	3328
$\mathsf{GM}^{\mathrm{MT}}$-128c	128	4	68	20	48	$2^{10} < N \leq 2^{16}$	$2^4 < B \leq 2^{10}$	35	16	3360
$\mathsf{GM}^{\mathrm{MT}}$-192a	192	4	64	16	48	$2 < N \leq 2^6$	$2^{10} < B \leq 2^{15}$	51	16	6480
$\mathsf{GM}^{\mathrm{MT}}$-192b	192	4	66	18	48	$2^6 < N \leq 2^{10}$	$2^8 < B \leq 2^{12}$	51	16	6528
$\mathsf{GM}^{\mathrm{MT}}$-192c	192	4	68	20	48	$2^{10} < N \leq 2^{16}$	$2^4 < B \leq 2^{10}$	51	16	6576
$\mathsf{GM}^{\mathrm{MT}}$-256a	256	4	64	16	48	$2 < N \leq 2^6$	$2^{10} < B \leq 2^{15}$	67	16	10688
$\mathsf{GM}^{\mathrm{MT}}$-256b	256	4	66	18	48	$2^6 < N \leq 2^{10}$	$2^8 < B \leq 2^{12}$	67	16	10752
$\mathsf{GM}^{\mathrm{MT}}$-256c	256	4	68	20	48	$2^{10} < N \leq 2^{16}$	$2^4 < B \leq 2^{10}$	67	16	10816

6.3 Security Analysis

In this section we show that $\mathsf{GM}^{\mathrm{MT}}$satisfies the security requirements of correctness, anonymity [63], and full-traceability [62]. We also analyze the security of the proposed

revocation mechanism and discuss the drawbacks of adopting a dynamic approach.

Theorem 3. ***(Correctness)*** *Let GM^{MT} be the multi-tree group Merkle signature algorithm described in Sec. 6.2. Then GM^{MT} achieves correctness as defined in Definition 18.*

Proof. $\mathsf{GM}^{\mathrm{MT}}$utilizes a multi-tree Merkle signing scheme for generating signatures and only uses extra shuffling and clustering procedures to assign the signing leaves to different members. Thus, the correctness of $\mathsf{GM}^{\mathrm{MT}}$ is achieved by the correctness of the underling Merkle signature scheme.

Theorem 4. ***(Anonymity)*** *Let GM^{MT} be the multi-tree group Merkle signature algorithm provided in Sec. 6.2 with secure hash function H and encryption algorithm E. Then GM^{MT} achieves anonymity for each cluster as defined in Def. 19.*

Proof. We adopt the $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ game (see Appendix A.6) on the group members. The proof follows the strategy in [20]. Assume that each group member is assigned B signing leaves in each cluster, i.e., each group member is assigned a total of $B \times 2^{h_{gm}}$ signing leaves over all clusters. An adversary $\mathcal{A}$ is given access to the signing and opening oracles, and can corrupt some group members. Assume there are only two members i_0 and i_1 that are uncorrupted. Moreover, $\mathcal{A}$ queries the signing and opening oracles for a maximum of $2^{h_{gm}} \times (B-1)$ messages for each uncorrupted member such that the signing oracle replies with $B-1$ signatures from each cluster for the two members, i.e., each member has the ability to sign at least one more message with a leaf from any cluster of the $2^{h_{gm}}$ clusters. Recall that the opening oracle when queried by a signature Σ replies with the decryption of the encrypted label Eb in the signature, $b = D(sk.enc_{gm}, Eb)$, which directly reveals the signing identity i. Thus, $\mathcal{A}$ has $B-1$ labels and their corresponding ciphertext pairs (b_{j,i_g}, Eb_{j,i_g}) for each group member i_g from each cluster where $g = \{0,1\}$ and $Eb_{j,i_g} = E(sk.enc_{gm}, b_{j,i_g})$, $0 \leq j \leq B-2$.

$\mathcal{A}$ queries the signing oracle with an arbitrary message M of their choice such that the signing oracle replies with the signature for either i_0 or i_1. From this signature, $\mathcal{A}$ retrieves the encrypted label Eb_{B-1,i_g}. Moreover, they are able to determine the signing cluster, and thus the corresponding $B-1$ label-encrypted label pairs (b_{j,i_g}, Eb_{j,i_g}), $0 \leq j \leq B-2$, for each group member i_g collected in the query phase. Then, $\mathcal{A}$ is required to correctly guess the identity of the signer. Since the labels for each group member are set sequentially, and $\mathcal{A}$ knows the first $B-1$ labels for each group member, then $\mathcal{A}$ knows with certainty the B-th labels for both group members, i.e., b_{B-1,i_0} and b_{B-1,i_1}. Accordingly, $\mathcal{A}$ must determine which label is the plaintext corresponding to the encrypted label Eb_{B-1,i_g} received in the queried signature. In other words, the adversary needs to win a distinguishability game that distinguishes the encryption of different plaintexts. As the encryption algorithm used is semantically secure, $\mathcal{A}$ has a negligible advantage in winning the $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ game.

Theorem 5. ***(Full-traceability)*** *Let GM^{MT} be the multi-tree group Merkle signature algorithm specified in Sec. 6.2 with secure hash function H, encryption algorithm E, and an underlying existentially unforgeable Merkle signing scheme. Then, GM^{MT} achieves full-traceability as in Definition 20.*

Proof. Recall that the group manager opens a signature by decrypting the encrypted label Eb in the signature. Assume that an adversary $\mathcal{A}$ collects all the signatures from all clusters. i.e., $\mathcal{A}$ knows (Eb_t, pk_t) where pk_t is the WOTS public key at leaf index t for $0 \leq t \leq 2^h - 1$. Assuming $\mathcal{A}$ corrupts a set of members C, then $\mathcal{A}$ wins the traceability game $Exp^{Full-Trace}_{\mathcal{GS},\mathcal{A}}$ in Appendix A.6 if they are successful in either of the following scenarios.

- $\mathcal{A}$ generates a valid signature of the i-th member where $i \in N \wedge i \notin C$. Since opening a signature depends on the signature element Eb, then $\mathcal{A}$ should include in the signature an element $Eb^{\star}$ from one of the signatures of any of the uncorrupted members so that

it decrypts to a valid label assigned to an uncorrupted member. Furthermore, $\mathcal{A}$ should pair $Eb^{\star}$ with one of the WOTS public keys of a corrupted member so that they can sign using the corresponding secret key. More precisely, $\mathcal{A}$ must find a pair $(pk_{j,i_c}, Eb^{\star})$ that is a second preimage of the pair (pk_{j,i_c}, Eb_{j,i_c}), i.e., $H(pk_{j,i_c}||Eb^{\star}) = H(pk_{j,i_c}||Eb_{j,i_c})$ where pk_{j,i_c} is the j-th WOTS public key of a corrupted member i_c and Eb_{j,i_c} is the corresponding encrypted label. The existence of such an adversary contradicts the assumption of a secure hash function. Conversely, $\mathcal{A}$ does not use any of the WOTS public keys of the corrupted members, but rather uses some $Eb^{\star}$ with a forged signature of the underlying Merkle signature scheme such that it passes verification and then decrypts to a valid assigned label. However, this contradicts the existential unforgeability assumption of the underlying signature scheme.

- $\mathcal{A}$ generates a valid signature which the group manager cannot open. In this case, $\mathcal{A}$ includes in the signature an encrypted label Eb' that is not equal to any of the valid encrypted labels which were collected in the query phase. Then following the steps in the previous scenario, $\mathcal{A}$ needs to either pair Eb' with a WOTS public key of a corrupted member, or include it with a forgery of the underlying signature scheme. In both cases, the existence of $\mathcal{A}$ contradicts the assumptions of a secure hash function and an existentially unforgeable signing scheme.

6.3.1 Revocation Security

For revoking a member with identity i, our revocation mechanism updates a revocation list, $Revlist$, by adding the member's encrypted labels that were assigned to their signing leaves, i.e., $Eb_{0,i}, Eb_{1,i}, \ldots, Eb_{la-iBu_{max},i} = E(sk.enc_{gm}, iBu_{max}), E(sk.enc_{gm}, iBu_{max} + 1), \ldots, E(sk.enc_{gm}, la)$, where $la = Ulist[i]$ denotes the last assigned label. Each of these

encrypted labels is part of a signature. Hence, an adversary $\mathcal{A}$ is able to recover the new set of encrypted labels that is added to $Revlist$ with updates by comparing the contents of $Revlist$ before and after the update. If $\mathcal{A}$ has collected signatures generated by the system before an update of the revocation list, then $\mathcal{A}$ can check if the encrypted labels in some of the collected signatures are in the newly revoked set. Accordingly, if such a set belongs to one revoked member, then $\mathcal{A}$ is able to link these signatures to the same revoked member. Otherwise, the signatures are for more than one revoked member and $\mathcal{A}$ is required to distinguishes the signatures over a small anonymity set (the newly revoked members). In all cases, only the encrypted labels of the revoked members are added to the revocation list, hence, it is infeasible to reveal the identities associated with these labels because they are encrypted. Note that if $\mathcal{A}$ is given only the last updated revocation list, then $\mathcal{A}$ cannot distinguish the newly revoked signatures from the old ones, and hence cannot link a set of signatures to one signer.

Theorem 6. ***(Revocation)*** *Let GM^{MT} be the multi-tree Merkle group signature algorithm provided in Sec. 6.2 with secure hash function H and encryption algorithm E. Then, GM^{MT} maintains the anonymity of revoked members and linkability of their signatures.*

Proof. Assume an adversary $\mathcal{A}$ has the previous and current states of the revocation list, and a set of signatures that has been collected between two updates of the revocation list. Then, $\mathcal{A}$ is able to recover the set of newly revoked signatures by running the revocation check on the collected signatures against the previous and current states of $Revlist$. If the update of $Revlist$ corresponds to revoking one member, then $\mathcal{A}$ is able to link these revoked signatures to this member without revealing their identity. However, if the current states are updated by revoking more than one member, then we adopt an anonymity game for the revoked members which can be seen as a variant of the $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ game that allows

$\mathcal{A}$ to be challenged with a set of revoked encrypted labels instead of a signature of their choice. $\mathcal{A}$ wins the game if they are able to attribute a subset of the challenge set to a given revoked signer out of two possible revoked signers. Precisely, $\mathcal{A}$ is given access to the opening algorithm for $B-1$ signatures from each cluster signed by each of two newly revoked members, i.e., $\mathcal{A}$ gets $B-1$ (label, encrypted label) pairs from each cluster for each revoked member. Then, they are challenged with the B-th encrypted label from each cluster for each revoked member and are required to determine which encrypted label belongs to which set of $B-1$ (label-encrypted label) pairs. If $\mathcal{A}$ is able to attribute the challenge encrypted labels to a given signer, then they can build another adversary that is able to distinguish between ciphertexts corresponding to a given plaintext, which contradicts the assumption of a secure encryption algorithm.

6.3.2 Security of Dynamic $\mathsf{GM}^{\mathrm{MT}}$

Our scheme can be adapted to allow adding new members at each cluster generation. In this case, the number of leaves assigned to each group member decreases because the maximum number of leaves in a cluster is 2^{20} and the number of group members is increased. A drawback of dynamic $\mathsf{GM}^{\mathrm{MT}}$ is that the anonymity game cannot be played on all clusters. More precisely, if the two challenge identities in $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ given in Appendix A.6 are for a newly joined member and an older member, then the game must be played on the clusters which contain signing leaves for both members. This is because if $\mathcal{A}$ is given a signature from clusters created before the new member has joined the group, then $\mathcal{A}$ can determine that this signature is signed by the older member. On the other hand, if the signature comes from clusters created with both members, the anonymity security is the same as that for static group construction given in Theorem 4.

6.4 Comparison and Performance Results

In this section, we compare $\mathsf{GM}^{\mathrm{MT}}$ with the hash-based group signature schemes GM [20] and DGM [21]. Due to the multi-tree construction, $\mathsf{GM}^{\mathrm{MT}}$ has a larger signature size than either GM or DGM, for instance, for 256-bit security, the signature size of $\mathsf{GM}^{\mathrm{MT}}$ is 10.816 KB whereas that of GM (resp. DGM) is 2.88 KB (resp. 2.72 KB). In Section 6.5, we provide an unoptimized C implementation for the purposes of performance evaluation.

6.4.1 $\mathsf{GM}^{\mathrm{MT}}$ and GM

Unlike GM, $\mathsf{GM}^{\mathrm{MT}}$ provides a revocation algorithm and is a multi-tree Merkle construction. Both schemes require comparable computations from the group manager for the opening and setup. Hence, we focus on the maximum number of signing leaves for these schemes and the storage requirements for each group member.

Maximum number of signing leaves. GM is a one-layer tree with a static group construction and the maximum number of signing leaves has been stated to be 2^{20} [20]. On the other hand, $\mathsf{GM}^{\mathrm{MT}}$allows the multi-tree structure to grow once the initial signing leaves are consumed by repeating the last two steps of the setup phase. Thus, the group members renew their signing keys each time a new cluster is generated while keeping the group public key unchanged. For a 4-layer $\mathsf{GM}^{\mathrm{MT}}$construction, up to 2^{64} signing leaves are created for the group depending on the tree height h.

Member storage requirements. In GM, the storage required for each group member is reported to be $B(1+\log N)$ nodes [20]. Note that since the first node of each authentication path and each leaf node contains an OTS public key and an AES-256 ciphertext, the required storage is in fact $B(3+\log N)$ n-bit elements. In $\mathsf{GM}^{\mathrm{MT}}$, for a cluster of N members, the

required storage is $B(2+\log N)+(d-1)l+h_{gm}$ n-bit elements. More precisely, a member stores the B nodes at the $(\log N)$-th level, each of which is n bits, and $B(1+\log N)$ n-bit elements for the authentication paths. Note that in GM, a group member stores 3 n-bit values per leaf node, while in $\mathsf{GM}^{\mathrm{MT}}$ a group member stores 2 n-bit values for each leaf node. Additionally, in $\mathsf{GM}^{\mathrm{MT}}$ each group member needs to store the signature of the group manager for the cluster tree root, which is composed of $d-1$ WOTS signatures, along with the corresponding authentication paths.

Table 6.3 gives the required storage for each group member in GM and $\mathsf{GM}^{\mathrm{MT}}$. We compare $\mathsf{GM}^{\mathrm{MT}}$ and GM when the total number of supported signatures is 2^{20} for the GM tree and $\mathsf{GM}^{\mathrm{MT}}$ cluster, which is the maximum number of signatures for GM, and with $N=2^{10}$ group members, so the number of signing leaves for each member is $B=2^{10}$. We choose $\mathsf{GM}^{\mathrm{MT}}$-256 instances for the comparison as it has the highest storage requirements among all instances. The results show that $\mathsf{GM}^{\mathrm{MT}}$-256c saves at least 5.8% of the required storage compared to GM-256. Note that the values in Table 6.3 are for 256-bit security where $l=67$. Thus, the above percentages will increase for lower bit security requirements, i.e., for 128 and 192 bit security with $l=35$ and 51, respectively. Given the recommended parameters in Table 6.2, the total required storage for each group member in $\mathsf{GM}^{\mathrm{MT}}$ is $B(2+\log N)+3l+48$ n-bit elements.

Table 6.3: Group member storage for GM and $\mathsf{GM}^{\mathrm{MT}}$ with $N=B=2^{10}$.

Algorithm	$B=N=2^{10}$	
	Required storage (number of nodes)	
GM	$B(3+\log N)$	$2^{10}\cdot 13=13312$
$\mathsf{GM}^{\mathrm{MT}}$-256c	$B(2+\log N)+3l^{\dagger}+48$	$2^{10}\times 12+3l+48=12537$

† The values are for $l=67$ and 256-bit security.

6.4.2 $\mathsf{GM}^{\mathrm{MT}}$ and DGM

Both DGM and $\mathsf{GM}^{\mathrm{MT}}$ are revocable GSSs, but DGM is a dynamic GSS that allows new members to be added to the group after the group public key is generated. Unlike $\mathsf{GM}^{\mathrm{MT}}$, DGM requires interaction between verifiers and the group manager to validate the authentication path for each signature verification. Moreover, the group manager in DGM generates the signing keys for the members and thus can sign on their behalf, so it does not satisfy exculpability [68]. A limitation of our scheme is that all group members simultaneously renew their signing keys periodically. Thus, a group member who has used all their signing leaves cannot renew them before a specific time as they need to wait until the new cluster generation occurs. On the other hand, DGM allows new leaves to be assigned on request. In what follows, we compare $\mathsf{GM}^{\mathrm{MT}}$with DGM with respect to the efficiency of the revocation mechanism.

Revocation efficiency. DGM utilizes symmetric puncturable encryption [66] in its revocation mechanism. With each new revoked member, the group manager punctures the encrypted indexes of the signing leaves of all revoked members. Hence, the group manager is required to store all the indexes assigned to all members. In $\mathsf{GM}^{\mathrm{MT}}$, the corresponding storage required is for the last assigned label of each member because all the encrypted labels assigned to a member can be regenerated from this label. For example, consider a $\mathsf{GM}^{\mathrm{MT}}$-256c instance which has 2^{15} members, supports 2^{64} signatures, and provides 256-bit security. The required storage in $\mathsf{GM}^{\mathrm{MT}}$ (resp. DGM) is $2^{15} \times 2^{8}$= 1 MB (resp. $2^{64} \times 2^{8} \approx 10^{8.7}$ TB). Both schemes have equal sized revocation lists and the revocation computational complexity of the group managers are comparable (linear in the size of the revocation list). However, for a revocation check in DGM, the verifier invokes a hash func-

tion for 3 times the number of revoked positions in the revocation list [66]. On the other hand, in $\mathsf{GM}^{\mathrm{MT}}$, the verifier must search for a hash output in a sorted revocation list which has logarithmic complexity. Hence, our revocation algorithm reduces the computational complexity for verification compared to DGM. Nevertheless, the revocation list is large, so in Appendix A.8 we provide an alternative revocation mechanism where the size of the revocation list is linear in the number of revoked members. The alternative mechanism is equivalent to traditional revocation by key, and may be suitable for some applications that do not require anonymity of the revoked members.

6.5 Implementation

In this section, we provide an unoptimized implementation of the main procedures of $\mathsf{GM}^{\mathrm{MT}}$ for the purposes of performance evaluation. This C language implementation uses the XMSS$^{\mathrm{MT}}$/WOTS standard implementation given in RFC 8391 [65], [14] employing SHA2-256 as a hash function, and AES256 for encryption. Shuffling the signing of leaf nodes is done by reordering the leaf nodes in ascending order using the Quicksort algorithm for 256 bit integers.

Table 6.4 provides the performance in kilocycles when the code is executed on an Intel(R) Core(TM) i5-5200U CPU at 2.20 GHz. The values in the table are the average of 100 runs. This table gives the performance for group public key generation, group member WOTS public keys generation, (cluster) label encryption, leaf shuffling, cluster root generation, cluster root signing, signature opening, message signing, and signature verification. The reported numbers are for the three instances $\mathsf{GM}^{\mathrm{MT}}$-256a with $(h_c, N, B) = (16, 2^6, 2^{10})$, $\mathsf{GM}^{\mathrm{MT}}$-256b with $(h_c, N, B) = (18, 2^8, 2^{10})$, and $\mathsf{GM}^{\mathrm{MT}}$-256c with $(h_c, N, B) = (20, 2^{10}, 2^{10})$. Other parameters are possible according to the application and member

Table 6.4: $\mathsf{GM}^{\mathrm{MT}}$ Performance results in kilocycles (kc).

Process	$\mathsf{GM}^{\mathrm{MT}}$-256a	$\mathsf{GM}^{\mathrm{MT}}$-256b	$\mathsf{GM}^{\mathrm{MT}}$-256c
	$(h_c, N, B) = (16, 2^6, 2^{10})$	$(h_c, N, B) = (18, 2^8, 2^{10})$	$(h_c, N, B) = (20, 2^{10}, 2^{10})$
Public key gen. (GM)	1,245,539,484		
WOTS public keys gen. (U)	6,147,667		
Label encryption (GM)	170,471	680,758	2,721,486
Shuffling (GM)	48,436	205,428	854,614
Cluster root gen. †(GM)	3,364,756	13,450,764	53,427,148
Cluster root signing (GM)	33,064		
Message signing (U)	2,957		
Signature verification (U)	12,174	15,124	19,326
Signature opening (GM)	46		

† The Merkle tree is constructed after the leaf nodes have been computed.

storage capabilities. A process is performed by a user (U) or the group manager (GM).

6.6 Conclusion

In this chapter we proposed $\mathsf{GM}^{\mathrm{MT}}$, a revocable hash-based group signature scheme that addresses some of the challenges identified by the designers of the GM and DGM hash-based group signature schemes. Unlike GM, $\mathsf{GM}^{\mathrm{MT}}$ is a multi-tree construction that allows up to 2^{64} signatures under one group public key. It was shown that $\mathsf{GM}^{\mathrm{MT}}$ saves at least 5.8% of the required storage for each group member compared to GM for an $\mathsf{GM}^{\mathrm{MT}}$-256c instance with 2^{10} group members each assigned 2^{10} signing leaves. Moreover, the required storage for the group manager in $\mathsf{GM}^{\mathrm{MT}}$ is linear in the number of members, while in DGM it is linear in the total number of signatures supported by the scheme. $\mathsf{GM}^{\mathrm{MT}}$ also reduces the computation complexity of checking revocations from linear in DGM to logarithmic in the size of the revocation list. An analysis of $\mathsf{GM}^{\mathrm{MT}}$ with respect to anonymity [63] and full traceability [62] was given which shows that its security relies on the standard security assumptions of hash functions and symmetric encryption, and the existential unforgeability of the underlying signing scheme. Finally, we compared $\mathsf{GM}^{\mathrm{MT}}$ to both GM and DGM, and presented the performance of its procedures using an unoptimized C implementation.

Chapter 7

Conclusion and Future Work

In this chapter, a conclusion of the ideas and contributions in this dissertation to improve the security and performance of (group) hash-based signature schemes is provided. In addition, ideas for the future work are provided.

7.1 Conclusion

In this work, the DFORS, vSPHINCS$^+$, DGM$^+$, and GM^{MT} hash-based signature schemes were introduced. DFORS is a few-time signing scheme. It provides security against adaptive chosen message attacks which is a factor $r + 1$ greater than FORS, the underlying hash-based few-time signing scheme of SPHINCS$^+$, where r is the number of messages that are signed by a key pair. The analysis showed that the claimed security of SPHINCS$^+$ is not affected and provides a better understanding of the security of its underlying signing scheme. A mechanism was given that it can be adopted by most HORS variants to provide security against adaptive chosen message attacks. vSPHINCS$^+$ is a SPHINCS$^+$ variant that utilizes a new ORS generation mechanism, v-ORS, which enables SPHINCS$^+$ to provide better performance at the same security level. For instance, some versions of vSPHINCS$^+$

offer around a 27% saving in the signing computational cost with minimal effect on the signature size. Other versions provide savings in the signing computational cost with shorter signature sizes. Given that high computational cost is the main reason for SPHINCS$^+$ being an alternate candidate in Round 3 of the NIST post-quantum cryptography competition, the results achieved for vSPHINCS$^+$ are a positive step towards practical adoption. A security analysis for DGM and GM was presented and DGM$^+$, a hash-based group signature scheme that is a DGM variant, was introduced based on these results. It mitigates multi-target attacks and provides 44 bits more security than DGM when $\mathsf{GM}^{\mathrm{MT}}$ and DGM sign 2^{64} messages. $\mathsf{GM}^{\mathrm{MT}}$is a revocable hash-based group signature scheme. It addresses some of the challenges identified in the current hash-based group signature schemes GM and DGM. Unlike GM, $\mathsf{GM}^{\mathrm{MT}}$ provides a revocation mechanism and a multi-tree construction that allows up to 2^{64} signatures under one group public key. $\mathsf{GM}^{\mathrm{MT}}$ lowers the required storage for each group member compared to GM. Moreover, the required storage for the group manager in $\mathsf{GM}^{\mathrm{MT}}$ is linear in the number of members, while in DGM it is linear in the total number of signatures supported by the scheme. $\mathsf{GM}^{\mathrm{MT}}$ also reduces the computational cost of checking revocations from linear in DGM to logarithmic in the size of the revocation list.

7.2 Future Work

Although stateless hash-based signature schemes provide the most secure schemes against (post-quantum) cryptanalysis attacks, more work should be done to improve their performance. vSPHINCS$^+$ outperforms other stateless hash-based signature schemes, but it could be improved if an OTS scheme that has smaller signature sizes and better performance is developed. Accordingly, when deployed in vSPHINCS$^+$, it would be comparable to the finalist algorithms in the current post-quantum cryptography standardization competition run by

NIST. Another area of research worth exploring is the use of hash functions in constructing privacy-enabling cryptographic primitives such as ring signatures, zero knowledge-proofs, and redactable signatures. Moreover a post-quantum key exchange algorithm is desirable to enable secure communications over inscure channels. Current post-quantum key-agreement NIST proposals to the are designed using a Key Encapsulation Mechanism (KEM), in which one communicating party generates a key pair (secret key and public key), and shares the public key, while the other party randomly generates a session secret key and sends this to the first party encrypted using their public key. Thus, exploring post-quantum key-agreement algorithm like the Diffie-Hellman algorithm will enable the communicating parties to agree on a shared secret key that can be used as a symmetric encryption key so both parties perform the same operations.

Appendix A

A.1 HORS Specification

The HORS key generation, signing, and verification procedures are given in Algorithm A.1.

A.2 Adaptive Chosen Message Attack Against HORS

In [6], the following adaptive chosen message attack against HORS was defined. Let $\mathcal{A}$ be an adaptive chosen message adversary against HORS such that given the key k, $\mathcal{A}$ can compute the hash of any message m and $ORS_\kappa(m)$ offline. Given a security parameter, n, under the birthday paradox, $\mathcal{A}$ can find $r+1$ messages in a cover relation C_κ^r with which to query the signing oracle, formally

$$\mathsf{Pr}[k \leftarrow K, (m_1, m_2, \ldots, m_{r+1}) \leftarrow \mathcal{A}(k) : C_\kappa^r(m_1, m_2, \ldots, m_{r+1})] \leq negl(n).$$

Aumasson and Endignoux [8] subsequently presented an adaptive chosen message attack against HORS and proved that the security level decreases by a factor of $r+1$ when compared to non adaptive chosen message attacks. Their attack is as follows. Given an adversary $\mathcal{A}$ and a key k, the hash value $H_k(m)$ for any message of their choice can be computed, and

Algorithm A.1 HORS Algorithm

procedure KEY GENERATION(t)
Generate the secret key SK at random, $SK = (sk_0, sk_1, \ldots, sk_{t-1})$
Compute the public key $PK = pk_0, pk_1, \ldots, pk_{t-1} = f(sk_0), f(sk_1), \ldots, f(sk_{t-1})$
Output (SK, PK)
end procedure

---

procedure SIGNING(m, κ, SK,k)
Compute $h = H_k(m)$, $h = h_0||h_1|| \ldots ||h_{\kappa-1}$.
$ORS_\kappa(m) = \{h_0, h_1, \ldots, h_{\kappa-1}\}$.
$\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{\kappa-1}) = (sk_{h_0}, sk_{h_1}, \ldots, sk_{h_{\kappa-1}})$
Output (σ)
end procedure

---

procedure VERIFICATION(m, κ, σ, PK,k)
Compute $h = H_k(m)$, $h = h_0||h_1|| \ldots ||h_{\kappa-1}$
$ORS_\kappa(m) = \{h_0, h_1, \ldots, h_{\kappa-1}\}$
for $0 \leq i \leq \kappa - 1$ **do**
if $f(\sigma_i) = pk_{h_i}$ **then**
$out = 1$
else
$out = 0$
break
end if
end for
Output (out)
end procedure

---

say there are $q > r$ messages. For all possible combinations of $(r + 1)$ messages from the q messages, $\mathcal{A}$ searches for $C_\kappa^{r-HORS}(m_1, m_2, \ldots m_{r+1})$ such that

$$C_\kappa^{r-HORS} \Leftrightarrow ORS(m_{r+1}) \in \bigcup_{j=1}^{r} ORS(m_j).$$

For any given subset, the probability of being an r-subset-cover relation is $(r\kappa/t)^\kappa$. The number of $(r + 1)$-message combinations which $\mathcal{A}$ can construct from the q messages are $\binom{q}{r+1}$ and each combination can form $\binom{r+1}{r}$ choices. Accordingly, their probability of success

in defeating the r-subset resilience (SR) is given by

$$Succ_{HORS}^{r-SR}(\mathcal{A}) \leq \binom{q}{r+1}\binom{r+1}{r}(\frac{r\kappa}{t})^{\kappa} \leq q\binom{q-1}{r}(\frac{r\kappa}{t})^{\kappa}.$$

Assuming a success probability close to 1, the security level of HORS against an adaptive chosen message attack is

$$\frac{\kappa}{r+1}(\log_2 t - \log_2 \kappa - \log_2 r) + \frac{\log_2 r!}{r+1}.$$

A.3 Hash Function Addressing Algorithm in SPHINCS$^+$

The hash function address (counter) consists of 256 bits (32 bytes). There are five address types for the five different instantiations of the tweakable hash function which are described below.

1. WOTS$^+$ hash address: The first field (4 bytes) is the layer address which indexes the layer in which the WOTS$^+$ exists. The tree address (12 bytes) indexes a tree within the layer and then the addressing type (4 bytes) which is equal to zero. The key pair address (4 bytes) denotes the index of the WOTS$^+$ within the hash tree. The chain address (4 bytes) denotes the number of the WOTS$^+$ secret key on which the chain is applied, and the hash address (4 bytes) denotes the number of one way function iterations within a chain.

2. WOTS$^+$ public key: The first field (4 bytes) is the layer address (the layer in which the WOTS$^+$ is exists). The tree address (12 bytes) identifies the tree within the layer and then the addressing type (4 bytes) which is equal to one. The key pair address (4 bytes) denotes the index of the WOTS$^+$ within the hash tree and finally there is

padding of 12 bytes of zeros.

3. main tree hash: The first field (4 bytes) is the layer address (the layer in which the WOTS$^+$ is exists). The tree address (12 bytes) denotes the tree within the layer and then the addressing type (4 bytes) which is equal to two. There is padding of 4 bytes of zeros, followed by the tree height (4 bytes) which denotes the height (level) within the tree in which the hash is applied, and the tree index (4 bytes) which denotes the node within that tree level.

4. FORS trees hashes: The first field (4 bytes) is the layer address which denotes the layer in which the FORS exists. The tree address (12 bytes) indexes a tree within the layer and then the addressing type (4 bytes) which is equal to three. The key pair address (4 bytes) indexes the FORS instance used and is equal to the value of the WOTS$^+$ used to authenticate it. The tree height (4 bytes) denotes the level within the FORS tree in which the hash is applied, and then the tree index (4 bytes) which indexes the FORS tree in which the hash is applied.

5. FORS public key: The first field (4 bytes) is the layer address. The tree address (12 bytes) indexes the tree within the layer and then the addressing type (4 bytes) which is equal to four. The key pair address (4 bytes) indexes the FORS instance used and finally there is padding of 8 bytes of zeros.

A.4 Sage Script for Evaluating The ITSR Bit Security for SPHINCS$^+$ and vSPHINCS$^+$

In what follows, we report the sage script python code that is used to evaluate the bit security for both SPHINCS$^+$ and vSPHINCS$^+$ using the new parameters. The code is adapted from that reported in NIST's PQC SPHINCS$^+$ submission [9].

```
import math
#Algrothm parameters: Set the following variables
k=12          #number of FORS trees for The FORS instance
t=14          # a FORS tree height
h=63          #The hyper tree height
leaves=2^h    #The number of FORS instances
qs=2^64       # maximum number of signatures

vSPHINCSprobITSR=0.0
SPHINCSprobITSR=0.0
vSPHINCStemp=0.0
SPHINCStemp=0.0
# prob of hitting a specific FORS instance r-times out of qs queries
F = RealIntervalField(228)
def qhitprob(leaves,qs,r):
    p = 1/F(leaves)
    return binomial(qs,r)*p^r*(1-p)^(qs-r)
# Pr[vORS (vSPHINCS+) forgery given that exactly r sigs hit the leaf]
def forgeryprobvSPHINCS(t,r,k):
    return (1-(1-(k+1)/(k*F(2^t)))^r)^(k+1)
# Pr[ORS (SPHINCS+) forgery given that exactly r sigs hit the leaf]
def forgeryprobSPHINCS(t,r,k):
    return (1-(1-1/F(2^t))^r)^(k)

for r in range(100):
  vSPHINCStemp=qhitprob(leaves,qs,r)*forgeryprobvSPHINCS(t,r,k)
  SPHINCStemp=qhitprob(leaves,qs,r)*forgeryprobSPHINCS(t,r,k)
  vSPHINCSprobITSR= vSPHINCSprobITSR+ vSPHINCStemp
  SPHINCSprobITSR=SPHINCSprobITSR+ SPHINCStemp

else:
  print "k=", k
  print "t=", t
  print "h=", h
  print "vSPHINCS+ ITSR bit Security =", -log(vSPHINCSprobITSR,2.0)
  print "SPHINCS+ ITSR bit Security =", -log(SPHINCSprobITSR,2.0)
```

A.5 XMSS-T Addressing Scheme

XMSS-T utilizes a hash function addressing scheme that enumerates each hash call in the scheme and outputs a distinct hash randomizer r and bit mask, q, for each hash call to mitigate multi-target attacks [65]. XMSS-T has three main substructures, WOTS-T, L-tree, and Merkle tree hash. The first substructure requires for each hash call a hash randomizer and bit mask, each of n bits. The others two substructures require a hash randomizer of n

bits and $2n$ bits for the bit mask. The hash function address consists of 256 bits. There are three address types for the three substructure mentioned above which are described below.

1. WOTS-T hash address: The first field (32 bits) is the tree layer address which indexes a given layer in which the WOTS-T exists (this value is set to zero for DGM). The tree address (64 bits) indexes a tree within the layer (this value is set to zero for DGM) and then the addressing type (32 bits) which is equal to zero. The key pair address (32 bits) denotes the index of the WOTS-T within the hash tree. The chain address (32 bits) denotes the number of the WOTS-T secret key on which the chain is applied. The hash address (32 bits) denotes the number of the hash function iterations within a chain. The last field is the KeyAndMask (32 bits) that is used to generate two different addresses for one hash function call (it is set to zero to get the hash randomizer R and it is set to one to get the bit mask each of n bits).

2. L-tree hash address: The first field (32 bits) is the layer address which indexes the layer in which the WOTS-T exists (this value is set to zero for DGM). The tree address (64 bits) indexes a tree within the layer (this value is set to zero for DGM) and then the addressing type (32 bits) which is equal to one. The L-tree address (32 bits) denotes the leaf index that is used to sign the message. The tree height (32 bits) encodes the node height in the L-tree. The tree index (32 bits) refers to the node index within that height. The last field is the KeyAndMask (32 bits) which in this substructure is used to generate three different addresses for one hash function call (it is set to zero to get the hash randomizer, R, it is set to one to get the first bit mask and it is set to two to get the second bit mask each of n bits).

3. Merkle tree hash: The first field (32 bits) is the layer address which indexes the layer

in which the WOTS-T exists (this value is set to zero for DGM). The tree address (64 bits) indexes a tree within the layer (this value is set to zero for DGM) and then the addressing type (32 bits) which is equal to two. Then a padding of zeros (32 bits). The tree height (32 bits) encodes the node height in the main Merkle tree. Then the tree index (32 bits) refers to the node index within that height. As the L-tree addressing, the last field is the KeyAndMask (32 bits), it is used to generate three different addresses for one hash function call (it is set to zero to get the hash randomizer, R, it is set to one to get the first bit mask and it is set to two to get the second bit mask).

A.6 GSS Security Notion Experiments

In the following security experiments, we assume an adversary is allowed a training phase where they can call the following oracles.

- $Corrupt(id_i)$: The adversary $\mathcal{A}$ has access to all secret keys belonging to member id_i.
- $chal_b(id_0, id_1, M)$: The oracle returns the signature of message M for a randomly chosen group member id_b for $b \in \{0, 1\}$.
- $Sign(M, id_i)$: The oracle returns the signature of a message M for a randomly chosen group member id_i where $1 \leq i \leq N$.
- $Open(\Sigma, GPK, M)$: The oracle returns the identity id_i of the member who issued the valid signature Σ of message M.

Anonymity. In the security experiment Exp^{Anon-b} for anonymity, an adversary $\mathcal{A}$ is given unrestricted access to both the signing and opening oracles and they have the ability to corrupt some of the group members. Then in the challenge phase, they are given a signature

of a message of their choice that is signed by one of two uncorrupted users i_0 or i_1. $\mathcal{A}$ wins Exp^{Anon-b} if they are able to identify the signer's identity with a non-negligible advantage. The security experiment $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ for CCA2-full-anonymity is given below.

$Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}(n, N)$
- $(GPK, sk^*) \leftarrow G.KGen(1^n, 1^N)$
- Unrestricted queries:
  * Corrupt(id_i)
  * $Sign(M, id_i)$:
  * $Open(\Sigma, PK, M)$
- $\sigma \leftarrow chal_b(id_0, id_1, M)$
- Return b

Figure A.1: Anonymity experiment

Full traceability. This security notion requires that the group manager is always able to reveal the identity of a signer of a valid signature and trace back every signature to the corresponding signer. Moreover, full traceability ensures that even if an adversary is capable of corrupting some group members, they are not able to generate a valid signature which is traced by the group manager to an uncorrupted member. The security experiment $Exp^{Full-Trace}_{\mathcal{GS},\mathcal{A}}$ for full traceability is given below.

$Exp^{Full-Trace}_{\mathcal{GS},\mathcal{A}}(n, N)$
- $(GPK, sk^*) \leftarrow G.KGen(1^n, 1^N)$
- Unrestricted queries:
  * Corrupt(id_i)
  * $Sign(M, id_i)$
  * $G.Open(M, \Sigma)$
- Generate Σ
- Return $G.Verify(\Sigma, M, gpk) == 1 \wedge G.Open(\Sigma) = \perp$ or id_j(non corrupted id_j)

Figure A.2: Full traceability experiment

A.7 Winternitz One-Time Signature Scheme Tightened (WOTS-T)

WOTS is a one-time signing scheme which has had many variants proposed to improve performance. This section gives a description of the latest version WOTS-T [14] that has been proposed to mitigate multi-target attacks. WOTS is parameterized by the security parameter $n \in \mathbb{N}$ which is also the input message length as well as the length of each secret and public key. The Winternitz parameter $w \in \mathbb{N}$, $w > 1$, identifies a tradeoff between the signature size and its generation\verification time. n and w are used to compute the required number of secret keys l as follows

$$l_1 = \lceil \frac{n}{\log(w)} \rceil, \qquad l_2 = \lfloor \frac{\log(l_1(w-1))}{\log(w)} \rfloor + 1, \qquad l = l_1 + l_2$$

The parameter w is a power 2 and the recommended values are $\{4, 16, 256\}$. The value of w determines the signature size and the algorithm speed. There is a trade off, as w increases the signature size decreases and the number of calls to the hash function increases, and vice versa. This value does not affect on the security.

Key generation algorithm. The algorithm takes as input the secret seed $SK.seed$, the public seed $PK.seed$, and the address $ADRS$ of the WOTS (the position of WOTS in the Merkle tree). The algorithm uses a pseudo random function PRF to generate the l WOTS secret keys $(sk_0, sk_1, \ldots, sk_{l-1} \leftarrow PRF(SK.seed, ADRS_i)$. The chain of the one-way function F with $w - 1$ iterations for the secret keys is defined as follows

$$C^j(sk_i) = F(PK.seed, ADRS_{i,j}, F^{j-1}(sk_i) \oplus r_{i,j}),$$

where $C^j(sk_i)$ is the j-th iteration for the i-th secret key sk_i, $0 \leq i \leq l-1$, $0 \leq j \leq w-1$,

$ADRS_{i,j}$ is the address for the j-th call for the i-th secret key (sk_i), $r_{i,j}$ is the bitmask generated using the public seed and ADRS, and $C^0(sk_i) = sk_i$. The last values of the l chains are the WOTS public key

$$PK = pk_0, pk_1, \ldots, pk_{l-1} = C^{w-1}(sk_0), C^{w-1}(sk_1), \ldots, C^{w-1}(sk_{l-1}).$$

Signing algorithm. The signing algorithm takes as input the l secret keys each of length n bits, the Winternitz parameter w, and the message to be signed of length n bits. These message bits are divided into $l_1 = \lceil \frac{n}{\log(w)} \rceil$ substrings $(a_0, a_1, \ldots, a_{l_1-1})$ each of $\log(w)$ bits. Then the checksum S is calculated as

$$S = \sum_{i=0}^{l_1-1} (w - 1 - a_i).$$

This checksum is divided into $l_2 = \lfloor \frac{\log(l_1(w-1))}{\log(w)} \rfloor + 1$ substrings $(S_0, S_1, \ldots, S_{l_2-1})$ each of length $\log(w)$ bits. Let $B = b_0, b_1, \ldots, b_{l-1} = a_0, a_1, \ldots, a_{l_1-1}, S_0, S_1, \ldots, S_{l_2-1}$. The signature σ is computed by mapping the secret keys into intermediate values in each chain as follows

$$\sigma = \sigma_0, \sigma_1, \ldots, \sigma_{l-1} = C^{b_0}(sk_0), C^{b_1}(sk_1), \ldots, C^{b_{l-1}}(sk_{l-1}).$$

The reason for using a checksum is to guarantee that for the message $B = b_0, b_1, \ldots, b_{l-1} = a_0, a_1, \ldots, a_{l_1}, S_0, S_1, \ldots, S_{l_2-1}$, it is impossible to find another message $B' = b'_0, b'_1, \ldots, b'_{l-1} = a'_0, a'_1, \ldots, a'_{l_1-1}, S'_0, S'_1, \ldots, S'_{l_2-1}$ such that $\forall i, 0 \leq i < l, b'_i \geq b_i$.

Verification algorithm. The verification algorithm takes as input the message M, the signature σ, the address ADRS, the public seed PK.seed, and the WOTS public key. It computes the integers $B = b_0, b_1, \ldots, b_{l-1}$ as described in the signing algorithm and then applies $w - 1 - b_i$ iterations on the received signature to get the public key, which is given

as follows

$$PK' = pk'_0, pk'_1, \ldots, pk'_{l-1} = C^{w-1-b_0}(sk_0), C^{w-1-b_1}(sk_1), \ldots, C^{w-1-b_{l-1}}(sk_{l-1}).$$

A formal verification algorithm compares the calculated public key PK' to the given public key PK and outputs 1 if they are equal (valid signature), and 0 otherwise. However, in Merkle tree constructions, this process is delegated to the Merkle scheme verification algorithm.

A.8 An Alternative Solution for a Large Revocation List

In this section we introduce a solution for the revocation list of $\mathsf{GM}^{\mathrm{MT}}$which is suitable for some applications that do not require anonymity of revoked members. The proposed solution is equivalent to the traditional revocation by a secret key mechanism. We propose the following modification to the leaf generation procedure.

- The group manager generates a secret key sk_i^* for each group member. This key is different from the group member secret key sk_i that is used to generate the WOTS signing keys.
- The encrypted label in $\mathsf{GM}^{\mathrm{MT}}$ is replaced by the output of hashing the concatenation of the corresponding $WOTS.pk$ and the group member key $A^* = H(WOTS.pk||sk_i^*)$.

The remaining procedures are the same as in $\mathsf{GM}^{\mathrm{MT}}$with the following three differences in the revocation, verification and opening procedures.

- To revoke the j-th member, the group manager adds their key sk_j^* to the revocation list, $RevList$.
- In the verification process, the verifier checks if the calculated WOTS from the signature and keys in the revocation list gives the value A^* in the received signature

(which means that the signature has been revoked), if not the verifier completes the verification procedures.

- In the opening process, the group manager checks which group member's secret key sk_i^* gives the value A^* in the signature $A^* = H(WOTS.pk||sk_i^*)$ for $0 \leq i \leq N-1$.

Applying the above modification has the following consequences.

- The revocation list size is linear in the number of revoked members, while in $\mathsf{GM}^{\mathrm{MT}}$ it is linear in the number of revoked leaves.
- Revocation does not maintain the anonymity of revoked members.
- The verification complexity is linear in the number of revoked members, while $\mathsf{GM}^{\mathrm{MT}}$ verification has logarithmic computational complexity with respect to the number of revoked leaves.
- The opening complexity is linear in the number of members, while $\mathsf{GM}^{\mathrm{MT}}$ has a constant opening complexity, i.e. one decryption operation.

References

[1] F. Song, "A note on quantum security for post-quantum cryptography," in *International Workshop on Post-Quantum Cryptography, (PQC'2014)*. Springer, 2014, pp. 246–265.

[2] L. Lamport, "Constructing digital signatures from a one-way function," Technical Report CSL-98, SRI International Palo Alto, Tech. Rep., 1979.

[3] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing, and M. Rückert, "On the security of the Winternitz one-time signature scheme," in *International Conference on Cryptology in Africa (AFRICACRYPT'2013)*. Springer, 2011, pp. 363–378.

[4] A. Hülsing, "W-OTS$^+$–Shorter signatures for hash-based signature schemes," in *International Conference on Cryptology in Africa, (AFRICACRYPT'2013)*. Springer, 2013, pp. 173–188.

[5] A. Perrig, "The BiBa one-time signature and broadcast authentication protocol," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2001, pp. 28–37.

[6] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," in *Australasian Conference on Information Security and Privacy, (ACISP'2002)*. Springer, 2002, pp. 144–153.

[7] J. Pieprzyk, H. Wang, and C. Xing, "Multiple-time signature schemes against adaptive chosen message attacks," in *International Workshop on Selected Areas in Cryptography, (SAC'2003)*. Springer, 2003, pp. 88–100.

[8] J.-P. Aumasson and G. Endignoux, "Clarifying the subset-resilience problem," *IACR Cryptology ePrint Archive*, 2017.

[9] D. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. Lauridsen *et al.*, "SPHINCS$^+$–submission to the NIST post-quantum project," 2017.

[10] R. C. Merkle, "A certified digital signature," in *Conference on the Theory and Application of Cryptology, (CRYPT0'89)*. Springer, 1989, pp. 218–238.

[11] J. Buchmann, E. Dahmen, and A. Hülsing, "XMSS - A Practical forward secure signature scheme based on minimal security assumptions," in *International Workshop on Post-Quantum Cryptography, (PQC'2011)*. Springer, 2011, pp. 117–129.

[12] A. Hülsing, C. Busold, and J. Buchmann, "Forward secure signatures on smart cards," in *International Conference on Selected Areas in Cryptography, (SAC'2012)*. Springer, 2012, pp. 66–80.

[13] A. Hülsing, L. Rausch, and J. Buchmann, "Optimal parameters for XMSS^{MT}," in *International Conference on Availability, Reliability, and Security*. Springer, 2013, pp. 194–208.

[14] A. Hülsing, J. Rijneveld, and F. Song, "Mitigating multi-target attacks in hash-based signatures," in *Public-Key Cryptography, (PKC'2016)*. Springer, 2016, pp. 387–416.

[15] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, "SPHINCS:

Practical stateless hash-based signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT'2015)*. Springer, 2015, pp. 368–397.

[16] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS$^+$ signature framework," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2129–2146.

[17] J.-P. Aumasson and G. Endignoux, "Improving stateless hash-based signatures," in *Cryptographers' Track at the RSA Conference*. Springer, 2018, pp. 219–242.

[18] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner *et al.*, "NISTIR 8309 status report on the second round of the NIST post-quantum cryptography standardization process," *National Institute of Standards and Technology, US Department of Commerce*, 2020.

[19] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems, (CHES'2018)*, pp. 238–268, 2018.

[20] R. El Bansarkhani and R. Misoczki, "G-Merkle: A hash-based group signature scheme from standard assumptions," in *International Conference on Post-Quantum Cryptography, (PQC'2018)*. Springer, 2018, pp. 441–463.

[21] M. Buser, J. K. Liu, R. Steinfeld, A. Sakzad, and S.-F. Sun, "DGM: A dynamic and revocable group Merkle signature," in *European Symposium on Research in Computer Security, ESORICS'2019*. Springer, 2019, pp. 194–214.

[22] M. Yehia, R. AlTawy, and T. A. Gulliver, "Hash-based signatures revisited: A Dynamic FORS with adaptive chosen message security," in *International Conference on Cryptology in Africa, (AFRICACRYPT'2020)*. Springer, 2020, pp. 239–257.

[23] ——, "Verifiable obtained random subsets for improving SPHINCS$^+$," in *Australasian Conference on Information Security and Privacy, (ACISP'2021)*. Springer, 2021.

[24] ——, "Security analysis of DGM and GM group signature schemes instantiated with XMSS-T," in *International Conference on Information Security and Cryptology, (Insecrypt'2021)*. Springer, 2021.

[25] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry, "Random oracles in a quantum world," in *International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT'2011)*. Springer, 2011, pp. 41–69.

[26] X. Bonnetain, A. Hosoyamada, M. Naya-Plasencia, Y. Sasaki, and A. Schrottenloher, "Quantum attacks without superposition queries: The offline Simon's algorithm," in *International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT'2019)*. Springer, 2019, pp. 552–583.

[27] J. W. Bos, A. Hülsing, J. Renes, and C. van Vredendaal, "Rapidly verifiable XMSS signatures," *IACR Transactions on Cryptographic Hardware and Embedded Systems, (CHES'2021)*, pp. 137–168, 2021.

[28] R. C. Merkle, "A certified digital signature," in *Conference on the Theory and Application of Cryptology, (CRYPT0'89)*. Springer, 1989, pp. 218–238.

[29] F. Shahid, I. Ahmad, M. Imran, and M. Shoaib, "Novel one-time signatures (NOTS): A compact post-quantum digital signature scheme," *IEEE Access*, vol. 8, pp. 15 895–15 906, 2020.

[30] P. Erdös, P. Frankl, and Z. Füredi, "Families of finite sets in which no set is covered by the union of r others," *Israel Journal of Mathematics*, vol. 51, no. 1, pp. 79–89, 1985.

[31] D. Chaum and E. Van Heyst, "Group signatures," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 257–265.

[32] D. Boneh and H. Shacham, "Group signatures with verifier-local revocation," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2004, pp. 168–177.

[33] J. Traoré, "Group signatures and their relevance to privacy-protecting offline electronic cash systems," in *Australasian Conference on Information Security and Privacy, (ACISP'99)*. Springer, 1999, pp. 228–243.

[34] R. AlTawy and G. Gong, "Mesh: A supply chain solution with locally private blockchain transactions," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 149–169, 2019.

[35] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *International Cryptology Conference, (CRYPTO'2002)*. Springer, 2002, pp. 61–76.

[36] ——, "Signature schemes and anonymous credentials from bilinear maps," in *International Cryptology Conference, (CRYPTO'2004)*. Springer, 2004, pp. 56–72.

[37] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *International Cryptology Conference, (CRYPTO'2004)*. Springer, 2004, pp. 41–55.

[38] B. Libert, T. Peters, and M. Yung, "Group signatures with almost-for-free revocation," in *International Cryptology Conference, (CRYPTO'2012)*. Springer, 2012, pp. 571–589.

[39] ——, "Scalable group signatures with revocation," in *International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT'2012)*. Springer, 2012, pp. 609–627.

[40] NIST, "Post-quantum cryptography project." http://csrc.nist.gov/groups/ST/post-quantum-crypto, 2016.

[41] S. D. Gordon, J. Katz, and V. Vaikuntanathan, "A group signature scheme from lattice assumptions," in *International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT'2010)*. Springer, 2010, pp. 395–412.

[42] F. Laguillaumie, A. Langlois, B. Libert, and D. Stehlé, "Lattice-based group signatures with logarithmic signature size," in *International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT'2013)*. Springer, 2013, pp. 41–61.

[43] A. Langlois, S. Ling, K. Nguyen, and H. Wang, "Lattice-based group signature scheme with verifier-local revocation," in *International Workshop on Public Key Cryptography, (PKC'2014)*. Springer, 2014, pp. 345–361.

[44] S. Ling, K. Nguyen, and H. Wang, "Group signatures from lattices: Simpler, tighter, shorter, ring-based," in *IACR International Workshop on Public Key Cryptography, (PKC'2015)*. Springer, 2015, pp. 427–449.

[45] P. Q. Nguyen, J. Zhang, and Z. Zhang, "Simpler efficient group signatures from lattices," in *IACR International Workshop on Public Key Cryptography, (PKC'2015)*. Springer, 2015, pp. 401–426.

[46] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang, "Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions," in *International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT'2016)*. Springer, 2016, pp. 373–403.

[47] R. Del Pino, V. Lyubashevsky, and G. Seiler, "Lattice-based group signatures and zero-knowledge proofs of automorphism stability," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 574–591.

[48] R. Yang, M. H. Au, Z. Zhang, Q. Xu, Z. Yu, and W. Whyte, "Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications," in *International Cryptology Conference, (CRYPTO'2019)*. Springer, 2019, pp. 147–175.

[49] Q. Alamélou, O. Blazy, S. Cauchie, and P. Gaborit, "A practical group signature scheme based on rank metric," in *International Workshop on the Arithmetic of Finite Fields*. Springer, 2016, pp. 258–275.

[50] ——, "A code-based group signature scheme," *Designs, Codes and Cryptography*, vol. 82, no. 1-2, pp. 469–493, 2017.

[51] M. F. Ezerman, H. T. Lee, S. Ling, K. Nguyen, and H. Wang, "Provably secure group signature schemes from code-based assumptions," *IEEE Transactions on Information Theory*, vol. 66, no. 9, pp. 5754–5773, 2020.

[52] B. E. Ayebie, H. Assidi, and E. M. Souidi, "A new dynamic code-based group signature scheme," in *International Conference on Codes, Cryptology, and Information Security*. Springer, 2017, pp. 346–364.

[53] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," *Journal of the ACM*, vol. 38, no. 3, pp. 690–728, 1991.

[54] I. Giacomelli, J. Madsen, and C. Orlandi, "Zkboo: Faster zero-knowledge for boolean circuits," in *USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1069–1083.

[55] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge proofs from secure multiparty computation," *SIAM Journal on Computing*, vol. 39, no. 3, pp. 1121–1152, 2009.

[56] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1825–1842.

[57] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2087–2104.

[58] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 525–537.

[59] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the Theory and Application of Cryptographic Techniques, (CRYPTO'86)*. Springer, 1986, pp. 186–194.

[60] D. Unruh, "Non-interactive zero-knowledge proofs in the quantum random oracle model," in *International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT'2015)*. Springer, 2015, pp. 755–784.

[61] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *International Cryptology Conference, (CRYPTO'92)*. Springer, 1992, pp. 390–420.

[62] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assump-

tions," in *International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT'2003)*. Springer, 2003, pp. 614–629.

[63] J. Camenisch and J. Groth, "Group signatures: Better efficiency and new theoretical aspects," in *International Conference on Security in Communication Networks*. Springer, 2004, pp. 120–133.

[64] M. Shafieinejad and N. N. Esfahani, "A scalable post-quantum hash-based group signature," *Designs, Codes and Cryptography*, vol. 89, no. 5, pp. 1061–1090, 2021.

[65] A. Hülsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme," in *RFC 8391*. IRTF, 2018.

[66] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 763–780.

[67] NIST, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process." https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals, 2016.

[68] G. Ateniese and G. Tsudik, "Some open issues and new directions in group signatures," in *International Conference on Financial Cryptography*. Springer, 1999, pp. 196–211.