Monash University

Post-quantum Privacy-Preserving Primitives Constructed with Symmetric Primitives

Author
Maxime Buser

Supervisors
Assoc. Prof. Joseph K. Liu
Assoc. Prof. Ron Steinfeld,
Dr. Amin Sakzad

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy at

Monash University

in the

Faculty of Information Technology

April 24, 2022

Copyright notice

MONASH UNIVERSITY

Abstract

Faculty of Information Technology

Doctor of Philosophy

Post-quantum Privacy-Preserving Primitives Constructed with Symmetric Primitives

by Maxime Buser

Digital privacy has become one of society's main concerns in recent years. Cryptography proposes different primitives to provide privacy, including group and ring signatures, which are considered the most promising. Group signatures allow the member of a group supervised by a central authority to sign a message anonymously on behalf of the group while ring signatures provide the same feature without requiring a central authority. The research community highlights the necessity to design cryptographic primitives that can resist quantum computers. A current popular research direction consists of using symmetric primitives-based protocols to provide quantum-safe schemes because of their well-understood security and algorithm efficiency. Therefore, this thesis focuses on the challenge of constructing practical post-quantum privacy-preserving protocols using only symmetric primitives. More specifically, this thesis aims to design new post-quantum group signatures and ring signatures based exclusively on symmetric primitives because they allow the member of a group to authenticate a message anonymously.

The quest for this target led to four major contributions: the design of three different fully dynamic group signatures and the development of a new generic construction for an ID-based ring signature. Our fully dynamic group signatures DGM, (T-)SGS and xGS all improve the current state-of-the-art in different ways. DGM extends the static G-Merkle group signature, reducing the influence of the group size on its competitive signature size. It also uses the innovative symmetric puncturable encryption technique to revoke the signing ability to group members. (T-)SGS innovatively extends the stateless digital signature SPHINCS+ to generate a stateless group signature with a non-interactive verification procedure. xGS is the first non-interactive fully dynamic group signature that can handle large groups. It is also the first post-quantum construction providing the feature of *user-controlled linkability*. Finally, the generic construction of ID-based ring signature is the first based on symmetric primitives from which two applicable constructions were derived: PicRS and XRS. Both improve the current state-of-the-art thanks to their nearly constant and competitive ring signature size. This thesis demonstrates that symmetric primitives are a valid candidate to preserve the privacy of users as we head towards the democratization of quantum computers.

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: **M. Buser**

Print Name: **Maxime Buser**

Date: **13.10.2021**

List of publications included in the thesis

Published paper

- Buser, Maxime, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, and Shi-Feng Sun."DGM: A Dynamic & Revocable Group Merkle Signature" published at the conference European Symposium on Research in Computer Security (ESORICS) 2019 [ESOa]

- Buser, Maxime, Joseph K. Liu, Ron Steinfeld, and Amin Sakzad."Post-Quantum ID-based Ring Signatures from Symmetric-key primitives" submitted at the conference Applied Cryptography and Network Security (ACNS) 2022 [ACN]

Publications in communication

- Buser, Maxime, Joseph K. Liu, Ron Steinfeld, and Amin Sakzad."Efficient Post-Quantum Group Signature from Symmetric Primitives with User-Controlled Linkability" submitted at the conference ACM ASIACSS 2022 [ACMa]

List of other publications related to the thesis

Publications in proceedings

- Buser, Maxime, Rafael Dowsley, Muhammed F. Esgin, Shabnam Kasra Kermanshahi, Veronika Kuchta, Joseph K. Liu, Raphael Phan, and Zhenfei Zhang. “Post-Quantum Verifiable Random Function from Symmetric Primitives in PoS Blockchain." submitted at the conference ESORICS 2022 [ESOb]

Publications in communication

- Buser, Maxime, Rafael Dowsley, Muhammed F. Esgin, Shabnam Kasra Kermanshahi, Veronika Kuchta, Jason T. LeGrow, Joseph K. Liu, Raphael Phan, Amin Sakzad, Ron Steinfeld, Jiangshan Yu. “SoK: Exotic Signatures for Post-Quantum Blockchain: Challenges & Research Directions" submitted at the journal ACM Computing Surveys (ACM CSUR 2022) [ACMb]

Acknowledgements

First, I would like to express my gratitude to my supervisors, Associate Professor Joseph K. Liu, Dr. Amin Sakzad and Associate Professor Ron Steinfeld. I have really appreciated to work with you on such an interesting topic. Your guidance, motivation and passion were of inestimable help to achieve the goal of this research project. More particularly, I would like to thank my main supervisor Joseph K. Liu, who believed in me from the start of this project and provided me with a full scholarship and supported me to attend conferences and workshops on different occasions.

I wish to thank Julie Holden for her discussions and suggestions of an inestimable value regarding the writing and proofreading of this thesis.

I am also grateful to have enjoyed the support of my colleagues and now friends at Monash University, including Dimaz Ankaa Wijaya, Muhammed Fethullah Esgin, Thusitha Thilina Dayaratne, Ahmad Salehi Shahrak and Viet Vo. They all made my life in Melbourne as likable as possible.

I am also taking advantage of these lines to thank my family and my partner, Lea Charriere for their unbounded support during this journey. I would not be able to complete this thesis without them by my side.

Contents

List of Figures

List of Tables

List of Symbols

General

RS	**R**ing **S**ignature
GS	**G**roup **S**ignature
PPT	Probabilistic polynomial time
NIZK	Non-interactive zero-knowledge proof system
m	Message
DS	Digital signature
OTS	One-time signature scheme
λ	Post-quantum security parameter
N	Group/ring Size
$\mathcal{M}$	Group manager
H	Cryptographic hash function
gpk	Group public key
gsk	Group secret key
PRF	Pseudorandom function

Chapter 3

DGM	Dynamic group Merkle signature
PPRF	Puncturable pseudorandom funtion
PRNG	Pseudorandom number generator
SPE	Symmetric puncturable encryption
h_{IMT}	Height of the Initial Merkle Tree
$rvks$	Number of revoked elements
GM	G-Merkle
DGM.rvk	Public revocation key

Chapter 4

C	Circuit
SGS	SPHINCS Group Signature
T-SGS	Traceable SPHINCS Group Signature
MS	Merkle Signature
FORS	Few-time signature scheme FORS
d	Number of layers in the hyper-tree

Chapter 5

x	Number of unlinkable signatures
xGS	Group signature with x unlinkable signatures
XMSS	XMSS digital signature

XMSS^{MT}	XMSS$^{\mathrm{MT}}$ digital signature
h	Height of the modified XMSS tree
h_{MT}	Height of the XMSS$^{\mathrm{MT}}$ hyper-tree
d	Number of layers in the hyper-tree

Chapter 6

IDRS	ID-based Ring Signature
C	Circuit
PKG	Private key generator
mpk	PKG's public key
msk	PKG's secret key
PicRS	ID-based ring signature constructed from Picnic signature
XRS	ID-based ring signature constructed from XMSS signature
L	Ring of identities

Chapter 1

Introduction

Preserving privacy has long been a concern for individuals, and it still is today. Indeed, we all want to keep our privacy, not only in the real world, but also in the digital world. One example of privacy in the digital world is e-voting where, similar to a voting poll situation, we need to assure the properties of anonymity, privacy and security On the one hand, the voters would like their votes to be taken into account, but they also want their votes to stay anonymous. On the other hand, the organizer of the poll wants to be sure that only votes of allowed individuals are taken into account. Therefore, the origin of data, in this case the owner of the vote, needs to be ensured through authentication procedures. Authentication and privacy can be provided by cryptography techniques., where cryptography is the study of secure communication techniques in the presence of adversarial behavior.

Currently, our digital world is mode of computers, which have specific computation power and abilities, we refer to them as classical computers. An alternative for these in the future are **quantum** computers, which are able to use the properties of quantum physics to store data and perform computations. Quantum computers generate more powerful adversaries and therefore, require a type of cryptography that can resist quantum attacks, which is referred to as **post-quantum** cryptography.

This thesis is dedicated to the design and study of cryptographic primitives that provide authentication and privacy and, at the same time, resist quantum attacks.

1.1 Research background

This section outlines the key terms, the current research and definitions in which our work is situated.

1.1.1 Authentication

Similar to the real world, authentication of documents, messages or content is essential in the digital world. The premise of digital authentication are digital signature schemes (see Definition 2.4.1), whose goal is to authenticate documents. Digital signature schemes allow the signing of a digital message m, which can be a document or a simple bit-string These useful cryptographic primitives were introduced and defined for the first time by Whitfield Diffie and Martin Hellman in [DH76]. A digital signature allows the authentication of a message, which can be a document or transaction for example, through a key pair. The key pair is composed of a private key used to generate the signature and a public key used to verify the validity of a digital signature. Three algorithms usually define a digital signature:

- DS.KeyGen: This generates a key pair composed of the public key DS.pk and the private key DS.sk.

- DS.Sign: This generates a digital signature $\mathsf{DS}.\sigma$ for a message m through the secret key DS.sk
- DS.Verify: This verifies the digital signature $\mathsf{DS}.\sigma$ for the message m with the help of the public key DS.pk.

Digital signatures provide authentication, which ensures the origin of the message, and non-repudiation, which means that a user who generates a signature cannot deny it.

A digital signature is said secure if it provides unforgeability. Unforgeability means that it is impossible for a malicious user, commonly named an adversary, to recover the secret key DS.sk from a signature or the public key and so forge a valid digital signature. The currently most commonly used signatures are the RSA signature and El-gamal signature. Both schemes' security depends on hardness assumptions. In cryptography, a hardness assumption can be considered as a hypothesis that a certain computational problem is hard to solve, for example, the discrete logarithm problem (DLP) [DH76]. Although digital signatures can prove the authenticity of a document or message, the identity of the signer is publicly known. Group and ring signatures give the possibility of proving the authenticity of a message and keeping its author hidden and so anonymous.

Stateful vs stateless schemes

The literature also distinguishes two types of digital signatures. On the one hand, we have stateless constructions. A stateless digital signature allows a signer to reuse his signing secret key DS.sk theoretically for an infinite number of signatures, without requiring any update. On the other hand, stateful digital signatures, the signer needs to keep a state up to date. This means, in practice, that after each signature, the signer needs to update his signing secret key DS.sk. Any errors made by the signer, (no-update of the signing key), will result in a security breach in the scheme. Therefore, in practice, stateless constructions are often preferred to stateful ones.

1.1.2 Privacy & anonymity

In cryptography, the premise of the definition of anonymity was given in 1983 by David Chaum [Cha83], where he introduced the concept of blind signature. David Chaum did not provide a formal definition of privacy, but rather just presented privacy as any information about an individual. His construction allows to prove that a payment is made without leaking any information on the payer. Through his design, David Chaum allowed individuals to perform anonymous payments, and therefore, suggested that to achieve privacy, we need to achieve anonymity. The definition of anonymity used in this thesis is as follows:

> "Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set"[PK01].

In the situation of a voting poll, we do not just need to assure authentication, but we also need to provide anonymity to the users. In cryptography, there exist two main cryptographic primitives which provide both authentication and anonymity. One of them is named group signature and appeared in the literature in the early 90's [CVH91] while the other, called ring signature, was created a few years later [RST01]. Both primitives allow the member of a group to sign a message anonymously. The verification of the signature only demonstrates that a member of the group has generated the signature without revealing their real identity.

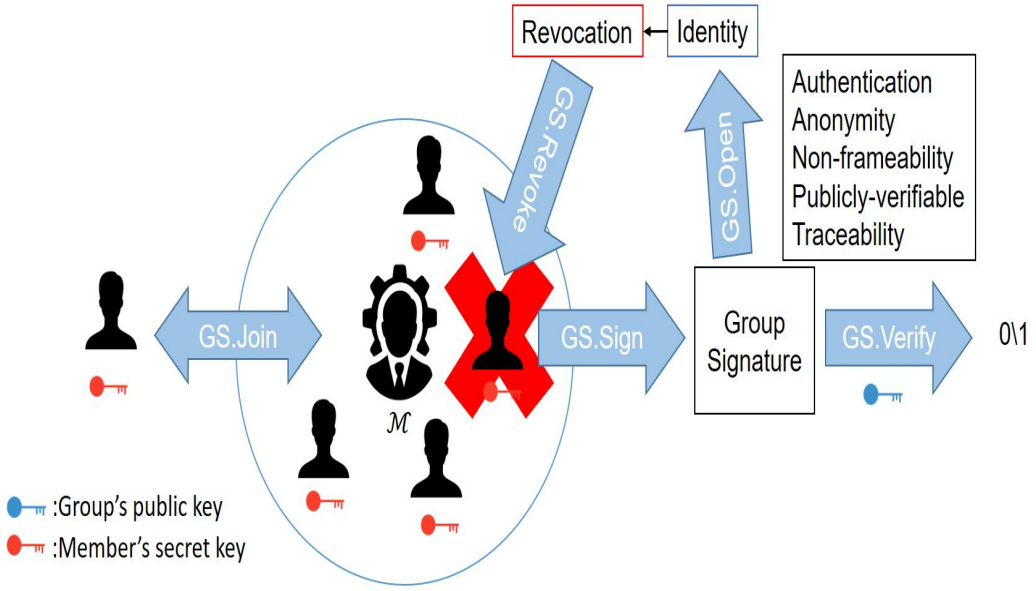


FIGURE 1.1: Group Signature

1.1.3 Group signature

Group signatures [CVH91] allow the member of a group of users to sign a message **anonymously**, and therefore, they are considered as a promising privacy-preserving instrument for real-world applications. A group signature allows members of a group to sign a digital message on behalf of the group while remaining anonymous. Thanks to this great promise, the research community has invested a great amount of effort to develop and improve this primitive. In this primitive, group members are managed by a central authority called the manager. This authority is responsible for initiating the group and every prospective member needs to interact with her in order to become a group member. The manager is also able to trace a signature and reveals the identity of its signer. In the literature, group signatures are divided into three different categories:

- Static [BMW03]: A static group signature has a fixed set of members determined during the setup phase and will remain unchanged during the whole group life.
- Partially dynamic [BSZ05]: The partially dynamic setting allows new members to join the group even after the completion of the setup phase.
- Fully dynamic [BCC+16]: The full dynamicity implies the revocation property that gives the central authority the possibility to revoke a member's signing ability. This means that in this setting, not only can new members join the group at any time, but members can also leave the group.

In this thesis, we will focus on fully dynamic group signatures (see an example in Figure 1.1) because we believe that they have more application opportunities as they allow more flexibility with the size of the group. In their work [BSZ05], Bellare et al. defined a group signature as a tuple of polynomial-time (PPT) algorithms (GS.KeyGen, GS.Sign, GS.Verify, GS.Open, GS.Join, GS.Revoke):

- GS.KeyGen: This algorithm generates the key pair of a user.
- GS.Sign: This algorithm is executed by a group member and generates a group signature for a message thanks to the member's secret key.

Table 1.1: Examples of group signatures

Group signature	Hardness Assumption
Chen et al. [CP94]	Discrete Logarithm Problem (DLP)
Camenisch et al. [CS97]	Discrete Logarithm Problem (DLP)
Camenish et al. [CM98]	Decision Diffie-Hellman assumption (DDH)
Ateniese et al. [ACJT00]	Strong RSA assumption
Boneh et al. [BBS04]	Discrete Logarithm Problem (DLP)

- GS.Verify: This deterministic algorithm verifies publicly the validity of a group signature for a message thanks to the shared group public key.
- GS.Open: This algorithm is executed by the manager and outputs the identity of a signer.
- GS.Join: This is an interactive algorithm between the manager and a prospective user who desires to join the group.
- GS.Revoke: This algorithm gives the possibility to the group manager to revokea corrupted members its ability to sign.

The challenges of designing applicable group signatures are not only limited to achieving the desired security properties but also require efficient algorithms and short signature sizes. More than just running time and actual signature size, it is crucial to optimize the influence of the size of the group, in other words the number of group members, on it.

1.1.4 Ring signature

Ring signatures, first defined in 2001 by Rivest et al. [RST01], are also considered to be one of the most valuable cryptographic primitives to ensure privacy. Ring signatures follow the same principle as group signatures, which means that they allow the member of a group of users to sign a message anonymously on behalf of a group, which is, in this case, name a **ring**. Ring signatures differ from group signatures as they do not require any central authority, and therefore, there is no joining process. The absence of a central authority allows users to sign in a **spontaneous** manner. The signer forms a group of his own choice and generates an anonymous signature. Thanks to their great promise to provide authenticity and anonymity, ring signatures have attracted a lot of interest from the research community. The current state-of-the-art of ring signature is summarized in Table 1.2 and formally defined in Chapter 2.

Ring signatures were firstly theorized in 2001 by Rivest et al. in [RST01] and, eight years later, Bender et al. extended this work by providing formal definitions of the construction and security of ring signature [BKM09]. The high-level representation of a ring signature is presented in Figure 1.2. Their work defines a ring signature as a tuple of PPT algorithms (RS.KeyGen,RS.Sign, RS.Verify):

- RS.KeyGen: This algorithm generates the key pair of a user.
- RS.Sign: This algorithm generates a ring signature thanks to the ring members' public keys and the signer secret key [BKM09].
- RS.Verify: This deterministic algorithm verifies the validity of a ring signature thanks to the ring members' public keys.

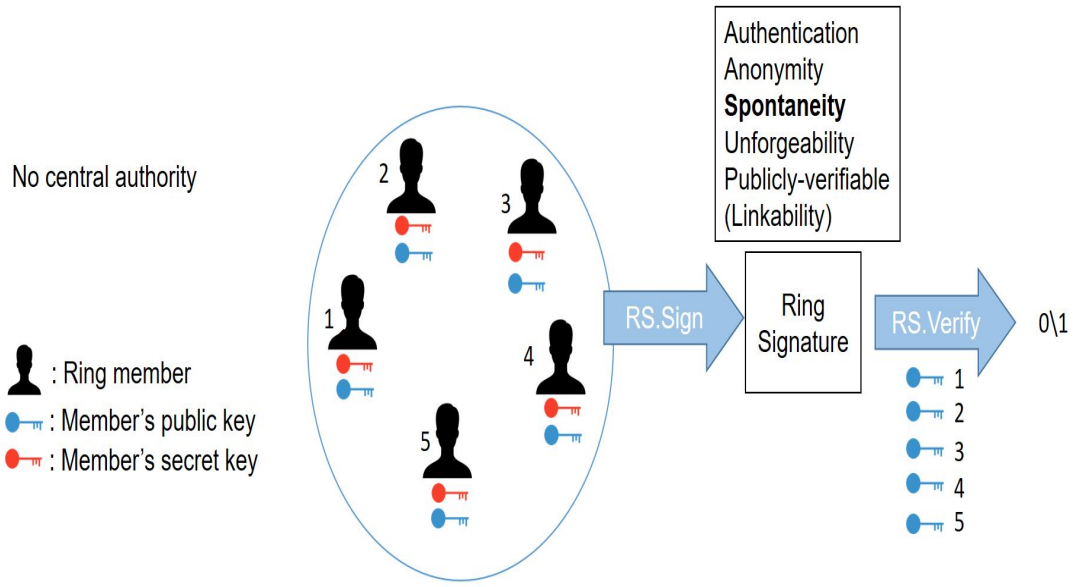


FIGURE 1.2: Ring Signature

TABLE 1.2: Examples of ring signatures

Ring signature	Hardness Assumption
Liu et al [LW05]	Discrete Logarithm Problem (DLP)
Au et al. [ACST06]	Integer factorization problem
Tsang et al. [TW05]	Integer factorization problem
Bootle et al. [BCC+15]	Decisionnal -Diffie-Helman problem (DDH)

Bender et al. [BKM09] argue that the security of ring signature should achieve anonymity, which means that we cannot determine who generates the signature, and unforgeability, which means that it is unfeasible to generate a valid signature for an adversary who does not belong to the ring. A ring signature is a type of digital signatures that can be performed by any member of a group of users, which all have keys. Therefore, a message signed with a ring signature is endorsed by someone in a particular group of users. An important advantage of ring signatures is spontaneity: No setup or joining processes are required to be part of the ring. One challenge in the construction of ring signatures is to achieve a small signature size. Works like Bootle et al.[BCC+15] attempt to address this issue. An example of application of the ring signature is the cryptocurrency Monero, which uses the linkable ring signature presented in [LW05]. The "linkability" property allows anyone to verify if two different signatures have been generated by the same member while still providing anonymity.

1.1.5 Post-quantum cryptography

Post-quantum cryptography [Ber11], [CCJ+16] is defined as the set of cryptographic algorithms that can resist an adversary who has access to a quantum computer. A quantum computer uses the properties of quantum physics to store data and perform computations, making it more powerful than a classical computer and hence enabling it to solve problems that are considered infeasible for a classical computer. For instance, a quantum computer is able to solve complex mathematical problems such as the factorization problem, which means that it can break the RSA algorithm

[RSA78]. To achieve post-quantum security, there exist different cryptographic settings that are commonly named **post-quantum cryptographic candidates**. The main candidates are the following:

- Lattice-based cryptography: This comprises all the cryptographic primitives that are constructed with lattices and whose security depends on hard mathematical problems around lattices. A common example of lattice-based cryptosystem is NTRU, which appeared in [HPS98].
- Symmetric primitives: The algorithms constructed on symmetric primitives are considered to be resistant to quantum computers because they are not impacted by the Grover's algorithm [Gro96] and therefore only require an increase of their secret keys. Symmetric primitives can be divided into the following categories:
  - Hash-based signatures: The most famous example of hash-based signatures is the one-time signature schemes presented in [Lam79] or in [Hül13]. The work of [Mer89] transformed one-time signature schemes into a multi-signature scheme thanks to binary trees commonly known as Merkle trees.
  - Secret-key cryptography: An example of secret-key cryptography is the block cipher AES [DR99].
- Code-based cryptography: This cryptographic primitives have a security which relies on the hardness assumption of a linear error correcting code. An example often cited is the CFS signature scheme [CFS01].

These are considered the three most important post-quantum candidates due to their better range of applicability. It is also important to reveal that there exist other post-quantum candidates such as multivariate polynomial cryptography, these are not in the scope of this thesis.

The advantages and disadvantages of the three primary post-quantum cryptographic candidates are presented in Table 1.3.

TABLE 1.3: Post-quantum cryptography candidates: summary

Post-quantum Candidates	Advantages	Disadvantages
Lattice-based	• Offers security guarantee • Relatively simple to implement • Parallelism is possible • Flexibility: Homomorphism is possible	• Difficult to provide precise estimations of the security some lattice schemes
Symmetric primitives	• Well defined post-quantum security • Efficient algorithms on classical computers	• Large signature size • No Homomorphism possible • Choice between performance and stateless feature • Possible security problems due to users' errors (stateful)
Code-Based	• Short signature size	• Large key sizes • Inefficient signing algorithm

1.1.6 Post-quantum group signature

When designing post-quantum group signatures, most researchers tend to choose lattice-based cryptography rather than symmetric primitives. This is principally due to the flexibility of lattice-based algorithm as mentioned in Table 1.3. The works of Libert et al. [LLM+16] [LLNW16], Gordon et al. [GKV10] and Ling et al [LNW15] [LNWX17] demonstrate this trend. The disadvantages of these group signatures, summarized in Table 1.3, are the difficulty to precisely estimate the security of lattice-based schemes and the inefficiency of their algorithms. Additionally, these works only present a theoretical construction and are limited in terms of the practical implementations for their constructed group signatures. A complete presentation, with a theoretical and applicability analysis, was done for the works of Del Pino et al.

Table 1.4: Group signatures and quantum computing

Group signature	Hardness Assumption	Quantum settings
Chen et al. [CP94]	Discrete Logarithm Problem (DLP)	Broken
Camenisch et al. [CS97]	Discrete Logarithm Problem (DLP)	Broken
Camenish et al. [CM98]	Decision Diffie-Hellman assumption (DDH)	Broken
Ateniese et al. [ACJT00]	Strong RSA assumption	Broken
Boneh et al. [BBS04]	Discrete Logarithm Problem (DLP)	Broken
Gordon et al. [GKV10]	Lattice-based	Secure
Alamelou et al. [ABCG17]	Code-based	Secure
Katz et al. [KKW18]	Symmetric-based	Secure

[DPLS18] and Esgin et al. [EZS+19]. These works are the state-of-the-art of post-quantum group signatures. The first work presented by Del Pino et al. introduces a group signature with has a constant signature size, which is interesting when working with extremely large group. The second work, which is presented by Esgin et al., provides the shortest signature size for stateless constructions. Both works are summarized in Table 1.5.

There also exist group signatures constructed with code-based cryptography such as in the work of Alamelou et al. [ABCG17]. The problem with this type of group signatures is summarized in table 1.3. Moreover, their work claims a signature size of 20MB, which cancels the expected advantage presented in Table 1.3 of code-based construction. Table 1.5 demonstrates that, in terms of signature size, their group signature is outperformed by group signatures constructed with symmetric primitives. Therefore, we omit code-based group signatures in our comparisons in the following chapters and only focus on the comparison with lattice-based constructions who achieve the best performances.

However, the construction of the zero-knowledge proof [GMW91] ZKBoo [GMO16], constructed only with symmetric primitives, shows that symmetric primitives can also be used to design post-quantum signatures and group signatures. ZKBoo uses the concept presented by Ishai et al, namely the "MPC in the head" [IKOS09]. From this, one can achieve new optimized zero-knowledge proofs such as [CDG+17], [AHIV17] or [KKW18]. The state-of-the-art post-quantum signature based on symmetric primitives are constructed by Chase et al., who built digital signature schemes by designing ZKB++ [CDG+17], an optimization of ZKBoo. In order to make ZKB++ a non-interactive zero-knowledge proof [BG92], they propose to use either Fiat-Shamir Transform [FS86] or Unruh Transform [Unr15].

A fully dynamic group signature constructed with ZKB++ was first proposed by Boneh et al. [BEF18]. Recently, a new draft of post-quantum group signature has been proposed by Katz et al. [KKW18] with an optimized non-interactive zero-knowledge proof based on symmetric primitives and “MPC-in-the-head". Both group signatures can be considered as the current benchmark in terms of dynamic group signature. Even though the group signature designed by Katz et al. [KKW18] is not dynamic, extending it by adding dynamicity should be relatively doable. Table 1.5 demonstrates the size of both group signature for different group sizes. It shows that the main challenge of designing group signature with symmetric primitives is

the large signature size. In 2018, El Bansarkhani and Misoczki. [EBM18] presented G-Merkle, a static group signature which is basically a modification of the Merkle signature [Mer89] and which achieves a significantly shorter signature size. However, G-Merkle faces too many limitations to find a possible application because the number of members is fixed and each member can only sign a limited amount of messages. G-Merkle's original paper advances a maximum group size of only 64 members who are to sign a maximum of 4096 signatures.

Table 1.5 shows that group signatures based on symmetric primitives must choose between performance and the stateless feature. For example, G-Merkle [EBM18] achieves the best signature size but is stateful. Stateless constructions such as the one presented by Katz et al. [KKW18] sacrifice their performance, in terms of signature size or algorithm efficiency, for the stateless feature.

The work of Boneh et al. [BEF19] and Katz et al. [KKW18] use the approach of a non-interactive proof of knowledge (NIZK) to construct a group and ring signature. With the NIZK approach, Katz et al. achieve the best performance as Table 1.5 demonstrates. Katz et al. use the standard technique to construct group signature with symmetric-based NIZK. Table 1.5 shows that it does not achieve great performance and is clearly outperformed by lattice-based constructions such as [EZS+19]. The signature size is large and the performance of the signing/verify algorithm is slow with a computation time of approximately 3 seconds. The standard approach being inefficient should motivate researchers to investigate other directions.

Table 1.5 summarizes the features and performance of all existing group signatures constructed from symmetric primitives and the **two most significant** lattice-based constructions [DPLS18],[EZS+19]. Both lattice-based works highlighted in Table 1.5 are considered as the current state-of-the-art when it comes to post-quantum group signatures. We evaluate group signatures through different aspects and characteristics. We compare their signature sizes in actual size, for example in KB, but also the influence of the group size on the signature size. We also compare their algorithm performances and their relation with the group size. The level of trust of the central authority is another major aspect of group signatures. When designing group signatures, the goal is to minimize the level of trust in the central authority as much as possible to avoid any risk of him framing any honest group members.

1.1.7 Post-quantum ring signature

Similar to the research on post-quantum group signatures, research on post-quantum ring signatures concentrates mainly on lattice-based ring signatures. The works [LLNW16], [TSS+18], [IKOS09], [WS11] or [EZS+19] confirm this trend. Lattice-based works have disadvantages, which are summarized in Table 1.3, but they currently outperform other schemes with shorter signature size of 103KB for a ring composed of 2^{13} users, particularly the work of Esgin et al. [EZS+19]. Table 1.7 presents the approximation of their signature size for different ring size.

The design of post-quantum ring signatures constructed with symmetric primitives, including cryptographic hash function and symmetric-key encryption schemes, was initially explored by Derler et al. [DRS18]. At the heart of their construction, there is the non-interactive zero-knowledge proof system ZKB++ [CDG+17], which allows the signer to proof his membership to the ring to generate a valid signature. Katz et al. [KKW18] provided an improved version of ZKB++, which also improves

[1]The times provided are taken from the original papers and were therefore implemented in different programming languages and executed on different types of machines

TABLE 1.5: Current state-of-the-art of group signatures for 128-bits of post-quantum security. N denotes the number of members in the group, N.A=Non Applicable, B=Maximum possible signatures per member, low= The group manager $\mathcal{M}$ cannot frame a signature, high=$\mathcal{M}$ can frame a signature

Schemes		[DPLS18]	[EZS+19]	[BEF19]	[KKW18]	G-Merkle [EBM18]
$\|\sigma\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$
$\|\sigma\|$ (KB)	$N = 2^7$	581	39	1370	315	2.72
	$N = 2^{10}$	581	54	1850	418	N.A
	$N = 2^{20}$	581	140	3450	770	N.A
$\|\mathsf{usk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$
$\|\mathsf{usk}\|$ (KB)		146	0.059	$0.032 \log N$	$0.032 \log N$	$2.2 \log B$
$\|\mathsf{gpk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\|\mathsf{gpk}\|$ (KB)		123	$N \cdot 11.1$	0.032	0.032	0.032
Sign (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$
Sign (ms)[1]		450	No data	No data	3000	N.A
Verify (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$
Verify (ms)[1]		169	No data	No data	3000	N.A
PQ canditate		Lattice	Lattice	Symmetric	Symmetric	Symmetric
Stateless		Yes	Yes	Yes	Yes	No
Dynamic		Static	Partially	Static	Fully	Static
Traceable		Yes	Yes	No	Yes	Yes
Non-Interactive verification		Yes	Yes	Yes	Yes	Yes
Level of Trust in $\mathcal{M}$		Low	Low	Low	Low	High

TABLE 1.6: Ring signatures and quantum computing

Ring Signature	Hardness Assumption	Quantum settings
Liu et al [LW05]	Discrete Logarithm Problem (DLP)	Broken
Au et al. [ACST06]	Integer factorization problem	Broken
Tsang et al. [TW05]	Integer factorization problem	Broken
Bootle et al. [BCC+15]	Decisionnal -Diffie-Helman problem	Broken
Libert et al. [LLNW16]	Lattice-based	Secure
Torres et al. [TSS+18]	Lattice-based	Secure
Esgin et al. [EZS+19]	Lattice-based	Secure
Gordon et al. [GKV10]	Lattice-based	Secure
Derler et al. [DRS18]	Symmetric-based	Secure
Katz et al. [KKW18]	Symmetric-based	Secure

TABLE 1.7: Post-quantum ring signature size

	Signature size			PQ
Ring size	2^7	2^{10}	2^{13}	candidate
Derler et al. [DRS18]	982KB	1.35KB	1.7MB	Symmetric
Katz et al. [KKW18]	285KB	388KB	495KB	Symmetric
Esgin et al. [EZS+19]	39KB	58KB	103KB	Lattice

the ring signature, especially in terms of signature size. Both schemes can be considered as the benchmark for possible ring signatures constructed with symmetric primitives. Table 1.7 shows that the challenges of designing ring signatures are exactly the same as those of group signature, in particular to **reduce the signature size**. But it also demonstrates gap with lattice-based ring signature is even larger than it is for group signatures.

1.2 Thesis scope

After studying the possible post-quantum candidates, we decided to work exclusively with symmetric key primitives to design privacy-preserving protocols. The rationale behind constructing privacy-preserving cryptographic primitives with symmetric primitives is given below:

- The research gap between lattice-based construction and symmetric-based construction is significant. Our initial review of literature showed that there were only three group signatures and two ring signatures based on symmetric primitives. Tables 1.4 and 1.6 summarize these works and show that the majority of researchers have worked with lattice-based cryptography. This research gap gives us more flexibility and the possible range of improvement of the state-of-the-art is larger.

- The symmetric primitive algorithms can be efficiently executed on classical computers. This gives us great hope to find applicable solutions to a real-life system. Future research directions indicate that systems need to transfer from “classical" security to post-quantum security before the spread and democratization of quantum machines.

- The post-quantum security of symmetric primitives is well understood and well defined by the research community, however, it remains difficult to estimate of the security of lattice schemes.

- The last element that convinced us to work with symmetric primitives only is that the security of symmetric-based protocols depends only on the security of the applied symmetric primitives. Indeed, there exist multiple ways to design the protocols and if one of them is broken, it does not mean that the others will also break.

After highlighting the advantages of symmetric primitives, we explain in more detail what we consider a symmetric primitive. We previously separated (see Chapter 1.1.5) them into two categories: Hash-based cryptography and Secret-key cryptography.

- Hash-based cryptography: In this work, we consider hash functions that satisfy Definition 2.2.1 on page 17. Our goals are to design applicable secure construction, we do not “idealize" hash functions and therefore, we do not consider them

as random oracles [BR93]. A random oracle generates a uniformly distributed random output for a query. In this work, we consider hash functions that have been standardized as SHA256 or hash functions that have been proven secure that offer properties allowing an improvement of performances, for example, the hash function Poseidon [GKR+21].

- Secret-key cryptography: In this work, we consider block ciphers that satisfy Definition 2.3.1 on page 18. For the same reason of targeting secure applications for our construction, we do not consider block ciphers as ideal ciphers. We consider the standardized block cipher AES and proven secure block cipher that provided interesting properties to improve the performance of our construction such as LowMC [ARS+15].

In this thesis, we allow our adversary, which is represented by $\mathcal{A}$, to access a quantum computer. However, we do not consider that she has access to a quantum oracle [BDF+11] but only to a classical oracle. The reason behind this choice is that we aim to design applicable constructions and there is currently no existing scenario wherein the real-word where an adversary could access a quantum generation of a group or ring signature. Nevertheless, when it comes to choosing the symmetric primitives, we try as much as possible to use the one that has been studied with quantum oracles. Obviously, we mention the different levels of post-quantum security of each of the primitives used.

1.3 General research aim

The main target of this thesis is to design different post-quantum symmetric-based group and ring signatures to fill the research gap between lattice-based constructions and symmetric-based construction. Thereby, newly designed protocols constructed with symmetric primitives will be generated from this target. These will be evaluated in terms of security, performance and applicability.

1.3.1 Problem identification

Designing post-quantum group signatures and ring signatures based only on symmetric primitives involves having to face different challenges.

- The first challenge is to achieve a short signature size. The smallest signature size was achieved in [EBM18] but this construction is stateful and not flexible (see Chapter 3.2).
- The next challenge is to provide efficient algorithms, which has not been achieved yet by the current state-of-the-art. For example, Katz et al. [KKW18] are limited with the inefficiency of their algorithms while El Bansarkhani and Misoczki [EBM18] provide an inefficient setup process.
- The last challenge, specific to cryptographic schemes based on symmetric primitives, is to avoid having to use a state. This challenge has been addressed by Boneh et al. [BEF19] and by Katz et al. [KKW18] but their schemes are outperformed by a factor of a least 100 when it comes to the signature size compared to stateful constructions like [EBM18].

Therefore, this thesis targets the design of quantum-safe cryptographic primitives which ensure privacy. In particular, it focuses on designing group and ring signature

schemes with the ambition to work only with symmetric primitives to provide quantum security. The goals of each design are first to reach a short signature size and then to design efficient algorithms.

1.3.2 Research objectives

This thesis focuses on the study of group and ring signature schemes because of their promise in privacy preservation. It is divided into two main objectives. The first objective focuses on the design of group signatures (O1) and the second is to design ring signature schemes (O2).

O1: Group Signature

(a) Construct fully dynamic group signature schemes and efficient algorithms based on symmetric primitives with the target of providing a significantly shorter signature size than the current state-of-the-art.

(b) Prove the security proofs of our construction with respect to traceability, anonymity, and unforgeability to provide formal guarantees of the reliability of the fully dynamic group signatures schemes.

(c) Evaluate the performances of all group signature schemes' algorithms to show that we reduced the signature size without compromising the algorithms' efficiency.

O2: Ring Signature

(a) Construct ring signature schemes based on symmetric primitives with the target of significantly reducing the signature size compared to the current state-of-the-art.

(b) Prove the security of our construction with respect to anonymity and unforgeability to provide formal guarantees of the reliability of the constructed ring signature schemes.

(c) Evaluate the performances of the ring signature schemes' algorithms to show that we reduced the signature size without compromising the algorithms' efficiency.

(d) Optimize and extend ring signature schemes by adding the linkability property, which is a required property for an application in Blockchain technology.

1.4 Research contributions

This research project contributes to the research community by designing three new group signatures and a generic ID-based ring signatures from which we propose two applicable constructions.

1.4.1 Research Objective O1: Group Signature

Research Objective O1 generated three fully functional post-quantum group signature schemes, namely DGM, (T-)SGS and xGS. They were all designed to achieve the shortest size and the best applicability possible. The results and the comparison with the current state-of-the-art are summarized in Table 7.1 on page 139, which is an updated version of Table 1.5 presented in this chapter. A summary of theses contributions can be summarized as follows:

1. *Dynamic & Revocable Group Merkle signature (DGM)*

   DGM:The first group signature is named DGM, which stands for Dynamic & Revocable Group Merkle signature (see Chapter 3). DGM is an extension of the static group signature G-Merkle. DGM is the first fully dynamic group signature based on symmetric primitives. It achieves a competitive signature size and members do not require to go through an update procedure after a new member joined the group. The full dynamicity is ensured thanks to an innovative technique implementing symmetric puncturable encryption. This innovation avoids the use of the public revocation lists, which is the traditional approach and can be impractical. DGM is summarized in Table 7.1 on page 139. This contribution culminates in our paper "DGM: A Dynamic & Revocable Group Merkle Signature " which was published at the conference European Symposium on Research in Computer Security (ESORICS) [ESOa] in 2019.

2. *(Traceable-) SPHINCS Group Signature (T-SGS)*

   The second group signature is the stateless (T-)SGS, which stands for (Traceable-) SPHINCS Group Signature (see Chapter 4). It was designed to solve the issues still present in DGM, namely to suppress the interactive verification procedure and the stateful nature of DGM. (T-)SGS was derived from the traditional stateless digital signature SPHINCS+. We first made a simple transformation to generate a stateless group signature without the traceability property. Then, an accumulator was integrated to the hyper-tree structure in order to ensure traceability, which gave the fully functional group signature named T-SGS. (T-)SGS's details and its comparison with other works is summarized in Table 7.1 on page 139.

3. xGS*: Efficient Post-Quantum Group Signature from Symmetric Primitives with User-Controlled Linkability*

   The last group signature is named xGS *(see Chapter 5). It solves the main issue of DGM with its non-interactive verification procedure. Additionally, it is the first post-quantum group signature proposing "user-controlled" linkability.* xGS *allows a group member to sign* x *unlikable group signatures.* xGS *gives group members the responsibility to generate their secret signing keys, which allows* xGS *to work with larger groups than DGM.* xGS *is compared and summarized in Table 7.1 on page 139. This contribution culminates in the paper "Efficient Post-Quantum Group Signature from Symmetric Primitives with User-Controlled Linkability" submitted at the conference ACM ASIACSS 2022 [ACMa].*

1.4.2 Research Objective O2: Ring Signature

Research Objective O2 generated the first ID-based ring signature (IDRS) from symmetric primitives, in which we derived two constructions named PicRS and xGS.

4. *Post-Quantum ID-based Ring Signatures from Symmetric-key primitives*

   The first generic construction of ID-based ring signature (IDRS) from symmetric primitives is presented in Chapter 6. The "ID-based" feature of our construction has been added to extend the applicability range to ring signature as it achieve better spontaneity than traditional ring signatures. We designed a one-way circuit involving an accumulator and the verification procedure of a digital signature scheme. We also implemented the generic construction by proposing two

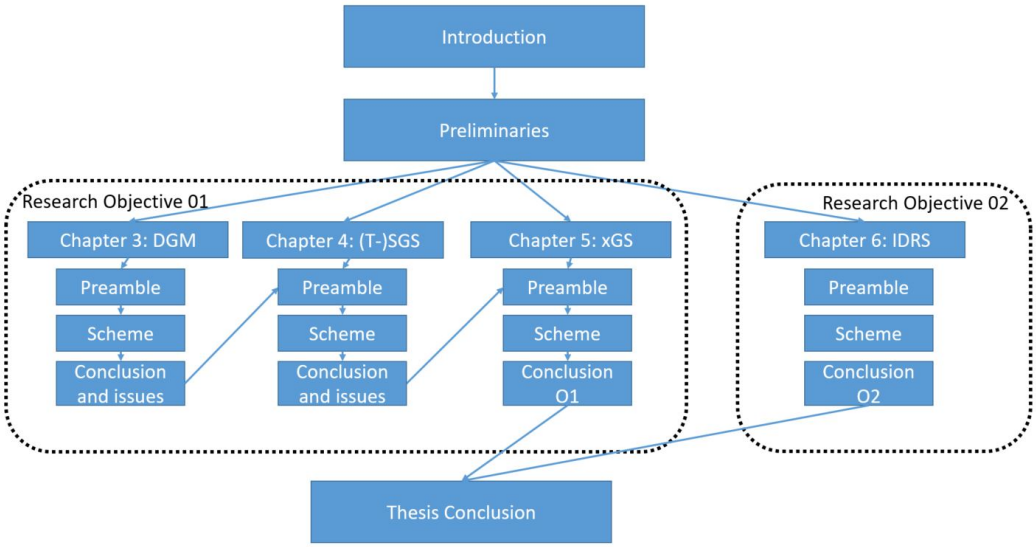


FIGURE 1.3: Thesis Road-map

constructions. The first construction, PicRS*, is derived from the Picnic digital signature and the second construction,* XRS*, is derived from the XMSS digital signature. Table 7.2 on page 140 situates our work with the current state-of-the-art. The major notable contribution of this work is that both* PicRS *and* XRS *implementations enjoy a nearly constant signature size while the current-state-of-the-art has a signature size growing linearly with the ring size. This contribution culminates in the paper "Post-Quantum ID-based Ring Signatures from Symmetric-key primitives" published in the conference Applied Cryptography and Network Security (ACNS) 2022 [ACN]*

1.5 Thesis structure

This section presents the thesis organization, which is summarized in Figure 1.3.

After this introductory Chapter 1, the thesis follows with Chapter 2, which presents all the definitions of cryptographic primitives used across this research project. In particular, it provides the formal definition of group and ring signature and their security requirements, including their definitions and security experiments. This chapter defines symmetric primitives such as cryptographic hash function or Pseudorandom function. Primitives used only in a specific chapter will be defined in their corresponding chapters.

Chapters 3, 4 and 5 present the contributions ensuing from Research Objective O1. At the end of each of these chapters, we update Table 1.5 to demonstrate and summarize the contributions of each of these three chapters.

Chapter 3 presents the first contribution of the research project: the group signature named DGM. The chapter starts by contextualizing this contribution before presenting the scheme in detail. To design DGM, we started from the static group signature G-Merkle, which is not applicable but achieves competitive signature size. We extended G-Merkle to a fully dynamic group signature thanks to the separation of different types of trees and the use of symmetric puncturable encryption to revoke members. The chapter concludes with the disadvantages of DGM, which orientates the future objectives and the research direction of this project.

Chapter 4 presents the stateless construction (T-)SGS, which is an attempt to transform the digital signature SPHINCS+ into a group signature. This starts with a reminder of the issues that (T-)SGS tries to solve, namely the stateful feature and the interactive verification process present in DGM. The starting point of this contribution was the current state-of-the-art digital signature SPHINCS+, which we extended to a non-traceable group signature before integrating an accumulator to the scheme to provide the traceability property. After the schemes presentation and analysis, this chapters concludes with the issues at the heart of T-SGS construction, which motivates a change in our approach for Chapter 5.

The last contribution resulting from Research Objective O1 is presented in Chapter 5. The chapter introduces the newest group signature designed in this project: xGS. xGS is the first post-quantum group signature to provide "user-controlled" likability and to solve the issues of our first construction, DGM. We started from our first designed group signature and tried to suppress the interaction in the verification procedure. This was achieved by separating the keys generation between the manager and the user. The chapter concludes with a presentation of the contributions of Research Objective O1.

The results of Research Objective O2 are presented in Chapter 6. The chapter presents the final contribution of this thesis. It starts with an important contextualization of the concept of ID-based ring signature and presents our generic construction from symmetric primitives. This chapter continues with the presentation of the first ID-based ring signatures from symmetric primitives, PicRS and XRS, which are derived from our ID-based ring signature and it finishes with their evaluation.

Finally, this thesis concludes with Chapter 7, which summarizes all the contributions of the thesis, highlights their limitations and discusses future works possible in this domain.

Chapter 2

Preliminaries

This chapter defines all primitives used by this research thesis. This chapter formally presents the definition of cryptographic hash functions, symmetric encryption schemes and their security requirements, the concept of digital signature and its derived one-time signature scheme (OTS). It also defines pseudorandom functions and non-interactive zero-knowledge proof systems (NIZK). This chapter concludes with the formal definition of group and ring signatures, which are the two privacy-preserving protocols that have been designed in this research project. It is important to note that that the cryptographic primitives relevant to only **one specific** chapter are formally defined in their corresponding chapter.

2.1 Post-quantum security parameters λ

In this research thesis, λ represents the **post-quantum** security parameter. Grover's search algorithm [Gro96] is a quantum algorithms which allows to find the unique input of a one-way function by brute-forcing in just $\mathcal{O}(\sqrt{|P|})$ iterations, where $|P|$ is the size of the domain function. This means that quantum computers could invert a cryptographic hash function or recover a secret key of in $\mathcal{O}(\sqrt{|P|})$ iterations for a hash/key size of $\log_2 |P|$ bits. Consequently, if we desire a level of λ-bits of post-quantum security, we require to have domain of 2λ bits in order to face Grover's search algorithm.

2.2 Cryptographic hash function

A cryptographic hash function is a one-way function that takes as input a binary string of any length and outputs a binary string of a fixed length of 2λ bits (see Section 2.1), where λ is the level of post-quantum security. The cryptographic hash function is presented by H for the rest of this research thesis. The formal definition of a cryptographic hash function can be presented as follows.

Definition 2.2.1 (Cryptographic Hash Function)**.** *A cryptographic hash function*

$$\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}, b = \mathsf{H}(a) \tag{2.1}$$

takes as input a message a of any length and outputs the hash value b of length 2λ bits. A cryptographic hash function fulfills the three following properties

- *Pre-image resistance (one-wayness): given a hash value b, where $b = \mathsf{H}(a)$ for a uniformly random $a \in \{0,1\}^*$, it is computationally infeasible (in polynomial-time) to find a such that $b = \mathsf{H}(a)$.*

- *Second Pre-image Resistance: knowing a pair* $(a_0, \mathsf{H}(a_0))$ *for a uniformly random* $a_0 \in \{0,1\}^*$*, it is computationally infeasible to find another input* $a_1 \in \{0,1\}^*$ *such that* $\mathsf{H}(a_1) = \mathsf{H}(a_0)$.
- *Collision Resistance: it is computationally infeasible to find two different inputs* a_0 *and* a_1 *such that* $a_0 \neq a_1$ *resulting with the same hash value* $b = \mathsf{H}(a_0) = \mathsf{H}(a_1)$

2.3 Symmetric-key encryption scheme

A Symmetric-key encryption scheme (SE) is used to ensure the confidentiality of a message. The term "symmetric" means to encrypt a plaintext and decrypt a ciphertext using the **same** secret key, which is a string of 2λ bits (see Section 2.1). All algorithms of symmetric-key encryption schemes start with SE as a prefix for the rest of this research thesis. The formal definition of a symmetric-key encryption key scheme can be presented as follows

Definition 2.3.1 (Symmetric Encryption [SYL+18]). *A symmetric encryption scheme* SE *is defined by three polynomial algorithms* (SE.KeyGen, SE.Enc, SE.Dec).

$\mathsf{SE.sk} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$*: This algorithm takes as input the post-quantum security parameter* λ *and outputs a secret key* $\mathsf{SE.sk} \in \{0,1\}^{2\lambda}$.

$c \leftarrow \mathsf{SE.Enc}(p, \mathsf{SE.sk})$*: This algorithm takes as input the secret key* SE.sk *and a plaintext* p*. It outputs the ciphertext* c.

$p \leftarrow \mathsf{SE.Dec}(c, \mathsf{SE.sk})$*: This algorithm takes the decryption takes as input the secret key* SE.sk *and a ciphertext* c*. It outputs the plaintext* p.

Definition 2.3.2 (IND-CPA security). *A symmetric encription scheme* SE *is IND-CPA secure if for security parameters* λ*, a PPT adversary* $\mathcal{A}$ *has an advantage*

$$\mathsf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) = |\Pr[\mathsf{Exp}^{IND\text{-}CPA}_{\mathcal{A},\mathsf{SE}}(\lambda) = 1] - 1/2| < \mathsf{negl}(\lambda) \tag{2.2}$$

where $\mathsf{Exp}^{IND\text{-}CPA}_{\mathcal{A},\mathsf{SE}}$ *is the security experiment described in Figure 2.1.*

$\mathsf{Exp}^{IND\text{-}CPA}_{\mathcal{A},\mathsf{SE}}(\lambda)$:	$\mathcal{O}^{\mathsf{Enc}}_{\mathsf{SE.sk}}(p)$
$b \xleftarrow{\$} \{0,1\}; \mathsf{SE.sk} \xleftarrow{\$} \{0,1\}^{2\lambda}$ $(p_0, p_1, c) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}}_{\mathsf{SE.sk}}(\cdot)}(\lambda)$ with $\lvert p_0\rvert = \lvert p_1\rvert$ $c' \leftarrow \mathsf{SE.Enc}(p_b, \mathsf{SE.sk})$ $b \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}}_{\mathsf{SE.sk}}(\cdot)}(c', c)$ **Return** 1	$c \leftarrow \mathsf{SE.Enc}(p, \mathsf{sk})$ **Return** c

FIGURE 2.1: IND-CPA security of SE

2.4 Digital signature

A digital signature (DS) scheme aims to authenticate a message m. It provides authentication, which ensures the origin of the message and non-repudiation, which ensures that a signer cannot deny his signature. The formal definition of DS is presented below and for the rest of this research project, all algorithms of DS have the prefix DS.

Definition 2.4.1 (Digital signature)**.** *A digital signature scheme* DS *is composed by the following algorithms:*

$(\mathsf{DS.pk}, \mathsf{DS.sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$*: This algorithms takes as input the post-quantum security parameter* λ *and outputs the keypair* (DS.pk, DS.sk)*.*

$\mathsf{DS}.\sigma \leftarrow \mathsf{DS.Sign}(m, \mathsf{DS.sk})$*: This algorithms takes as inputs a message* m *to be signed and a secret key* DS.sk*. It outputs a valid digital signature* $\mathsf{DS}.\sigma$*.*

$0/1 \leftarrow \mathsf{DS.Verify}(m, \mathsf{DS}.\sigma, \mathsf{DS.pk})$*: This algorithms takes as inputs the signed message* m*, a digital signature* $\mathsf{DS}.\sigma$*, and the public key* DS.pk*. It outputs* 1 *if* $\mathsf{DS}.\sigma$ *is valid and* 0*, otherwise.*

We say that DS is secure if it reaches existential unforgeability under adaptive chosen message attacks (EU-CMA) which can be defined as follows.

Definition 2.4.2 (EU-CMA)**.** *A digital signature scheme* DS *reaches existential unforgeability under adaptive chosen message attacks (EU-CMA) security if and only if for a security parameter* λ *and an integer* $q = polynomial(\lambda)$ *the advantage* $\mathsf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) = \Pr[\mathsf{Exp}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{2.3}$$

where $\mathsf{Exp}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda)$ *is defined in Figure 2.2.*

$\mathsf{Exp}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda)$:
$(\mathsf{DS.pk}, \mathsf{DS.sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
$(m', \mathsf{DS}.\sigma) \leftarrow \mathcal{A}^{\mathsf{DS.Sign}(\mathsf{DS.sk})}(\mathsf{DS.pk})$
The set $\mathcal{Q}$ contains q pairs $(m', \mathsf{DS}.\sigma)$ generated by DS.Sign.
if $1 \leftarrow \mathsf{DS.Verify}(\mathsf{m}', \mathsf{DS}.\sigma', \mathsf{DS.pk})$ and $m' \notin \mathcal{Q}$
then return 1

FIGURE 2.2: EU-CMA security experiment

2.5 One-time signature scheme

A one-time signature scheme (OTS) [Lam79] follows Definition 2.4.1 of a digital signature scheme, but as its name suggests, the security is guaranteed if an OTS key pair is used **once**. In this thesis, we will take into consideration only OTS constructed with symmetric primitives only and respect the following definition.

Definition 2.5.1 (One-time signature scheme (OTS))**.** *An OTS scheme is a digital signature scheme constructed with the help of three polynomial time algorithms:* OTS.KeyGen*,* OTS.Sign *and* OTS.Verify*.*

- $(\mathsf{OTS.pk}, \mathsf{OTS.sk}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$*: This algorithms takes as input the security parameter* λ *and generates one key pair, which is composed of one public key* OTS.pk*, and one secret key* OTS.sk*.*
- $\mathsf{OTS}.\sigma \leftarrow \mathsf{OTS.Sign}(m, \mathsf{OTS.sk})$*: This algorithms takes as input a digital message* m *and a OTS secret key* OTS.sk*. It signs a digital message* m *with the secret key* OTS.sk *and outputs a valid OTS signature* $\mathsf{OTS}.\sigma$*.*

- $0/1 \leftarrow \mathsf{OTS.Verify}(m, \mathsf{OTS}.\sigma, \mathsf{OTS.pk})$: *This deterministic algorithms takes as input a digital message m, a OTS signature $\mathsf{OTS}.\sigma$ and a OTS public key $\mathsf{OTS.pk}$. It verifies the validity of $\mathsf{OTS}.\sigma$ for the message m. It outputs 1 in case of a valid signature and 0 otherwise.*

By definition, a OTS achieves unforgeability under adaptive chosen message attacks (EU-CMA) presented in Definition 2.4.2, if the key pair $\mathsf{OTS.pk}, \mathsf{OTS.sk}$ is used only once.

2.6 Pseudorandom function

This section introduces pseudorandom functions (PRF), which generate a pseudorandom output from a secret key and a public seed. PRF is a one-way and deterministic function, which means that for the same seed and same secret key, a PRF always outputs the same output, and a function is a PRF if this output is computationally indistinguishable from a truly random function. The formal definition of a PRF is presented below.

Definition 2.6.1 (Pseudorandom function (PRF))**.** *A function*

$$\mathsf{PRF} : \{0,1\}^{2\lambda} \times \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}, c \leftarrow \mathsf{PRF}(a, b) \tag{2.4}$$

is a pseudo-random function which takes as input a key a with a length of 2λ bits (see Section 2.1), where λ is the level of post-quantum security, and a message b of any length, and outputs the value c with a length of 2λ bits if for a PPT adversary $\mathcal{A}$, his advantage

$$\mathsf{Adv}^{PRF}_{\mathcal{A},\mathsf{PRF}}(\lambda) = |\Pr[\mathcal{A}^{PRF(a,.)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{f(.)}(1^\lambda) = 1]| < \mathsf{negl}(\lambda), \tag{2.5}$$

where f is random function.

2.7 Accumulator

An accumulator [DRS18, BEF19] allows to concatenate a set of element into a shorter and common accumulated value. This accumulated value can be used to prove or deny the membership of an element into the accumulator. The formal definition of an accumulator is presented below and is ensued from [BEF19]. The accumulator algorithms have A as a prefix.

Definition 2.7.1 (Accumulator)**.** *An accumulator [BEF19] A is defined by the following PPT algorithms:*

$\mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda)$: *The setup algorithm takes as input the post-quantum security parameter λ and outputs the public key $\mathsf{A.pk}$.*

$(\mathsf{A}_\mathcal{X}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X})$: *The evaluation algorithm takes as inputs the public key $\mathsf{A.pk}$ and the set $\mathcal{X}$ and outputs the accumulator $\mathsf{A}_\mathcal{X}$ and an updated public key $\mathsf{A.pk}$ for the new accumulated set $\mathcal{X}$.*

$\mathsf{w}_{x_i} / \perp \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_\mathcal{X}, \mathcal{X}, x_i)$: *The witness generation algorithm takes as inputs the public key $\mathsf{A.pk}$, the accumulator $\mathsf{A}_\mathcal{X}$, the set $\mathcal{X}$, and an element x_i. It outputs the witness w_{x_i} if $x_i \in \mathcal{X}$ and $\perp$ otherwise.*

$0/1 \leftarrow \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_{\mathcal{X}}, \mathsf{w}_{x_i}, x_i)$: *The verification algorithm takes as inputs the public key* $\mathsf{A.pk}$*, the accumulator* $\mathsf{A}_{\mathcal{X}}$*, the witness* w_{x_i}*, and the element* x_i*. It outputs* 1 *if* w_{x_i} *is a valid witness for* $x_i \in \mathcal{X}$ *and* 0 *otherwise.*

Similar to [BEF19], we assume that an accumulator A *achieves correctness, soundness and collision-freeness as follows:*

- *Correctness: For all* $x \in \mathcal{X}$ *with* $\mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda)$, $(\mathsf{A}_{\mathcal{X}}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X})$*, and* $\mathsf{w}_x \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_{\mathcal{X}}, \mathcal{X}, x)$*, we have that* $\Pr[0 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_{\mathcal{X}}, \mathsf{w}_x, x)] < \mathsf{negl}(\lambda)$.
- *Soundness: For all* $x \notin \mathcal{X}$ *with* $\mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda)$, $(\mathsf{A}_{\mathcal{X}}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X})$*, and* $\mathsf{w}_x \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_{\mathcal{X}}, \mathcal{X}, x)$*, we have that* $\Pr[1 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_{\mathcal{X}}, \mathsf{w}_x, x)] < \mathsf{negl}(\lambda)$.
- *Collision-freeness: An accumulator achieves collision-freeness if for a PPT adversary* $\mathcal{A}$ *we have:*

$$\begin{aligned}\Pr[\mathsf{A.Verify}(\mathsf{A.pk}^*, \mathsf{A}_{\mathcal{X}^*}, \mathsf{w}^*_{x_i}, x_i^*) = 1 \wedge x_i^* \notin \mathcal{X}^* | \mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda),\\ (\mathcal{X}^*, \mathsf{w}^*_{x_i}, x_i^*) \leftarrow \mathcal{A}(\mathsf{A.pk}), (\mathsf{A}_{\mathcal{X}^*}, \mathsf{A.pk}^*) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X}^*)] < \mathsf{negl}(\lambda).\end{aligned} \tag{2.6}$$

2.8 Non-interactive zero-knowledge proof system

This section defines non-interactive zero-knowledge proof systems [GMW91] (NIZK) which are useful cryptographic tools. Zero-knowledge proof systems were first designed as an interactive protocol [GMR89] between a prover and a verifier. The great interest coming from this primitive relies on the fact it allows to proof allow to prove the knowledge of a witness that satisfies certain properties without leaking any information except of the validity of this witness. NIZK achieves this zero-knowledge property but without requiring any interaction with the verifier. This means that the generated proof for a witness is publicly verifiable. We present the definition of NIZK as follows and all algorithms of NIZK have NIZK as a prefix.

Definition 2.8.1 (Non-interactive Zero-knowledge Proof System (NIZK))**.** *Non-interactive zero-knowledge proof system (NIZK) [BHSB19] aims to prove that a public statement* x *and a private witness* w *belong to a defined relation* R *(i.e.* $(x, w) \in R$*). We also let* $\mathcal{L}_R = \{x | \exists w \text{ s.t. } (x, w) \in R\}$*. NIZK consists of the following three algorithms:*

$\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$: *This generates the common reference string* crs *from the security parameters* λ*.*

$\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w)$*: This generates a proof* π *for the common reference string* crs*, the statement* x *and the witness* w *that satisfies the relation* R *(to be more specific, we have* $(x, w) \in R$*).*

$0/1 \leftarrow \mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi)$*: This returns* 1 *if the proof* π *based on the common reference string* crs *and the public statement* x *is valid,* 0 *otherwise.*

Remark 1. *In this research thesis, we omit the use of the common reference string* crs*.*

Definition 2.8.2 (Completeness). *A proof system* NIZK *achieves completeness if for an adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{Comp}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$ *is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Comp}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = \Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda); (x,\pi) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) : \\ \mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi) = 0 \wedge x \in \mathcal{L}_R] < \mathsf{negl}(\lambda).\end{aligned} \tag{2.7}$$

Definition 2.8.3 (Soundness). *A proof system* NIZK *achieves soundness if for an adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{Sound}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$ *is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Sound}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = \Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda); (x,\pi) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) : \\ \mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi) = 1 \wedge x \notin \mathcal{L}_R] < \mathsf{negl}(\lambda).\end{aligned} \tag{2.8}$$

Definition 2.8.4 (Zero-knowledge). *The verifier learns nothing but the validity of the statement. A* NIZK *is said to be zero-knowledge if the advantage of* $\mathcal{A}$ $\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$ *is:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = |\Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda) : \mathcal{A}^{\mathsf{NIZK.Prove}(\mathsf{crs},x,w)} = 1] - \\ \Pr[(\mathsf{crs}^*, \pi^*) \leftarrow \mathsf{NIZK.Sim}(1^\lambda, x) : \mathcal{A}^{(x,\mathsf{crs}^*,\pi^*)} = 1]| < \mathsf{negl}(\lambda),\end{aligned} \tag{2.9}$$

where $(\mathsf{crs}^*, \pi^*) \leftarrow \mathsf{NIZK.Sim}(1^\lambda, x)$ *is a simulator that takes as input the security parameter* λ *and the statement* x *and outputs a common reference string* crs^* *and a simulated proof* π^**.*

Definition 2.8.5 (Simulation extractability). *A proof system* NIZK *satisfies simulation extractability if there is algorithms* S *and* NIZK.Sim *satisfying the zero-knowledge definition and an extractor* $\mathcal{E}$ *such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SimE}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = \Pr[(x,\pi) \leftarrow \mathcal{A}^{\mathsf{S},\mathsf{NIZK.Sim}}(1^\lambda); w \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}, t, x, \pi) : \\ \mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi) = 1 \wedge (x,\pi) \notin \mathcal{Q} \wedge (x,w) \notin R] < \mathsf{negl}(\lambda)\end{aligned} \tag{2.10}$$

where $\mathcal{E} = ((\mathsf{crs}, t) \leftarrow \mathsf{NIZK.ExtGen}(1^\lambda, t), w \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}, t, x, \pi)$ *is a extractor,* $\pi \leftarrow \mathsf{S}(t, x)$*,* t *is a state and* $\mathcal{Q}$ *is the list of queries done by* $\mathcal{A}$ *to* NIZK.Sim.

2.9 Group signature

This section introduces the formal definition of a fully dynamic group signature (GS) inspired from [BHSB19], [BCC+16]. A group signature is composed of three different identities.

- Manager $\mathcal{M}$: The trusted central authority, which starts the group, allows prospective users to join the group, has the possibility to divulge the identity of a signer and can remove the capacity to generate a valid signature for a corrupted member. In this research thesis, we defined only one manager instead of spreading the joining and tracing authorities such as [BCC+16]. For the rest of this research project, the manager will be represented by $\mathcal{M}$.
- Member: One of the identities of the group that can anonymously generate a signature on behalf of the group. Each member needs to execute an interactive joining process with the central authority $\mathcal{M}$ to join the group. Each group member owns his personal secret key, but shares a common group public key. In this work, we assume that all interactions between a (prospective) member and $\mathcal{M}$ take place under a secure channel.

- Verifier: This last identity which can be a group member or $\mathcal{M}$ or neither, can only verify the validity of a group signature using public parameters.

The architecture of a group signature can be illustrated in Figure 1.1 on page 3

Definition 2.9.1 (Fully dynamic group signature)**.** *A fully dynamic group signature* GS *is defined by the following polynomial-time algorithms:*

$(\mathsf{gpk}, \mathsf{gsk}, \boldsymbol{members}, \mathsf{param}) \leftarrow \mathsf{GS.Setup}(1^\lambda)$*: This algorithm, which is executed by the central authority* $\mathcal{M}$*, takes as inputs the security parameter* λ *and outputs the group public key* gpk*, the master secret key* gsk*, an empty list* ***members*** *which will store the group members' identities upon their joining and the parameters* param *necessary for the working of the group.*

$(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{GS.MKeyGen}(1^\lambda)$*: This algorithm, which is executed by a prospective group member before initiating the joining procedure, takes as input the security parameter* λ *and outputs the user's public key* upk *and the user's secret key* usk*.*

$\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \boldsymbol{members} \rangle_{\mathcal{M}} \leftarrow \mathsf{GS.Join}(\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \boldsymbol{members} \rangle_{\mathcal{M}})$*: This is an interactive protocol between* $\mathcal{M}$ *and an individual user* $\mathcal{U}$*, who wants to join the group. The user takes as inputs a user secret key and a user public key; and* $\mathcal{M}$ *takes as inputs the group public key* gpk*, the group secret key* gsk *and the list of members* ***members****. At the end of this protocol, the user updates his* usk *while* $\mathcal{M}$ *updates the group public key* gpk*, the group secret key* gsk *and the list* ***members*** *by adding* $\mathcal{U}$ *to it.*

$\sigma \leftarrow \mathsf{GS.Sign}(m, \mathsf{usk}, \mathsf{upk}, \mathsf{gpk}, \mathsf{param})$*: This algorithm is executed by a group member* $\mathcal{U}$ *and takes as input his secret key* usk*, public key* upk*, a message* m*, the group public key* gpk *and the group parameters* param*. It outputs a valid group signature* σ*.*

$0/1 \leftarrow \mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param})$*: This is a deterministic and public algorithm that verifies the validity of the signature* σ *for a message* m *with the group public key* gpk *and the group parameters* param*. It outputs* 1 *if* σ *is valid,* 0 *otherwise.*

$\mathcal{U}/\perp \leftarrow \mathsf{GS.Open}(m, \sigma, \mathsf{gsk})$*: This algorithm is executed by the central authority* $\mathcal{M}$ *and takes as inputs, a message* m*, a signature* σ *and the group secret key* gsk *and it outputs all identities* $\mathcal{U}$*s that generated* σ *or* $\perp$ *if no member could be assigned to* σ*.*

$(\mathsf{gpk}, \mathsf{gsk}, \boldsymbol{members}) \leftarrow \mathsf{GS.Revoke}(\mathcal{U}, \mathsf{gsk}, \mathsf{gpk}, \boldsymbol{members}, \mathsf{param})$*: This algorithm revokes the ability of the user* $\mathcal{U}$ *to generate valid signature and is executed by the central authority* $\mathcal{M}$*. It takes as input the member* $\mathcal{U}$ *to be revoked, the group secret key* gsk*, the group public key* gpk*, the list of group members' identities* ***members*** *and the group parameters* param*. It outputs an updated group public key* gpk *(including an updated list of all revoked members* $\mathcal{R}$*), master secret keys* gsk *and the updated list of group members' identities* ***members****.*

2.9.1 Security of Group signature

Security Oracles

A secure group signature achieves the security properties of Correctness, Non-frameability, Anonymity, Traceability, Tracing-Soundness and Revocability. To prove these requirements, we use a set of games, in which an adversary $\mathcal{A}$ can access oracles.

There are also a number of different global sets: $\mathcal{S}$ is list of signatures obtained from the $Sign$ oracle, $\mathcal{H}$ is the the list of honest group members, $\mathcal{CO}$ is the list of corrupted members and $\mathcal{CH}$ is the list of outputs of the challenge oracle $Chal_b$.

- $\sigma \leftarrow Sign(\mathcal{U}, m)$: It returns a signature for the message m by the group member $\mathcal{U}$ ($\sigma \leftarrow \mathsf{GS.Sign}(m, \mathsf{usk}, \mathsf{upk}, \mathsf{gpk}, \mathsf{param})$) if and only if $\mathcal{U} \in \mathcal{H}$ and $\mathcal{U} \notin \mathcal{CO}$. All queries performed with this oracle are inserted in a set $\mathcal{S}$ ($\mathcal{S} \leftarrow \mathcal{S} \bigcup \{(m, \sigma, \mathcal{U})\}$).
- $\mathcal{U} \leftarrow Open(m, \sigma)$: It returns the identity of the generator of the signature σ for the message m. More formally, it returns $\mathsf{GS.Open}(\sigma, \mathsf{gpk}, \mathsf{gsk})$ if and only if $\mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param}) = 1$ and $(m, \sigma) \notin \mathcal{CH}$ or, in other words, if the signature is valid and was not obtained from the $Chal_b$ oracle.
- $\mathsf{usk} \leftarrow RevealM(\mathcal{U})$: It returns the personal secret key usk of an honest group member $\mathcal{U}$. $\mathcal{U}$ is added to the list of corrupted members $\mathcal{CO}$, $\mathcal{CO} \leftarrow \mathcal{CO} \bigcup \{\mathcal{U}\}$
- $\mathsf{upk} \leftarrow AddM(\mathcal{U})$: It adds a new honest member $\mathcal{U}$ to the group by generating its public and secret keys $(\mathsf{upk}, \mathsf{usk})$ and initiates the joining process with $\mathcal{M}$. It returns the public key upk and updates the set of honest members: $\mathcal{H} \leftarrow \mathcal{H} \bigcup \{\mathcal{U}\}$
- $\mathsf{usk} \leftarrow CorruptM(\mathcal{U})$: It creates a new corrupted member $\mathcal{U}$ and initiates the joining process with $\mathcal{M}$. $\mathcal{U}$ is added to the list of corrupted members $\mathcal{CO}$, $\mathcal{CO} \leftarrow \mathcal{CO} \bigcup \{\mathcal{U}\}$.
- $\sigma \leftarrow Chal_b(\mathcal{U}^0; \mathcal{U}^1, m)$: It returns a signature generated by the member $\mathcal{U}^b$ only if both $\mathcal{U}^0$ and $\mathcal{U}^1$ are honest members ($\mathcal{U}^0 \in \mathcal{H}$ and $\mathcal{U}^0 \notin \mathcal{CO}$ and $\mathcal{U}^1 \in \mathcal{H}$ and $\mathcal{U}^1 \notin \mathcal{CO}$). This oracle can be called only once in the anonymity experiments (see Figure 5.1). The message and the generated signature are added to the set $\mathcal{CH}$, $\mathcal{CH} \leftarrow \mathcal{CH} \bigcup \{(m, \sigma)\}$.

Security definitions

Definition 2.9.2 (Correctness)**.** *A* GS *achieves correctness if and only if for a security parameter λ the advantage $\mathbf{Adv}^{Corr}_{\mathcal{A}}$ of a PPT adversary $\mathcal{A}$ satisfies*

$$\mathbf{Adv}^{Corr}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{Corr}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{2.11}$$

where the correctness experiment $\boldsymbol{Exp}^{Corr}_{\mathcal{A},\mathsf{GS}}(\lambda)$ is defined in Figure 2.3.

Definition 2.9.3 (Non-frameability)**.** *A* GS *provides non-frameability if a coalition of group members and/or outsiders, is not able to forge a valid signature σ' which can be linked to an honest member $\mathcal{U}$ outside the coalition. A* GS *achieves non-frameability if and only if for a security parameter λ the advantage $\mathbf{Adv}^{Frame}_{\mathcal{A}}(\lambda)$ of a PPT adversary $\mathcal{A}$ satisfies*

$$\mathbf{Adv}^{Frame}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{Fame}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{2.12}$$

where the non-frameability experiment $\boldsymbol{Exp}^{Frame}_{\mathcal{A},\mathsf{GS}}(\lambda)$ is described in Figure 2.3.

Definition 2.9.4 (Anonymity)**.** *A* GS *provides anonymity if and only if a signature σ does not leak any information on the identity of the signer. A* GS *achieves anonymity if and only if for a security parameter λ the advantage $\mathbf{Adv}^{Anon}_{\mathcal{A}}$ of a PPT adversary $\mathcal{A}$ satisfies*

$$\begin{aligned}\mathbf{Adv}^{Anon}_{\mathcal{A}}(\lambda) = |\Pr[\boldsymbol{Exp}^{Anon-0}_{\mathcal{A},\mathsf{GS}}(\lambda) = 0] - \Pr[\boldsymbol{Exp}^{Anon-1}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1]| \\ < \mathsf{negl}(\lambda),\end{aligned} \tag{2.13}$$

where the anonymity experiment $\boldsymbol{Exp}^{Anon-b}_{\mathcal{A},\mathsf{GS}}(\lambda)$ *is presented in Figure 2.3.*

Definition 2.9.5 (Traceability)**.** *A* GS *achieves traceability if and only if it is computationally infeasible for a PPT adversary to generate a valid signature* σ *which cannot be linked with a group member (i.e* $\perp \leftarrow \mathsf{GS.Open}(m, \sigma, \mathsf{gsk})$*). A* GS *achieves traceability if and only if for a security parameter* λ *the advantage* $\mathbf{Adv}^{Trace}_{\mathcal{A}}(\lambda)$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}^{Trace}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{Trace}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{2.14}$$

where the traceability experiment $\boldsymbol{Exp}^{Trace}_{\mathcal{A},\mathsf{GS}}(\lambda)$ *is illustrated in Figure 2.3.*

Definition 2.9.6 (Tracing Soundness)**.** *A* GS *achieves tracing soundness if and only if it is computationally infeasible for a PPT adversary to generate a valid signature* σ *which can be traced to two different group members (* $\mathcal{U}^0 \leftarrow \mathsf{GS.Open}(m, \sigma, \mathsf{gsk})$ *and* $\mathcal{U}^1 \leftarrow \mathsf{GS.Open}(m, \sigma, \mathsf{gsk})$ *with* $\mathcal{U}^0 \neq \mathcal{U}^1$*). A* GS *achieves tracing soundness if and only if for a security parameter* λ *the advantage* $\mathbf{Adv}^{Trace\text{-}Sound}_{\mathcal{A}}(\lambda)$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}^{Trace\text{-}Sound}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{Trace\text{-}Sound}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{2.15}$$

where the tracing soundness experiment $\boldsymbol{Exp}^{Trace\text{-}Sound}_{\mathcal{A},\mathsf{GS}}(\lambda)$ *is illustrated in Figure 2.3.*

Definition 2.9.7 (Revocability)**.** *A* GS *achieves revocability if and only if it is computationally infeasible for a adversary to generate a valid signature* σ *on the behalf of a revoked member. A* GS *achieves revocability if and only if for a security parameter* λ *the advantage* $\mathbf{Adv}^{Revoke}_{\mathcal{A}}(\lambda)$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}^{Revoke}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{Revoke}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{2.16}$$

where the revocability experiment $\boldsymbol{Exp}^{Revoke}_{\mathcal{A},\mathsf{GS}}(\lambda)$ *is illustrated in Figure 2.3.*

2.10 Ring signature

Similar to group signature, Ring signature (RS) allows a member of a group, or in this case, a ring, to sign a digital message anonymously. The main difference with group signature is that there is no central authority in other words, no manager is needed, and a ring member can **spontaneously** sign a message by using the public key of ring members. We will now provide the formal definition of ring signature, which is ensued from the works of Bender et al. [BKM06] and Malavolta et al. [MS17].

Definition 2.10.1 (Ring signature)**.** *A ring signature is defined by a tuple of PPT algorithms* $(\mathsf{RS.KeyGen}, \mathsf{RS.Sign}, \mathsf{RS.Verify})$

- $(\mathsf{RS.pk}, \mathsf{RS.sk}) \leftarrow \mathsf{RS.KeyGen}(1^\lambda)$*: This algorithm takes input the security parameter* λ *and outputs the public key* RS.pk *and the secret key* RS.sk*.*
- $\mathsf{RS}.\sigma \leftarrow \mathsf{RS.Sign}(m, \mathsf{RS.sk}, \mathsf{L})$*: This algorithm takes as input the message* m*, the secret key of the signer* RS.sk *and the ring* L *of public key, which includes the public key* RS.pk *of the signer. This outputs a ring signature* $\mathsf{RS}.\sigma$*.*
- $0/1 \leftarrow \mathsf{RS.Verify}(m, \mathsf{RS}.\sigma, \mathsf{L})$*: This deterministic algorithm takes as input the message* m*, the ring signature* $\mathsf{RS}.\sigma$ *and the ring* L *of public key. This outputs* 1 *if the signature is correct and* 0 *otherwise.*

$\underline{\mathbf{Exp}^{Corr}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{GS.Setup}(1^\lambda)$
$(\mathsf{id}_\mathcal{U}, m) \leftarrow \mathcal{A}^{AddM}(\mathsf{gpk}, \mathbf{members})$
if $\mathcal{A}$'s outputs are not valid
  then return 0
$\sigma \leftarrow \mathsf{GS.Sign}(m, \mathsf{usk}, \mathsf{upk}, \mathsf{gpk}, \mathsf{param})$
if $\mathsf{id}_\mathcal{U} \notin \mathbf{members}$ or $\mathsf{upk} \in \mathcal{R}$
  then return 1
if $\mathsf{GS.Verify}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{param}) = 0$
  then return 1
then return 0

$\underline{\mathbf{Exp}^{Frame}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{GS.Setup}(1^\lambda)$
$(\mathsf{id}_\mathcal{U}, m, \sigma) \leftarrow$
$\mathcal{A}\left\{\begin{array}{cc} AddM & CorruptM \\ Sign & RevealM \end{array}\right\}(play : \mathsf{gpk})$
if $\mathcal{A}$'s outputs are not valid
  then return 0
if $\mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param}) = 0$ or $(m, \mathsf{id}_\mathcal{U}) \in \mathcal{Q}$
  then return 0
$\mathsf{id}_\mathcal{U} \leftarrow \mathsf{GS.Open}(m, \sigma, \mathsf{gsk})$
if $\mathsf{id}_\mathcal{U} \notin \mathcal{R}$
  then return 1
return 0

$\underline{\mathbf{Exp}^{Anon-b}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{GS.Setup}(1^\lambda)$
$d \leftarrow \mathcal{A}\left\{\begin{array}{cc} AddM & Chall_b \\ RevealM & Open \\ CorruptM & Sign \end{array}\right\}(play : \mathsf{gpk}, \mathbf{members})$
Return d

$\underline{\mathbf{Exp}^{Trace}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{GS.Setup}(1^\lambda)$
$(m, \sigma, \mathsf{gpk}, \mathbf{members}) \leftarrow$
$\mathcal{A}\left\{\begin{array}{cc} AddM & Sign \\ CorruptM & RevealM \end{array}\right\}(play : \mathsf{gpk}, \mathbf{members})$
if $\mathcal{A}$'s outputs are not valid
  then return 0
if $\mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param}) = 0$
  then return 0
$\mathsf{id}_\mathcal{U} \leftarrow \mathsf{GS.Open}(m, \sigma, \mathsf{gsk})$
if $\mathsf{id}_\mathcal{U} \notin \mathbf{members}$ or $\mathsf{id}_\mathcal{U} = \perp$
  then return 1

$\underline{\mathbf{Exp}^{Trace\text{-}Sound}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{GS.Setup}(1^\lambda)$:
$\emptyset \leftarrow \mathcal{Q}$
$(m, \sigma, \mathsf{gpk}, \{\mathsf{id}^i_\mathcal{U}\}^1_{i=0}) \leftarrow \mathcal{A}^{CorruptM}(play : \mathsf{gpk}, \mathbf{members})$
if $\mathcal{A}$'s outputs are not valid
  then return 0
if $\mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param}) = 0$
  then return 0
if $\mathsf{id}^0_\mathcal{U} = \mathsf{id}^1_\mathcal{U}$ or $\mathsf{id}^0_\mathcal{U} = \mathsf{id}^1_\mathcal{U} = \perp$
  then return 1
return 0

$\underline{\mathbf{Exp}^{Revoke}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{GS.Setup}(1^\lambda)$:
$(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathcal{A}^{CorruptM}(play : \mathsf{gpk}, \mathbf{members})$
if $\mathcal{A}$'s outputs are not valid
  then return 0
$\sigma \leftarrow GS.Sign(m, \mathsf{usk}, \mathsf{upk}, \mathsf{gpk}, \mathsf{param})$ **if** $\mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param}) = 1$
  then return 1
Return 0

FIGURE 2.3: Security Experiments

2.10.1 Security of ring signature

According to [BKM06], ring signature achieve three main security properties. The correctness ensures that if a ring signature has been correctly generated, then the verification will pass with a probability equal to one. The unforgeability ensures that it is infeasible to generate a valid signature for a user not belonging to the ring. Finally, the anonymity ensures that guessing the identity of the signer in a ring of N members is close to $1/N$. The formal definition is presented below.

Definition 2.10.2 (Correctness)**.** *A* RS *achieves correctness if and only if* $\Pr[\mathsf{Verify}(m, \mathsf{L}, \mathsf{Sign}(m, \mathsf{RS.skL}))] = 1$ *for any* $(\mathsf{RS.pk}, \mathsf{RS.sk}) \leftarrow \mathsf{RS.KeyGen}(1^\lambda)$.

Definition 2.10.3 (Unforgeability)**.** *A* RS *achieves unforgeability if it is infeasible for a PPT adversary* $\mathcal{A}$ *to generate a valid signature* $\mathsf{RS}.\sigma$ *from the identities of the ring. A* RS *achieves unforgeability if and only if for a security parameter* λ *and a polynomial function* f*, the advantage* $\mathsf{Adv}_{\mathcal{A}}^{Forge}$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}_{\mathcal{A}}^{Forge}(\lambda) = \Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{RS}}^{Forge}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{2.17}$$

where the unforgeability experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{RS}}^{Forge}(\lambda)$ *is presented in Figure 2.4.*

Definition 2.10.4 (Anonymity)**.** *An* RS *for a ring* L*, a message* m *and a signature* $\sigma = \mathsf{RS.Sign}(m, \mathsf{RS.sk}, \mathsf{L})$*, achieves anonymity if and only if for a security parameter* λ *and a polynomial function* $\{$*, the advantage* $\mathsf{Adv}_{\mathcal{A}}^{Anon}$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}_{\mathcal{A}}^{Anon}(\lambda) = |\Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{RS}}^{Anon}(\lambda) = 1] - 1/N| < \mathsf{negl}(\lambda), \tag{2.18}$$

where the anonymity experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{RS}}^{Anon}(\lambda)$ *is defined in Figure 2.5 and* N *is number of ring members.*

$\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{RS}}(\lambda)$

Setup:
This game is executed by the challenger $\mathcal{CH}$ who initializes the sets $\mathcal{Q}, \mathcal{S}$: $\mathcal{Q}, \mathcal{S} \leftarrow \emptyset$. The $\mathcal{Q}$ stores the corruption queries and $\mathcal{S}$ stores the signing queries
$\mathcal{CH}$ runs $(\mathsf{RS.pk}_i, \mathsf{RS.sk}_i) \leftarrow \mathsf{RS.KeyGen}(1^\lambda, r_i)$ for $1 \leq i \leq f(\lambda)$ and provides the public key $\{\mathsf{RS.pk}_i\}_{i=1}^{f(\lambda)}$ to the adversary $\mathcal{A}$. **Query**:
$\mathsf{RS.sk} \leftarrow \mathcal{CO}(\mathsf{RS.pk})$: This oracle corrupts a user. The adversary $\mathcal{A}$ sends the user's public key $\mathsf{RS.pk}$ to the challenger $\mathcal{CH}$ which returns the corresponding secret key $\mathsf{RS.sk}$ and sends $\mathsf{RS.sk}$ to $\mathcal{A}$. $\mathsf{RS.pk}$ is added to the set $\mathcal{Q}$.
$\sigma \leftarrow \mathcal{SO}(m, \mathsf{L}, \mathsf{RS.pk})$: This signing oracle starts with $\mathcal{A}$ sending to the challenger $\mathcal{CH}$ a list of user's identities L, message m and the identity of the signer $\mathsf{RS.pk} \in \mathsf{L}$ and $\mathcal{CH}$ returns a valid signature σ for message m generated by the user associated with the public key $\mathsf{RS.pk}$ ($\sigma \leftarrow \mathsf{RS.Sign}(m, \mathsf{L}, \mathsf{RS.sk})$). The tuple (m, L) is added to the set $\mathcal{S}$.
Forgery:
$\mathcal{A}$ forges $(m', \mathsf{L}', \sigma')$.
if $1 \leftarrow \mathsf{RS.Verify}(m', \mathsf{L}', \sigma') \wedge ((m', \mathsf{L}') \notin \mathcal{S}) \wedge (\mathsf{RS.pk} \notin \mathcal{Q}$ for all $\mathsf{RS.pk} \in \mathsf{L}')$
then return 1
return 0

FIGURE 2.4: RS Unforgeability Game.

$\mathsf{Exp}^{Anon}_{\mathcal{A},\mathsf{RS}}(\lambda)$

Setup:
The challenger $\mathcal{CH}$ executes $(\mathsf{RS.pk}_i, \mathsf{RS.sk}_i) \leftarrow \mathsf{RS.KeyGen}(1^\lambda, r_i)$ for $1 \leq i \leq f(\lambda)$ and provides the public key $\{\mathsf{RS.pk}_i\}_{i=1}^{f(\lambda)}$ to the adversary $\mathcal{A}$.
Query:
$\sigma \leftarrow \mathcal{SO}(m, \mathsf{L}, \mathsf{RS.pk})$: This signing oracle starts with $\mathcal{A}$ sending to the challenger $\mathcal{CH}$ a list of user's identities L, message m and the identity of the signer $\mathsf{RS.pk} \in \mathsf{L}$ and $\mathcal{CH}$ returns a valid signature σ for message m generated by the user associated with the public key $\mathsf{RS.pk}$ ($\sigma \leftarrow \mathsf{RS.Sign}(m, \mathsf{L}, \mathsf{RS.sk})$). **Challenge**:
$\sigma \leftarrow \mathcal{CHO}(m, \mathsf{L})$: This interactive oracle is initiated by the adversary $\mathcal{A}$, who sends a message m to be signed and a list of identities L to the challenger $\mathcal{CH}$. $\mathcal{CH}$ randomly picks $\mathsf{RS.pk} \in \mathsf{L}$ and generates a valid signature σ for the selected $\mathsf{RS.pk}$ ($\sigma \leftarrow \mathsf{RS.Sign}(m, \mathsf{RS.sk}, \mathsf{L})$). $\mathcal{CH}$ sends σ to $\mathcal{A}$.
Output:
$\mathcal{A}$ chooses $\mathsf{RS.pk}^*$ based on the received σ.
if $\mathsf{RS.pk}^* = \mathsf{RS.pk}$
then return 1
return 0

FIGURE 2.5: RS Anonymity Game.

Chapter 3

DGM: A Dynamic & Revocable Group Merkle Signature

Group signatures are considered as one of the most prominent cryptographic primitives to ensure privacy. In essence, group signatures ensure the authenticity of messages while the author of the message remains anonymous.

This chapter presents a group signature named DGM which aims to improve the applicability of the state-of-the-art at the time of the design as presented in Table 3.1 and 3.5. In this chapter, we propose a dynamic post-quantum group signature that extends the static G-Merkle group signature [EBM18] (PQCRYPTO 2018). In particular, our dynamic G-Merkle (DGM) allows new users to join the group at any time. Similar to the G-Merkle scheme, our DGM only involves symmetric primitives and makes use of a One-Time Signature scheme (OTS). Each member of the group receives a certain amount of OTS key pairs and can ask the Manager $\mathcal{M}$ for more if needed. Our DGM also provides an innovative way of signing revocation by employing Symmetric Puncturable Encryption (SPE) appeared in ACM CCS 2018 [SYL+18]. DGM provides a significantly smaller signature size than other group signatures based on symmetric primitives and also reduces the influence of the number of group members on the signature size and on the limitations of the application of G-Merkle.

One of the main issues of the previous post-quantum symmetric solutions is the size of the generated signatures. Since the standard symmetric cryptographic algorithms ,for example the block cipher AES or the hash function SHA, are efficient and short, our DGM design and algorithms are also simple and efficient, the signature sizes are short compared to other group signatures based on symmetric primitives.

This chapter includes content of our paper "DGM: A Dynamic & Revocable Group Merkle Signature" [BLS+19] published at ESORICS 2019 [ESOa].

3.0.1 Detailed contributions

In this chapter, we introduce a dynamic post-quantum group signature using only symmetric primitives, in particular cryptographic hash functions and block ciphers. Our starting point is the static group signature G-Merkle, designed by El Bansarkhani and Misoczki [EBM18]. Our specific contributions are listed below:

- **Introducing DGM**: We propose a dynamic G-Merkle (DGM) group signature and solve the problem of G-Merkle to deal with large group by using multiple parallel Merkle trees (Sections 3.3 and 3.4) (See Figure3.3). Moreover, our extension assures that each correctly generated signature will go through the verification process. This is in contrast to dynamic group signature of [BEF18], in which a signature will be rejected if a new member has joined the group after the last update of parameters with the manager (Section 3.3).

TABLE 3.1: Comparison of different group signatures and their functionalities for $N = 2^l$ being the number of group members, B be the number of OTS keys per member, λ be the post-quantum security parameter, and rvk be the number of revoked OTS keys.

Group Signature	Sig. Size	Group PK size	Dynamicity	Revocation
[LLLS13]	$\mathcal{O}(\lambda^2 \cdot \log N)$	$\mathcal{O}(\lambda^2 \cdot \log N)$	Yes	No
[GKV10]	$\mathcal{O}(\lambda^2 \cdot N)$	$\mathcal{O}(\lambda^2 \cdot N)$	No	No
[KKW18]	$\mathcal{O}(\lambda \cdot \log N)$	$\mathcal{O}(\lambda)$	No	No
[BEF18]	$\mathcal{O}(\lambda \cdot \log N)$	$\mathcal{O}(\lambda)$	Yes	Yes
[EBM18]	$\mathcal{O}(\log(N \cdot B))$	$\mathcal{O}(\lambda)$	No	No
[LNWX18]	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	Yes	No
[DPLS18]	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	Yes	No
[EZS+19]	$\mathcal{O}(\lambda \cdot \log N)$	$\mathcal{O}(\lambda \cdot N)$	Yes	No
DGM	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda \cdot \mathsf{rvk})$	Yes	Yes

- **Competitive signature size**: Our DGM signatures are significantly shorter (in term of size) than other dynamic group signatures based on symmetric primitives (see Section 3.4). Moreover, the influence of the number of group members on the size (length) of the signature is diminished comparing to G-Merkle. All these have also been verified by numerical results derived from group signature applications (See Table 3.1, 3.5 and 3.3).

- **Innovative revocation process**: We also propose an innovative way of revoking the ability to sign of a misbehaved group member with the use of symmetric puncturable encryption (See Section 3.3.1).

Table 3.1 and 3.5 compare DGM with other post-quantum group signature schemes and their functionalities. It shows that more than providing dynamicity and revocation, DGM's main advantage is to remove the dependency of the signature size on the number of group members N. Indeed, the size of the group signature is set during the setup phase and will remain unchanged afterward.

3.0.2 Related works

Chaum et al. [CVH91] were the firsts to theorize the concept of anonymously signing on behalf of a group. However, the most commonly used definition for dynamic group signature is the one presented by Bellare et al. in [BSZ05] formalize the definition. In the field of post-quantum security, the first hash-based signature scheme designed are one-time signature schemes (OTS) presented in [Lam79], or more recently in [Hül13]. The work of [Mer89] transformed OTS to a multi-signature scheme thanks to a Merkle tree. However, the weakness with the Merkle signature is that the number of possible signatures is fixed after setup phase. Therefore, Chalkias et al. [CBH+18] proposed to add a fall back mechanism along with the Merkle tree to add more flexibility. Even if there exist hash-based signatures, when designing post-quantum group signatures, most researchers choose lattice-based cryptography rather than using symmetric primitives. The works of Libert et al. [LLM+16] [LLNW16], Gordon et al. [GKV10], and Ling et al [LNW15] [LNWX17] demonstrate this trend.
Nevertheless, the construction of the Zero-Knowledge proof [GMW91] ZKBoo [GMO16], constructed only with symmetric primitives, shows that symmetric primitive can also be used to design post-quantum signature and group signature. ZKBoo uses the concept presented by Ishai et al., namely the "MPC in the head" [IKOS09]. From this, one can achieve new optimized Zero-Knowledge proofs such as [CDG+17]

[AHIV17] or [KKW18]. 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++ [CDG+17], an optimization of ZKBoo. Alongside ZKB++, either Fiat-Shamir Transform [FS86] or Unruh Transform [Unr15] were employed to construct an Non-Interactive Zero-Knowledge proof (NIZK) [BG92].
A dynamic group signature constructed with symmetric primitives is proposed first by Boneh et al. [BEF18]. Recently, a new draft of post-quantum group signature has been proposed by Katz et al. [KKW18], with an optimized NIZK based on symmetric primitives and "MPC in the head". El Bansarkhani and Misoczki. [EBM18] presented G-Merkle, a static group signature, which is basically a modification of the Merkle signature.

3.0.3 Outline of chapter 3

Chapter 3 starts with Section 3.1 which names the cryptographic primitives used to design DGM and defines the ones only used in this chapter 2. Section 3.2 introduces the static group signature G-Merkle to understand the construction of DGM. Section 3.3 presents the detailed construction of our group signature DGM and its performance is discussed in Section 3.4. This chapter concludes with the addition of DGM to Table 3.5 in order to summarized its impact on the current state-of-the-art.

3.1 Additional preliminaries

This section aims to introduce all of the theoretical concepts on which our DGM is constructed. Our scheme relies on the idea of Merkle tree [Mer89] and is based on two main symmetric primitives: cryptographic hash function H (see Definition 2.2.1) and symmetric encryption SE composed by a tuple of polynomial-time algorithms $\mathsf{SE} = (\mathsf{SE.KeyGen}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ (See Definition 2.3.1 for more details). We also use a One-Time Signature scheme OTS, that can be easily constructed by symmetric primitives, only is defined by a tuple of polynomial-time algorithms $\mathsf{OTS} = (\mathsf{OTS.KeyGen}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$ (See Definition 2.5.1 for details). All these cryptographic primitives are presented in Chapter 2.

DGM is also constructed from a primitive named symmetric puncturable encryption (SPE). Therefore, this section formally defines puncturable pseudorandom function and their applications to puncturable encryption scheme.

3.1.1 Puncturable pseudorandom function

We first introduce the syntax of a keyed puncturable pseudorandom function (PPRF) [HKW15] [SYL+18]

Definition 3.1.1 (Puncturable pseudorandom function)**.** $\mathsf{PRF} : K \times X \leftarrow Y$*, where* K *is the key space,* X *is the input space, and* Y *is to the output space. This function takes an input* $x \in X$ *and a key* $k \in K$ *and outputs* $y = \mathsf{PRF}(k, x)$*. Furthermore, it has the following two functions*

- $k_x \leftarrow \mathsf{F.Punc}(k, x)$*: takes as input a PRF key* $k \in K$ *and an element* $x \in X$*, and outputs a punctured secret key* $k_x \in K_p$ *and*

- $y \leftarrow \mathsf{F.Eval}(k_x, x')$: *takes as input a punctured key* $k_x \in K_p$ *and an element* $x' \in X$*, and outputs an element* y*, where*

$$\mathsf{F.Eval}(k_{x'}, x) = \begin{cases} \mathsf{PRF}(k, x) & \text{if } x \neq x' \\ \perp & \text{else.} \end{cases} \tag{3.1}$$

3.1.2 Symmetric puncturable encryption

In this chapter, we use another symmetric primitive called symmetric puncturable encryption (SPE) [SYL+18]. This was used in the context of searchable encryption. The term “puncture" is used because the secret key has been revoked the ability to decrypt some ciphertext.

Definition 3.1.2 (symmetric puncturable encryption (SPE))**.** *A* d*-puncturable SPE with message space* $\mathcal{PM}$ *and tag space* $\mathcal{T}$*, is defined by the following four polynomial time algorithms:*

- $\mathsf{SPE.msk} \leftarrow \mathsf{SPE.KeyGen}(1^\lambda, d)$: *take as inputs a security parameter* λ *and a positive integer* d*, which indicates the maximum number of allowed punctured tags. It outputs a random secret key* $\mathsf{SPE.sk}_0$ *and sets* $\mathsf{SPE.msk} = (\mathsf{SPE.sk}_0, d)$*,*
- $\mathsf{SPE.ct} \leftarrow \mathsf{SPE.Enc}(m, \mathsf{SPE.msk}, t)$: *the encryption algorithm taking* $\mathsf{SPE.msk}$*, a message* m*, and a tag* t *as inputs and outputs a ciphertext* $\mathsf{SPE.ct}$*,*
- $\mathsf{SPE.sk}_i \leftarrow \mathsf{Punc}(\mathsf{SPE.sk}_{i-1}, t')$: *the puncture algorithm, which takes as inputs* $\mathsf{SPE.sk}_{i-1}$ *and a new tag* t' *and outputs a new key* $\mathsf{SPE.sk}_i$*. The new* $\mathsf{SPE.sk}_i$ *can decrypt every ciphertext that* $\mathsf{SPE.sk}_{i-1}$ *can except the one encrypted with* t'*, and*
- $m/\perp \leftarrow \mathsf{SPE.Enc}(\mathsf{SPE.ct}, \mathsf{SPE.sk}_i, t)$: *a deterministic decryption algorithm, which takes as inputs a punctured key* $\mathsf{SPE.sk}_i$*, a ciphertext* $\mathsf{SPE.ct}$*, and a tag* t*, and finally outputs a plaintext* m*.*

A security model for SPE is formalized in [SYL+18], however, we only use the correctness of SPE in our setting with a security parameters λ, which is defined as:

$$\Pr[\mathsf{SPE.Dec}(\mathsf{SPE.Enc}(\mathsf{SPE.msk}, m, t), \mathsf{SPE.sk}_i, t) = m] = 1. \tag{3.2}$$

where $t \in T \setminus T_i$, where $T_i = \{t_1, t_2, \ldots, t_n\}$ is an arbitrary set of distinct tags punctured at $\mathsf{SPE.sk}_i$.

3.2 G-Merkle

G-Merkel [EBM18] is a post-quantum group signature constructed from symmetric primitives and based on the idea of the Merkle signature [Mer89].G-Merkle uses a hash function H (see Definition 2.2.1), an OTS scheme (see Definition 2.5.1), and a symmetric encryption scheme SE (see Definition 2.3.1). In the following, we give an overview of such a scheme.

3.2.1 G-Merkle overview

The manager $\mathcal{M}$ has the responsibility for a group of N users, and each user will be allowed to sign B messages. This means that each user possesses B OTS key pairs. Therefore, the group public key $\mathsf{GM.gpk}$ will be the Merkle root of the tree generated

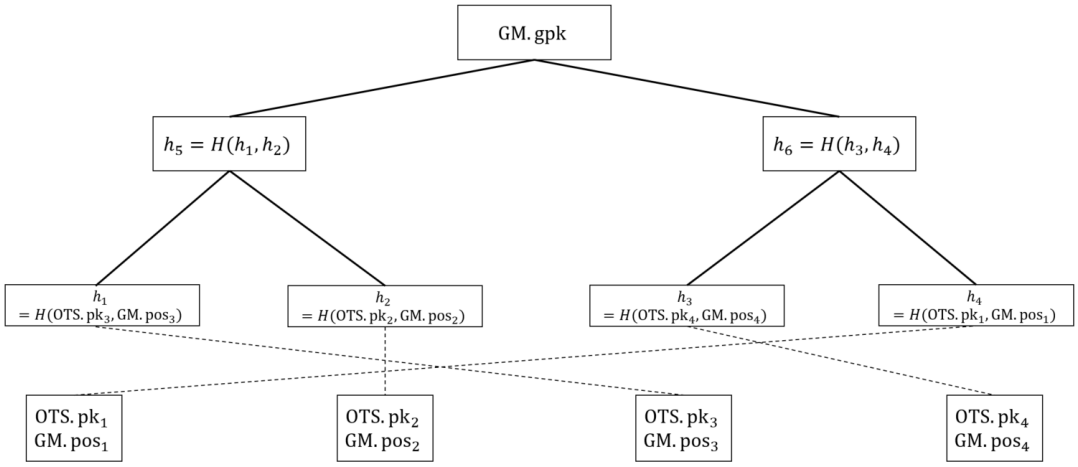


FIGURE 3.1: G-Merkle: Tree structure

over $B{\cdot}N$ leaves where each leaf is an OTS public key. Each leaf is labelled from 1 to $B{\cdot}N$ and the i-th member will receive the OTS keys corresponding to leaves $\{B\cdot(i-1)+1,\ldots,B\cdot i\}$. Similar to the traditional Merkle signature [Mer89], a signature is composed of an OTS signature and the path from $\mathsf{OTS.pk}$ to $\mathsf{GM.gpk}$. However, in order to ensure user anonymity, $\mathcal{M}$ "Shuffles" the set of leaves L. During the "Shuffle" process, the set L is composed of tuples $\{(\mathsf{OTS.pk}_1, \mathsf{GM.pos}_1), \ldots, (\mathsf{OTS.pk}_{B\cdot N}, \mathsf{GM.pos}_{B\cdot N})\}$, where $\mathsf{GM.pos}_i = \mathsf{SE.Enc}(i, \mathsf{GM.msk})$ and $\mathsf{GM.msk}$ is the $\mathcal{M}$'s secret key. The "Shuffle" procedure ends by ordering the set L according to $\mathsf{GM.pos}$. Another modification of the first layer of nodes is then built by not only including the leaves in the hashes, but also the encrypted values of the respective leaves, e.g. $\mathsf{H}(\mathsf{OTS.pk}_i, \mathsf{GM.pos}_i)$ (see Figure 3.1). Finally, $\mathcal{M}$ generates the Merkle tree which gives $\mathsf{GM.gpk}$. The "Shuffle" process prevents an adversary from identifying sub-tree and hence guessing the identity of the signer. Figure 3.1 shows the G-Merkle tree structure for $N = 2$ and $B = 2$, where $\mathsf{GM.pos}_3 \leq \mathsf{GM.pos}_2 \leq \mathsf{GM.pos}_4 \leq \mathsf{GM.pos}_1$.

3.2.2 G-Merkle limitations

The lack of flexibility of G-Merkle appears to be its main limitation. Undeniably, G-Merkle is reserved for static groups because the maximum number of available OTS key pairs will be fixed, and so will the number of members after the setup phase. We address this issue by introducing DGM as follows. Another issue of G-Merkle, already mentioned in [BEF18], is the lack of efficiency in the setup process for large groups, which limits its application in practical scenarios. We further demonstrate that our DGM solves this lack of efficiency in G-Merkle with respect to time taken to perform the setup phase (see Figure 3.3).

3.3 DGM

This section aims to present our group signature and starts with an overview of DGM by highlighting the differences with G-Merkle. Then, we will present the construction of DGM explicitly. DGM is constructed with a hash-based OTS scheme OTS, a symmetric encryption SE, a symmetric puncturable encryption SPE, and a hash function H. It follows the definition of dynamic group signature presented in Chapter 2.9.1.

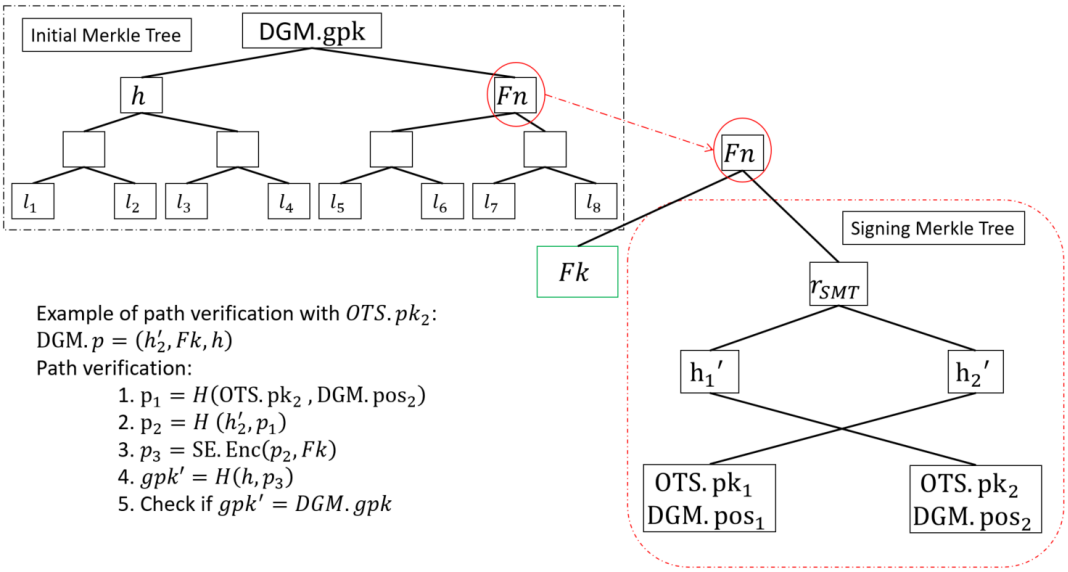


FIGURE 3.2: DGM Merkle Trees example

3.3.1 DGM overview

DGM is a dynamic extension of G-Merkle by allowing new members to join the group after the setup phase. Contrary to G-Merkle, DGM members are not limited to have only B OTS key pairs and therefore they are not limited to generate only B signatures. Indeed, when a member is running out of OTS key pairs, he can request new ones from manager $\mathcal{M}$.

Two types of Merkle tree

To achieve dynamicity, we change the tree structure (See Figure 3.2). Indeed, DGM distinguishes two types of Merkle trees. There is a unique Merkle tree named *Initial Merkle tree* (IMT), which is generated during the setup phase. The IMT is generated on randomly choosing leaves and its Merkle root $\mathsf{DGM.gpk}$, which is an element of the group public key gpk. An example of an IMT is shown in Figure 3.2, where the elements l_1 to l_8 are randomly chosen strings. The second type of Merkle trees are the *Signing Merkle Tree* (SMT). Those trees are parallel trees to IMT and linked with an internal node of IMT called "fallback node" and denoted with Fn. The link between a SMT and the IMT is created with Fn and a "fallback key" Fk. After the generation of a SMT, Fk is computed as $\mathsf{SE.Dec}(Fn, r_{SMT}) = Fk$, where r_{SMT} is the root of the SMT. Therefore, for the path verification from an OTS key to $\mathsf{DGM.gpk}$, $\mathsf{SE.Enc}(Fk, r_{SMT}) = Fn$ should be computed. SMTs are generated when a group member requests new OTS key pairs and are constructed in a similar manner to G-Merkle tree. Their leaves are kept secret to hide the time of the generation. Each of its leaves have an index $\mathsf{DGM.i} = (t, l)$, where t is the SMT number and l is the leaf number. DGM "Shuffles" the leaves with a symmetric encryption scheme. All leaves of the SMT are sorted by $\mathsf{DGM.pos}$, where $\mathsf{DGM.pos} = \mathsf{SE.Enc}(\mathsf{DGM.i}, \mathcal{M}.\mathsf{sk})$. In Figure 3.2, we have $\mathsf{DGM.pos}_2 < \mathsf{DGM.pos}_1$. $\mathcal{M}.\mathsf{sk}$ is the manager secret key and is one element of the group secret key gsk.

Join and Request OTS keys

DGM separates the join process from the request of OTS keys. A prospective user needs to first join the group and then it will have the possibility to request OTS keys. The request for new OTS keys is a subroutine (See Algorithm 1, 2 and 3) of the signing algorithms which is executed when the signer does not have an available an OTS key pair. When $\mathcal{M}$ receives a request for new OTS keys, he selects randomly an internal IMT node n_i. If n_i is a fallback node, then it takes the next available key pair of the corresponding SMT. Otherwise, he needs to generate a new SMT. All trees are generated by $\mathcal{M}$, who owns the group secret key gsk which is composed of all private lists and the manager secret key $\mathcal{M}.\mathsf{sk}$. When the SMT associated with n_i has used all its leafs, n_i is not considered as a fallback node. A new SMT will be generated when n_i will be selected.

The DGM signature σ

The format of σ is identical to G-Merkle. Therefore, $\sigma = (m, \mathsf{OTS}.\sigma, \mathsf{OTS.pk}, \mathsf{DGM.pos}, \mathsf{DGM.p})$, where $\mathsf{OTS}.\sigma$ is the OTS signature of the message m, which can be verified with $\mathsf{OTS.pk}$. $\mathsf{DGM.pos}$ is the corresponding position of $\mathsf{OTS.pk}$ in the tree and $\mathsf{DGM.p}$ is the path from the $\mathsf{OTS.pk}$ to $\mathsf{DGM.gpk}$. The only difference between σ and $\mathsf{GM}.\sigma$ is the fallback key Fk, which is an element of the path. In order to verify the path, the verifier needs to use $\mathsf{SE.Enc}$ instead of H and also verify the validity of Fk with $\mathcal{M}$, because SE does not provide collision resistance, as a hash function (see Definition 2.2.1). An example of path verification is illustrated in the Figure 3.2.

Revocation using SPE

DGM provides an innovative way of revoking the ability to a group member to sign. The $\mathsf{DGM.pos}$ is an element of the signature specific to a member allowing $\mathcal{M}$ to perform such a task. To revoke a member $\mathcal{U}$, $\mathcal{M}$ will first generate a new $\mathsf{DGM.rvk}_0 = \mathsf{SPE.msk}$, then he will puncture the key with every $\mathsf{DGM.pos}$ of all revoked members producing the punctured key $\mathsf{DGM.rvk}_1 = \mathsf{SPE.sk}$. During the verification, the verifier runs $\mathsf{SPE.Dec}(\mathsf{SPE.Enc}(m, \mathsf{DGM.rvk}_0, \mathsf{DGM.pos}), \mathsf{DGM.rvk}_1, \mathsf{DGM.pos})$ (line 1 of $\mathsf{DGM.Verify}$ in Algorithm 2) and if it outputs something rather than m, then it means that by the correctness of SPE (See Section 3.1.2), that $\mathsf{DGM.pos}$ has been punctured so the signer has been revoked. SPE offers a revocation alternative compared to the traditional use of lists. Previously, the only way to revoke members for group signature constructed with symmetric primitives was the use of lists. For example, Boneh et al. [BEF18] used two lists, one of revoked signature and another with the secret key of revoked members, which leads to their identities. In contrary to their work, DGM preserves anonymity of revoked members (See Section 3.3.3). Furthermore, it allows the verifier to work with only the signature and $\mathcal{M}$'s public parameter to perform and check the revocation.

3.3.2 Detailed DGM construction

Our group signature follows Definition 2.9.1 presented in Chapter 2 on pager 23. The main parameters are summarized in the Table 3.2. All algorithms are presented in Algorithm 1 and 2. The parts in Algorithm 1 and 2 highlighted in red are the ones which differ from G-Merkle [EBM18]. In the following, we describe each algorithm in more details:

- $(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{DGM.Setup}(1^\lambda)$: This algorithm is executed by $\mathcal{M}$ and takes as input the security parameter λ. It initialize the group parameters $\mathsf{param} = h_{\mathsf{IMT}}$. The algorithm generates $2^{h_{\mathsf{IMT}}}$ random leaves and then constructs IMT. Figure 3.2 shows an example with the parameter $h_{\mathsf{IMT}} = 3$. It initializes the group public key gpk which is composed of the Merkle root of the initial Merkle tree (IMT) DGM.gpk, which is a tree of height h_{IMT}, the revocation key DGM.rvk which is intially set to $(\perp, \perp)$, as no member has been revoked and *rvks* is initialized to 0. It also generates the group secret key gsk which is composed of the manager secret key $\mathcal{M}$.sk, which is a secret key of an symmetric encryption scheme, and of all secret lists (**OTS**, **FN**, **FK**, $priv\mathcal{L}$). This algorithm is detailed in Algorithm 1.

- $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{DGM.MKeyGen}(1^\lambda)$: This algorithm is executed by a prospective group member before initiating the joining process. This takes the security parameter λ and outputs the user's public key upk and the user's secret key usk. At the end of this procedure, the user' secret key usk is composed of a variable *state* initially set to 0. *state* will be updated after each signature generated by this user. This algorithm is detailed in Algorithm 2.

- $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}} \leftarrow \mathsf{DGM.Join}(\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}})$: This interactive protocol is taking place between $\mathcal{M}$ and a prospective member $\mathcal{U}$. $\mathcal{M}$ will add $\mathcal{U}$ to his both list **members** and $priv\mathcal{L}$, and $\mathcal{U}$ stores the challenges sent by $\mathcal{M}$. The user uses these to prove its own identity when requesting OTS keys. During this process, $\mathcal{U}$ will receive a challenge c which will be used for the request of new OTS key pairs. This algorithm is detailed in Algorithm 1.

- $\sigma \leftarrow \mathsf{DGM.Sign}(m, \mathsf{usk}, \mathsf{upk}, \mathsf{gpk}, \mathsf{param})$: The signing algorithm is similar as in G-Merkle [EBM18], it takes as input the message m and the secret key usk of the group member composed of a private list of OTS key pairs and an integer *state* which protects $\mathcal{U}$ to use twice the same OTS key pair. The difference with G-Merkle [EBM18] is the subroutine named DGM.OTSRequest (See Algorithm 3) and that is executed if $\mathcal{U}$ needs new OTS keys.

  - $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}} \leftarrow \mathsf{DGM.OTSRequest}(\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}})$: This is an interactive protocol that takes place over a secure channel between a group member $\mathcal{U}$ who desire B new OTS key pairs and $\mathcal{M}$. When receiving request for new OTS key pairs, $\mathcal{M}$ will call the procedure DGM.Update which will assign B new OTS key to $\mathcal{U}$. from already generated SMT or from newly generated SMT ones. The detail of DGM.Update procedure is presented in Algorithm 3, where the part written in blue is the "Shuffle" process similar to G-Merkle [EBM18] (See Section 3.2). At the end of the procedure, usk is updated by adding the new received OTS keys and the manager update is his list As an example, in Figure 3.2, if the node Fn is selected at step 2 and the first OTS key pair have already been assigned, the second OTS key pair will be assigned to the members, if he is not the second leaf of the tree because of the "Shuffle" process. If another node is selected, we need to generate a new SMT. This algorithm is detailed in Algorithm 3.

  This algorithm is detailed in Algorithm 1.

- $0/1 \leftarrow \mathsf{DGM.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param})$: This deterministic verification algorithm is very similar to the one from G-Merkle, but differs into two aspects. The

TABLE 3.2: DGM parameters summary

IMT	Initial Merkle tree
SMT_t	t-th Signing Merkle tree
$\mathcal{M}.\mathsf{sk}$	Manager's secret key
$\mathsf{DGM.rvk}$	The public revocation key, which is equal to ($\mathsf{DGM.rvk}_0 = \mathsf{SPE.msk}, \mathsf{DGM.rvk}_1 = \mathsf{SPE.sk}_{rvks}$)
$\mathsf{DGM.i}$	Tuple (t, i) denoting the i-th OTS key of the t-th SMT
$\mathsf{DGM.pos}_i$	This correspond to the encryption $\mathsf{SE.Enc}(\mathsf{DGM.i}, \mathcal{M}.\mathsf{sk})$ of the index of the OTS public key and it allows $\mathcal{M}$ to trace signatures
$priv\mathcal{L}$	Private list of members composed of all identities with their associated $\mathsf{DGM.pos}$
OTS	List of non-assigned OTS keys
FK	List of fallback keys
FN	List of fallback nodes
members	Public list of group members
h_{IMT}	The height of IMT
usk	The secret key of $\mathcal{U}$ including its OTS **keys** and a variable *state*, the element c used to request OTS key to $\mathcal{M}$.
$rvks$	Total number of revoked OTS keys.
$n_i.level$	Level of the fallback node n_i in IMT, for example the node FN in Figure 3.2 is $Fn.level = 1$ and $\mathsf{DGM.gpk}$ is at level 0
gpk	group public key composed of the IMT Merkle root $\mathsf{DGM.gpk}$, the revocation key $\mathsf{DGM.rvk}$, the total of revoked OTS keys $rvks$
gsk	group secret key composed of the manager secret key $\mathcal{M}.\mathsf{sk}$, the private lists **FK**,**FN**,**OTS**, $priv\mathcal{L}$

first difference is the path verification (See Figure 3.2). The second is the interaction between $\mathcal{M}$ and the verifier, to assure the validity of Fk in the path and the last one is the execution of $\mathsf{SPE.Enc}$ and $\mathsf{SPE.Dec}$ procedures to verify that the signer was not revoked. The $\mathsf{DGM.pos}$ of all revoked signatures have been used to generate a new punctured key such that $\mathsf{DGM.Enc}$ will not output a valid plaintext if $\mathsf{DGM.pos}$ has been punctured.This algorithm is detailed in Algorithm 2.

- $\mathcal{U}/\perp \leftarrow \mathsf{DGM.Open}(m, \sigma, \mathsf{gsk})$: This algorithm executed by $\mathcal{M}$ returns the identity of a group member, who provides σ. It decrypts the element $\mathsf{DGM.pos}$ from the signature and searches the corresponding $\mathsf{DGM.i}$ in the list $priv\mathcal{L}$. This algorithm is detailed in Algorithm 2.

- $(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}) \leftarrow \mathsf{DGM.Revoke}(\mathcal{U}, \mathsf{gsk}, \mathsf{gpk}, \mathbf{members}, \mathsf{param})$: This algorithm, executed by $\mathcal{M}$, revokes the ability of a malicious member to sign a message by puncturing (See Definition 3.1.2 on SPE) all $\mathsf{DGM.poss}$ associated with $\mathcal{U}$ and all previous revoked members. This algorithm is detailed in Algorithm 2.

Algorithm 1 DGM algorithms part 1

DGM.Setup

Input: λ

Output: gpk, gsk, **members**, param

1: $\mathcal{M}$.sk $\leftarrow$ SE.KeyGen(1^λ)
2: Initialize h_{IMT}
3: param $\leftarrow h_{\mathsf{IMT}}$
4: Initialize **FK**, **FN**, **members**, **OTS**, $privL \leftarrow \emptyset$
5: Generate of a set $2^{h_{\mathsf{IMT}}}$ (element l_1 to l_8 in Figure 3.2) of random leaves of size 2λ bits. Generate IMT where the Merkle root is the group public key DGM.gpk
6: $rvks \leftarrow 0$
7: DGM.rvk $\leftarrow (\perp, \perp)$ ▷ No member has been revoked at this point
8: gpk $\leftarrow$ (DGM.gpk, DGM.rvk, $rvks$)
9: gsk $\leftarrow$ ($\mathcal{M}$.sk, **FK**, **FN**, **OTS**, $privL$)
10: **return** (gpk, gsk, **members**, param)

DGM.MKeyGen

Input: λ

Output: upk, usk

1: Initialize usk, upk
2: $state \leftarrow 0$
3: usk $\leftarrow state$
4: **return** (upk, usk)

DGM.Join

Input: $\langle$usk$\rangle_{\mathcal{U}}$, $\langle$gpk, gsk, **members**$\rangle_{\mathcal{M}}$

Output: $\langle$usk, upk$\rangle_{\mathcal{U}}$, $\langle$gpk, gsk, **members**$\rangle_{\mathcal{M}}$

1: Parse usk $\rightarrow$ ($state$)
2: Parse gpk $\rightarrow$ (DGM.gpk, DGM.rvk, $rvks$)
3: Parse gsk $\rightarrow$ ($\mathcal{M}$.sk, **FK**, **FN**, **OTS**, $privL$)
4: $\mathcal{U}$ sends a request to join the group
5: $\mathcal{M}$ generates a challenge $c \in \{0,1\}^\lambda$ sends it to $\mathcal{U}$
6: $\mathcal{U}$: usk $\leftarrow (c, state)$
7: $\mathcal{M}$ adds $\mathcal{U}$ to the lists **members** and $privL$.
8: **return** $\langle$usk, upk$\rangle_{\mathcal{U}}$, $\langle$gpk, gsk, **members**$\rangle_{\mathcal{M}}$

DGM.Sign

Input: m, usk, upk, gpk, param

Output: σ

1: Parse usk $\rightarrow$ (c, $state$, **keys**)
2: Parse gpk $\rightarrow$ (DGM.gpk, DGM.rvk, $rvks$)
3: Parse DGM.rvk $\rightarrow$ (DGM.rvk$_0$, DGM.rvk$_1$)
4: Parse param $\rightarrow h_{\mathsf{IMT}}$
5: **if** $\mathcal{U}$ has not OTS keys available **then**
6: $(\langle$usk, upk$\rangle_{\mathcal{U}}$, $\langle$gpk, gsk$\rangle$, **members**$\rangle_{\mathcal{M}} \leftarrow$ DGM.OTSRequest($\langle$usk, upk$\rangle_{\mathcal{U}}$, $\langle$gpk, gsk, **members**$\rangle_{\mathcal{M}}$) ▷see Algorithm 3
7: **end if**
8: OTS.$\sigma \leftarrow$ OTS.Sign(m, OTS.sk$_{state}$).
9: Construct the signature: $\sigma \leftarrow$ (OTS.σ, OTS.pk$_{state}$, DGM.pos, DGM.p)
10: $state \leftarrow state + 1$
11: usk $\leftarrow$ (c, $state$, **keys**)
12: **return** σ

Algorithm 2 DGM algorithms part 2

DGM.Verify

Input: $m, \sigma, \mathsf{gpk}, \mathsf{param}$
Output: $0/1$

Parse $\sigma \rightarrow (\mathsf{OTS}.\sigma, \mathsf{OTS.pk}, \mathsf{DGM.pos}, \mathsf{DGM.p})$
Parse $\mathsf{gpk} \rightarrow (\mathsf{DGM.gpk}, \mathsf{DGM.rvk}, rvks)$
Parse $\mathsf{DGM.rvk} \rightarrow (\mathsf{DGM.rvk}_0, \mathsf{DGM.rvk}_1)$
Parse $\mathsf{param} \rightarrow h_{\mathsf{IMT}}$
if $\mathsf{SPE.Dec}(\mathsf{SPE.Enc}(m, \mathsf{DGM.rvk}_0, \mathsf{DGM.pos}), \mathsf{DGM.rvk}_1, \mathsf{DGM.pos}) = \perp$ **then**
  return 0
end if
if $\mathsf{OTS.Verify}(m, \mathsf{OTS}.\sigma, \mathsf{OTS.pk}) = 0$ **then**
  return 0
end if
Verify the path DGM.p starting with $\mathsf{H}(\mathsf{OTS.pk}, \mathsf{DGM.pos})$. This outputs the root $\mathsf{DGM.gpk}'$ (See Figure 3.2)
if $\mathsf{DGM.gpk}' \neq \mathsf{DGM.gpk}$ **then**
  return 0
end if
verification of validity of Fk with $\mathcal{M}$
return 1

DGM.Open

Input: m, σ, gsk
Output: $\mathcal{U}/\perp$

Parse $\mathsf{gsk} \rightarrow (\mathcal{M}.\mathsf{sk}, \mathbf{FK}, \mathbf{FN}, \mathbf{OTS}, priv\mathcal{L})$
Parse $\sigma \rightarrow (\mathsf{OTS}.\sigma, \mathsf{OTS.pk}, \mathsf{DGM.pos}, \mathsf{DGM.p})$
$\mathsf{DGM.i} \leftarrow \mathsf{SE.Dec}(\mathsf{DGM.pos}, \mathcal{M}.\mathsf{sk})$
return $\mathcal{U}$ from $priv\mathcal{L}$ corresponding to DGM.i or $\perp$ if no member has been found in $priv\mathcal{L}$

DGM.Revoke

Input: $\mathcal{U}, \mathsf{gsk}, \mathsf{gpk}, \mathbf{members}, \mathsf{param}$
Output: $\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}$

Parse $\mathsf{gsk} \rightarrow (\mathcal{M}.\mathsf{sk}, \mathbf{FK}, \mathbf{FN}, \mathbf{OTS}, priv\mathcal{L})$
Parse $\mathsf{gpk} \rightarrow (\mathsf{DGM.gpk}, \mathsf{DGM.rvk}, rvks)$
Parse $\mathsf{DGM.rvk} \rightarrow (\mathsf{DGM.rvk}_0, \mathsf{DGM.rvk}_1)$
Parse $\mathsf{param} \rightarrow h_{\mathsf{IMT}}$
$rvks \leftarrow rvks + 1$ number of DGM.pos associated with $\mathcal{U}$
$\mathsf{DGM.rvk}_0 \leftarrow \mathsf{SPE.KeyGen}(1^\lambda, rvks)$
for 1 to $rvks$ **do**
  $rvk_i \leftarrow \mathsf{SPE.Punc}(rvk_{i-1}, \mathsf{DGM.pos}_i)$ ▷Puncture of all DGM.poss belonging to revoked members
end for
$\mathsf{DGM.rvk}_1 \leftarrow rvk_i$
$\mathsf{DGM.rvk} \leftarrow (\mathsf{DGM.rvk}_0, \mathsf{DGM.rvk}_1)$
$\mathsf{gpk} \leftarrow (\mathsf{DGM.gpk}, \mathsf{DGM.rvk}, rvks)$
return gpk

Algorithm 3 DGM subroutines

DGM.OTSRequest

Input: $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}}$
Output: $\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}}$

Parse $\mathsf{gpk} \rightarrow (\mathsf{DGM.gpk}, \mathsf{DGM.rvk}, rvks)$
Parse $\mathsf{gsk} \rightarrow (\mathcal{M}.\mathsf{sk}, \mathbf{FK}, \mathbf{FN}, \mathbf{OTS}, priv\mathcal{L})$
Parse $\mathsf{param} \rightarrow h_{\mathsf{IMT}}$
Parse $\mathsf{usk} \rightarrow (c, state, \mathsf{keys})$
$\mathcal{U}$ generates $n = \mathsf{H}(B, c)$.
$\mathcal{U}$ sends (B, n) to $\mathcal{M}$
$\mathcal{M}$ verifies the validity of n
If the verification succeeds, then $\mathcal{M}$ runs the $\mathsf{Update}(B, \mathcal{U}, \mathsf{gsk})$ subroutine resolving new assigned OTS keys $\mathsf{OTS.Keys}$ to $\mathcal{U}$ and updated the lists of gsk.
$\mathcal{M}$ sends the $\mathsf{OTS.Keys}$ to $\mathcal{U}$ and stores all indexes of the $\mathsf{OTS.Keys}$ in the list $priv\mathcal{L}$ at the entry $\mathcal{U}$.
$\mathcal{U}$: $\mathsf{usk} \leftarrow (c, state, \mathbf{keys})$
return $\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}}$

DGM.Update

Input: $B, \mathcal{U}, \mathsf{gsk}$
Output: gsk

Parse $\mathsf{gsk} \rightarrow (\mathcal{M}.\mathsf{sk}, \mathbf{FK}, \mathbf{FN}, \mathbf{OTS}, priv\mathcal{L})$
Select B random internal Nodes n_i of IMT
for $i = 1$ to B **do**
  if $n_i \in \mathbf{FN}$ **then**
    Assign the next available leaf to $\mathcal{U}$ of the corresponding SMT associated with n_i
    if n_i has no leaf left **then**
      Remove n_i from **FN**
    end if
  else
    Create a new SMT t for n_i
    $newOTS \leftarrow 2^{h_{\mathsf{IMT}} - n_i.level - 1}$
    for $i = 1$ to $newOTS$ **do**
      $(\mathsf{OTS.pk}_i, \mathsf{OTS.sk}_i) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$
      $\mathsf{DGM.i} \leftarrow (t, i)$
      $l_i \leftarrow (\mathsf{OTS.pk}_i, index)$
    end for
    Generate Merkle tree on the shuffled *leaves* which give the Merkle root r_{SMT} (See Figure 3.2)
    $Fk \leftarrow \mathsf{SE.Dec}(n_i, r_{SMT})$
    assign the first of OTS key pair to $\mathcal{U}$
    add all other OTS keys to **OTS**
    add Fk to **FK**
    add n_i to **FN**
  end if
end for
$\mathsf{gsk} \leftarrow (\mathcal{M}.\mathsf{sk}, \mathbf{FK}, \mathbf{FN}, \mathbf{OTS}, priv\mathcal{L})$
return gsk

3.3.3 Security analysis

DGM assumes that the manager $\mathcal{M}$ is trusted. Therefore, $\mathcal{M}$ is available, generates trees and OTS keys correctly and will not collide with the adversary. We also assume than the communications for DGM.Join and DGM.OTSRequest take place over secure channels. The adversary $\mathcal{A}$ follows the security games presented in Chapter 2.9 in the Figure 2.3. He has access to private information of corrupted members and to the private list $priv\mathcal{L}$. DGM achieves correctness, non-frameability, anonymity, traceability, tracing-soundness and revocability (Chapter 2.9).

Theorem 1 (Correctness)**.** *Let* DGM *be the construction provided in Algorithm 1 and 2 with a secure hash function* H*, an* IND − CPA *secure symmetric encryption* SE*, a secure* OTS *scheme, and a secure symmetric puncturable encryption* SPE*.* DGM *achieves correctness in Definition 2.9.2.*

Proof. The correctness of DGM is ensued directly from the correctness of its underlying primitives H, SE, OTS and SPE. □

Theorem 2 (Non-frameability)**.** *Let* DGM *be the construction provided in Algorithm 1 and 2 with a secure hash function* H*, a* IND − CPA *secure symmetric encryption* SE*, a secure* OTS *scheme, and a secure symmetric puncturable encryption* SPE*.* DGM *achieves non-frameability based on Definition 2.9.3.*

Proof. To forge a signature, $\mathcal{A}$ (Figure 2.3) needs to have a valid path from OTS.pk to DGM.gpk by the properties of a collision-resistant hash function, it is computationally not feasible instead knowing every single element of DGM.p and the DGM.pos of OTS.pk in the tree. The first possibility consists of $\mathcal{A}$ taking a valid path of a corrupted member and trying to find DGM.pos′ associated with a non-corrupted member such that $\mathsf{H}(\mathsf{OTS.pk}', \mathsf{DGM.pos}') = \mathsf{H}(\mathsf{OTS.pk}, \mathsf{DGM.pos})$. By the collision resistance property of H, it is not computationally feasible (See Definition 2.2.1). If $\mathcal{A}$ has access to a valid OTS.pk, he needs to recover OTS.sk to generate a valid OTS.σ and thanks to the security property of OTS, it not feasible. Therefore the probability of forging a signature is negligible as $\Pr[\mathsf{DGM.Verify}(m, \sigma', \mathsf{gpk}, \mathsf{param}) = 1] < \mathsf{negl}(\lambda)$. Another possibility for $\mathcal{A}$ is to create his own path to DGM.gpk by generating his own Fk. However, because of the assumption that $\mathcal{M}$ is trusted, the Fk verification will not pass. Moreover, if he uses the parameters of corrupted members, DGM.Verify would still reject σ' because we assumed that $\mathcal{M}$ is trusted and works correctly. Therefore, all DGM.poss of corrupted member has been punctured. Thanks to the correctness of SPE (See Definition 3.1.2), the execution of DGM.Verify outputs 0 (see line 1 of Algorithm 2). □

Theorem 3 (Anonymity)**.** *Let* DGM *be the construction provided in Algorithm 1 and 2 with a secure hash function* H*, an* IND − CPA *secure symmetric encryption* SE*, a secure* OTS *scheme, and a secure symmetric puncturable encryption* SPE*.* DGM *achieves anonymity based on Definition 2.9.4.*

Proof. $\mathcal{A}$ (see Figure 2.3) has access to all OTS key pairs and paths except for two pairs of two different non-corrupted members and the private lists of $\mathcal{M}$. Therefore, when calling the challenge, he knows the index associated with the non-corrupted members. The only element of the signature σ, which reveals information about the identity of the signer is DGM.pos, which corresponds to $\mathsf{SE.Enc}(\mathsf{DGM.i}, \mathcal{M}.\mathsf{sk})$. Thanks to the IND − CPA security (See Definition 2.3.1) of SE, $\mathcal{A}$ can only guess the identity if he breaks the scheme SE. Therefore, if we assume that SE is secure and that $\mathcal{M}$ does not reveal $\mathcal{M}$.sk, then $\mathcal{A}$ can not distinguish between associated DGM.pos′ with

the ones from the member. Moreover, to insure anonymity the trusted $\mathcal{M}$ have to keep secret the list of **FN** and **FK** to not give precisely the apparition time of new Fks which could lead to leak some information on the call DGM.OTSRequest of a member $\mathcal{U}$. This secrecy added with the "Shuffle" process hide the creation of a new SMT. Moreover, revoked members preserved their anonymity because the revocation process works on the DGM.pos elements of revoked members. In this framework, $\mathcal{A}$ cannot find the identity of the revoked users due to the security of SE scheme. □

Theorem 4 (Traceability). *Let* DGM *be the construction provided in Algorithm 1 and 2 with a secure hash function* H*, an* IND − CPA *secure symmetric encryption* SE*, a secure* OTS *scheme, and a secure symmetric puncturable encryption* SPE*.* DGM *achieves traceability based on Definition 2.9.5.*

Proof. The adversary $\mathcal{A}$ (see Figure 2.3) needs to construct a valid signature which cannot be attributed to any members. Therefore, he needs to change DGM.pos in σ. As $\mathcal{A}$ can corrupt members, he has access to private parameters of corrupted members, hence he knows the valid path to DGM.gpk. The need to generate a valid OTS signature first, which can be successfully done by using an OTS key pair of corrupted members. Secondly, he needs to change the variable DGM.pos associated with the OTS key pairs that he used to sign the message m by a random element $\mathsf{DGM.pos}'$, such that $\mathsf{H}(\mathsf{OTS.pk}, \mathsf{DGM.pos}) = \mathsf{H}(\mathsf{OTS.pk}, \mathsf{DGM.pos}')$ However, because of the collision resistance property of H (see Definition 2.2.1) it is not computationally feasible. Therefore, the probability that $Pr[\mathsf{DGM.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param}) = 1] < negl(\lambda)$ with $\sigma = (\mathsf{OTS}.\sigma, \mathsf{OTS.pk}, \mathsf{DGM.p}, \mathsf{DGM.pos}')$. □

Theorem 5 (Tracing-soundness). *Let* DGM *be the construction provided in Algorithm 1 and 2 with a secure hash function* H*, an* IND − CPA *secure symmetric encryption* SE*, a secure* OTS *scheme, and a secure symmetric puncturable encryption* SPE*.* DGM *achieves tracing-soundness based on Definition 2.9.6.*

Proof. $\mathcal{A}$ (see Figure 2.3) needs to construct a valid signature that can be assigned to two different honest group members. The only way to break the tracing-soundness, it requires to have two element $\mathsf{DGM.pos} = \mathsf{SE.Enc}(\mathsf{DGM.i}, \mathcal{M}.\mathsf{sk})$ and $\mathsf{DGM.pos}' = \mathsf{SE.Enc}(\mathsf{DGM.i}', \mathcal{M}.\mathsf{sk})$ belonging to two different group member such that $\mathsf{DGM.pos} = \mathsf{DGM.pos}'$. The tracing soundness is provided directly from the security property of the block cipher used by $\mathcal{M}$. Indeed, this is computationally infeasible due to the collision-resistant property of a block cipher under the same secret key. This means that the only way to have $\mathsf{DGM.pos} = \mathsf{DGM.pos}'$ is having $\mathsf{DGM.i} = \mathsf{DGM.i}'$, which is only possible if $\mathcal{M}$ is corrupted. However, we assume in this work that $\mathcal{M}$ is doing his job correctly when interacting with group members. Therefore, we can conclude that DGM achives tracing-soundness. □

Theorem 6 (Revocability). *Let* DGM *be the construction provided in Algorithm 1 and 2 with a secure hash function* H*, an* IND − CPA *secure symmetric encryption* SE*, a secure* OTS *scheme, and a secure symmetric puncturable encryption* SPE*.* DGM *achieves revocability based on Definition 2.9.7.*

Proof. To break the revocability of DGM, the adversary $\mathcal{A}$ (see Figure 2.3) needs to construct a valid signature from the secret key of a revoked member. In DGM, we assume that the central authority $\mathcal{M}$ is doing his work correctly and, therefore, all elements DGM.pos belonging to revoked member have been punctured. Therefore, from the definition of SPE (see Definition 3.1.2), we have $\mathsf{SPE.Dec}(\mathsf{SPE.Enc}(m, \mathsf{DGM.rvk}_0, \mathsf{DGM.pos}), \mathsf{DGM.rvk}_1, \mathsf{DGM.pos}) = \perp$, when DGM.pos

TABLE 3.3: Signature size comparison

Group signature	Signature size (KB)	Max message size (KB)
Boneh et al. [BEF18]	1331	-
Katz et al. [KKW18]	315	-
DGM[1]	2.82	5.03
DGM[2]	9.54	2.51

has been punctured by $\mathcal{M}$, so the probability of having DGM.Verify outputs 1, or in other words having a valid signature, is negligible, when the group member has been revoked. □

3.4 Evaluation

In this section, we compare the performance of different phases of our DGM construction. We also evaluate the numerical performance of our DGM setup algorithm DGM.Setup and show its advantage compared to GM.Setup. This section concludes with a discussion over the overhead generated by our revocation technique on the verification procedure. All experiments have been implemented in Java and executed on a machine Xeon-Gold 6150 with a RAM of 16GB.

3.4.1 σ's size

The main advantage of DGM is its constant signature size (with respect to the number of group members N) compared to the current state-of-the-art. Table 3.1 and 3.5 demonstrate this trend. Although, Ling et al. [LNWX18] proposed a lattice-based group signature with a signature size independent of the number of group members, they have not explored the applicability of their results. A weakness of our scheme is the size of the public parameters, which are increasing when members are revoked, because the size of DGM.rvk increases after each revocation if we use SPE proposed in [SYL+18]. Moreover, in terms of practical application, we also compare our scheme with the group signature provided in [KKW18], which has the current dynamic group signature benchmark with symmetric primitives. Although the group signature given by Katz et al. [KKW18] does not explicitly provide a dynamic group signature, their construction is similar to the Boneh et al. [BEF18] group signature, so it could be considered as a dynamic group signature. The smallest signature size provided by Katz et al. is 315KB for a group of 2^7 members. The size $|\sigma|$ of σ is $|\mathsf{OTS}.\sigma| + |\mathsf{OTS}.\mathsf{pk}| + |H| \cdot h_{\mathsf{IMT}} + |\mathsf{DGM}.\mathsf{pos}|$. Using all symmetric standard and we set $h_{\mathsf{IMT}} = 20$, which is the maximum applicable h_{IMT} size. Table 3.3 compares signature sizes of two different instances of DGM, achieving different levels of security, with those group signature from [KKW18] and [BEF18]. Table 3.5 situates DGM signature size with the current state-of-the-art It shows that our instances approximately achieves a signature size 115 times smaller than Katz et al. group signature [KKW18]. The third column presents the maximum message size for which our σ can achieve a smaller signature size compared to those in [KKW18].

[1] SHA-256, AES-256, W-OTS, message size = 256 bits
[2] SHA3-512, AES-512, W-OTS, message size = 512 bits

TABLE 3.4: Verification: DGM vs G-Merkle

	Dynamicity	Revocation
DGM overhead	Path verification (Figure 3.2) Interaction with $\mathcal{M}$	Execution of SPE.Enc and SPE.Dec

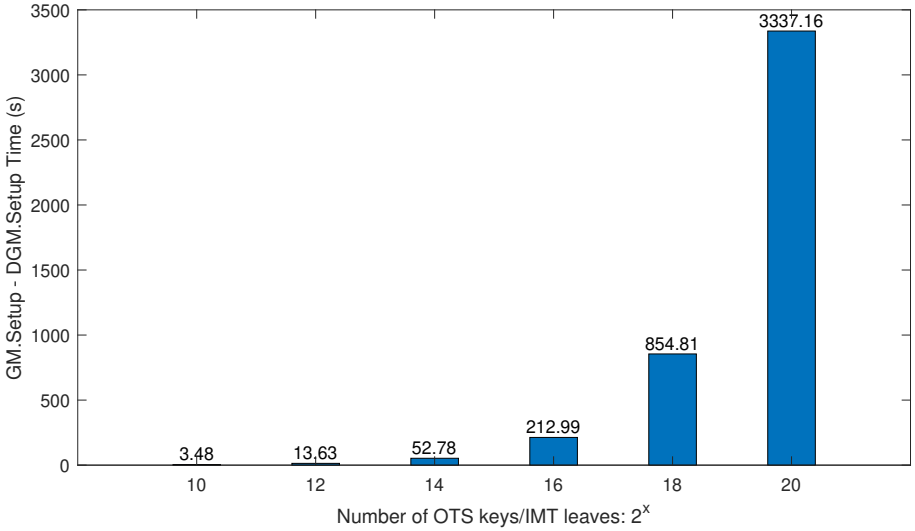


FIGURE 3.3: Setup performance comparison: GM.Setup vs DGM.Setup

3.4.2 DGM signature verification

Another advantage of our scheme is that a valid signature will remain valid during the whole life of the group. In DGM, DGM.gpk and the element DGM.p of a signature allow a member to prove its membership, although other schemes work with accumulators to prove membership in a group [BEF18]. To sign, a member has to be updated about the state of accumulators in order to generate a valid membership proof in the signature. Nevertheless, when a new member joins the group, the membership proof will be updated. This update has as consequence that all of the other members of the group need to update to prove their membership in their next signature. Although a member keeps updating regularly, the risk of generating a signature with a non-valid proof of membership still exists because if a new member joins between the last update and the signing time, then it will have an invalid signature. This interactive process DGM.OTSRequest can be seen as an update process and hence the risk of generating a signature with a non-valid DGM.p does not exist in our scheme. The reason why this risk is lifted is that the joining of new members in the group has no influence on the older members. However, the DGM verification has a major cost due to the extension (see Table 3.4). The dynamicity brings an interaction between $\mathcal{M}$ and the verifier, which could be an important limitation for possible application. Our futures objectives will be to suppress the interaction without jeopardizing the security of DGM. Furthermore, the use of SPE to provides another additional cost, which is discussed in a more detailed way in Section 3.4.4.

3.4.3 DGM setup performance

Figure 3.3 shows the difference in numerical performances of both setup phases of G-Merkle and DGM implemented in Java. The implementation relies on the SHA-256 as hash functions. For OTS purpose, we decided to implement W-OTS [Hül13]. The "Shuffle" of the leaves was done with the block cipher AES and the Java Interface Comparator with the Timsort algorithm inspired from [McI93]. The results of our implementation are shown in Figure 3.3 which represents an average of five experiments, highlighting the difference of performance in seconds between setup processes with different sizes of Merkle trees. A difference between G-Merkle and DGM is the time of OTS key pairs generation. During the setup, G-Merkle needs to generates all $N \cdot B$ OTS keys for a group of N members with B possible signature for each of the members. While DGM delays and divides the generation of the OTS keys over the time and needs only to generate IMT. Our results presented in Figure 3.3 shows that this delay improves the applicability of the group signature. For example, when $N \cdot B = 2^{20}$, DGM saves more than 3300 seconds, and therefore the DGM can start functioning 3300 seconds earlier than G-Merkle. Additionally, our approach allows the group members sign an unlimited number of signature while G-Merkle has to limit its member to a maximum of 4096 signatures because of its inefficient setup process.

3.4.4 DGM revocation procedure with symmetric puncturable encryption

The next section discusses a possible implementation of the innovative revocation technique using symmetric puncturable encryption (SPE) and its influence over the DGM verification procedure. We follow the proposition of SPE's original paper [SYL+18], which works using exclusively cryptographic hash functions and block ciphers.

GGM puncturable pseudorandom function

To build SPE, we first need a puncturable pseudorandom function (PPRF) (see Definition 3.1.1) which can be implemented using GGM PRF [Lyn20]. Figure 3.4 illustrates an example for GGM PPRF, which can be represented as a binary tree. The secret key k is the element at the top of a binary tree. At the heart of GGM PPRF, there is a pseudorandom number generator (PRNG) defined s $\mathsf{PRNG} : \{0,1\}^{2\lambda} \to \{0,1\}^{4\lambda}$ where the output of PRNG can be divided into two parts of the same length: $\mathsf{PRNG}(x) = (\mathsf{PRNG}_0(x), \mathsf{PRNG}_1(x))$, where $\mathsf{PRNG}_i : \{0,1\}^{2\lambda} \to \{0,1\}^{4\lambda}$. This takes as input an element x of n bits $\mathsf{PRF}(k,x) \to (\mathsf{PRNG}_{x_n}(\mathsf{PRNG}_{x_{n-1}}(\cdots \mathsf{PRNG}_{x_1}(x)\cdots)))$. In the example illustrated in Figure 3.4, we have $x \in \{0,1\}^4$. Then, for example in Figure 3.5, the puncture algorithm is called on $x = 0011$(i.e. $\mathsf{F.Punc}(k, 0011)$). The puncture key k_x is represented by the green nodes in Figure 3.5. Due to the property of the PRNG (one-wayness), it is infeasible to compute the value of the red node in Figure 3.5, the punctured element, from the punctured key k_x

SPE algorithms

To implement SPE, we follow the construction presented in [SYL+18], which means that $\mathsf{SPE.Enc}(msk, m, t)$, $msk = (\mathsf{sk}_0)$ executes the following steps:

- $\mathsf{sk}_i \leftarrow \mathsf{H}(\mathsf{sk}_{i-1}, i)$ for all $1 \leq i \leq d$
- $k \leftarrow \bigoplus_{i=0}^{d} \mathsf{PRF}(\mathsf{sk}_i, t)$, where PRF is the GGM construction previously described.

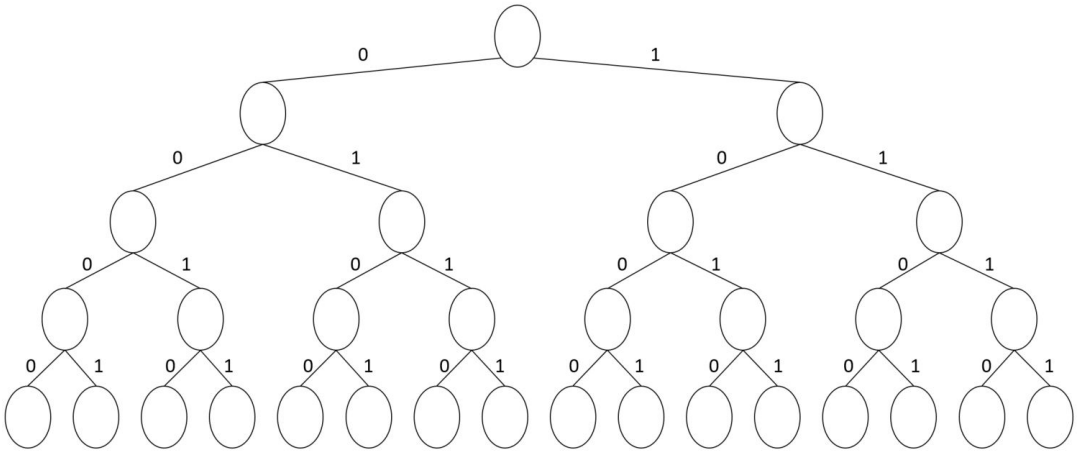


FIGURE 3.4: GGM keys intial tree

- $\mathsf{SPE.ct} \leftarrow \mathsf{SE.Enc}(m, k)$.

In the context of DGM, we have $\mathsf{DGM.rvk} = (\mathsf{DGM.rvk}_0, \mathsf{DGM.rvk}_1)$ where $msk = (\mathsf{sk}_0) = \mathsf{DGM.rvk}_0$, which is the root of the GGM tree (see Figure 3.4), and the public $rvks$ which indicates the total number of punctured elements or, in DGM, the total of revoked DGM.pos elements. When $\mathcal{M}$ revokes the ability to sign to a group member, he execute the puncture algorithm $\mathsf{F.Punc}(\mathsf{DGM.rvk}_0, \mathsf{DGM.pos})$ on all DGM.pos elements belonging to this member. The puncture algorithm is simply an execution of the puncture algorithm of the GGM PPRF presentation. For example, if the revoked member owns two $\mathsf{DGM.pos}_1 = 0011$ and $\mathsf{DGM.pos}_2 = 1001$, the puncture algorithm would output the puncture key $\mathsf{DGM.rvk}_1$ which corresponds to the green nodes in Figure 3.6.

Then in DGM's verification procedure, it executes the SPE.Dec algorithm line 5 in Algorithm 2. $\mathsf{SPE.Dec}(\mathsf{SPE.Enc}(m, \mathsf{DGM.rvk}_0, \mathsf{DGM.pos}), \mathsf{DGM.rvk}_1, \mathsf{DGM.pos})$. The algorithm $\mathsf{SPE.Dec}(m, \mathsf{SPE.sk}_i, t)$, where $\mathsf{SPE.sk}_i = (msk_i, psk_1, \cdots, psk_i), msk_i = (\mathsf{sk}_i, d)$ can be implemented with the following algorithm:

- if $i < d$, $\mathsf{sk}_l \leftarrow \mathsf{H}(\mathsf{sk}_{l-1}, l)$ forall $i + 1 < l \leq d$
- $k \leftarrow \bigoplus_{s=1}^{i} \mathsf{F.Eval}(psk_s, ct, t) \oplus \bigoplus_{l=i}^{d} \mathsf{PRF}(\mathsf{sk}_l, t)$
- $m \leftarrow \mathsf{SE.Dec}(\mathsf{SPE.ct}, k)$

In the case of DGM, we have $psk_s = \mathsf{DGM.rvk}_1$, which is, as previously stated, the green nodes in Figure 3.6. $\mathsf{DGM.rvk}_1$ allows to recompute each leaf of the tree, except for the red ones. Due to this puncture algorithm, it is infeasible to compute the punctured elements (the red nodes) from $\mathsf{DGM.rvk}_1$ (the green nodes). This comes from the one-wayness of the PRGN at the heart of the construction of GGM PPRF.

SPE performance evaluation

We evaluate the computation cost of the revocation procedure described previously and its overhead generated on the DGM verification procedure. Similar to the rest of the DGM's evaluation section, we implemented the SPE using SHA-256 as PRNG and AES as a block cipher. We increase the number of revoked elements DGM.pos to evaluate the applicability of our innovative technique.

Figure 3.7 shows the results of the revocation analysis by summarizing the overhead caused by the revocation procedure for different sizes of DGM.pos, so for different

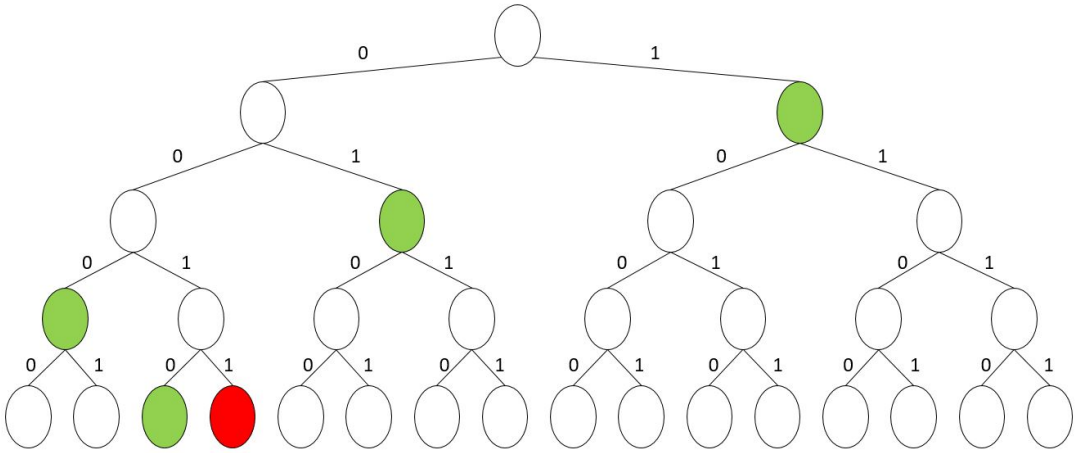


FIGURE 3.5: Puncture keys with the element $\mathsf{DGM.pos} = 0011$ punctured/revoked

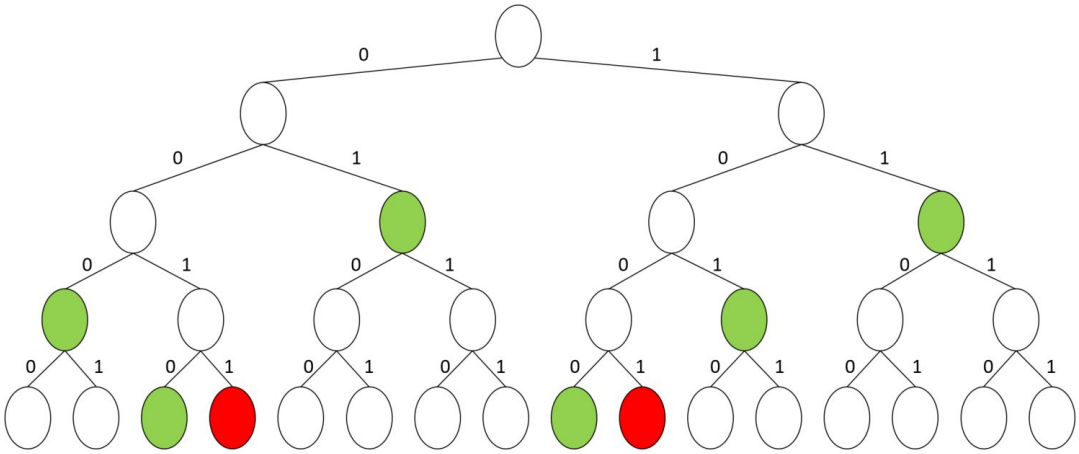


FIGURE 3.6: Puncture keys with the element $\mathsf{DGM.pos}_1 = 0011$ and $\mathsf{DGM.pos}_1 = 1001$ punctured

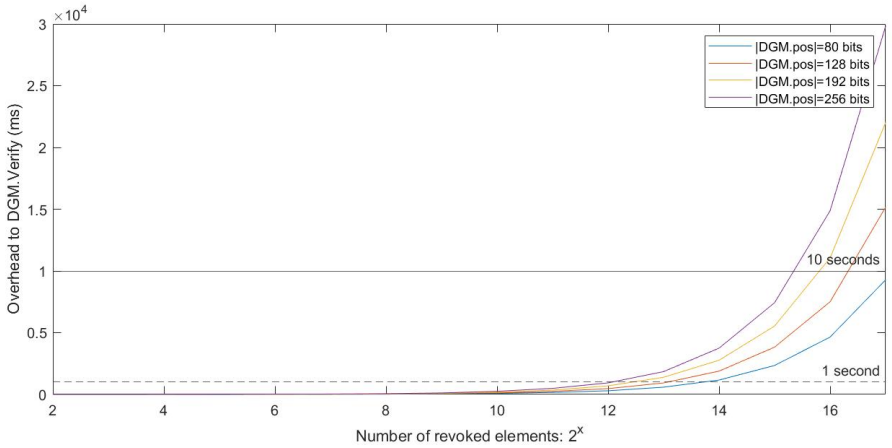


FIGURE 3.7: Overhead of SPE revocation for DGM.Verify

security levels. Figure 3.7 presents the average of five experiments We examine the overhead coming from our revocation procedure by increasing the number of revoked elements (e.i. DGM.pos). We revoke up to 2^{18} DGM.pos elements and we also evaluate with different size of DGM.pos. This demonstrates that future works could focus on improving the running time of the SPE algorithms. The overhead complexity of the revocation on DGM.Verify algorithm can be represented by $t \cdot |\mathsf{DGM.pos}|$ calls of the PRNG.

Because of the quantum Grover's search algorithm [Gro96], if the level of 128-bits of post-quantum security is wanted, we require $|\mathsf{DGM.pos}| = 256$ bits. Our results in Figure 3.7 demonstrate that the overhead of our revocation technique is acceptable until 2^{12} revoked elements as it generates less than 1 seconds. When at least 2^{15} elements have been punctured (i.e. revoked) the overhead coming from this revocation techniques is more than 7 seconds. Consequently, a public list of revoked elements would most likely offer a better alternative when more than 2^{15} elements are revoked.

We also evaluate the expected overhead on the verification procedure with different level of post-quantum security. For a level of 96 bits of post-quantum security, we can have $|\mathsf{DGM.pos}| = 192$ bits. In this case, we evaluate an overhead of less than one second for 2^{13} revoked elements and it can verify if the signer has been revoked in just over 10 seconds when 2^{16} element has been revoked.

For a level of 64 bits of post-quantum security, we can have $|\mathsf{DGM.pos}| = 128$ bits. In this case, we evaluate an overhead of less than one second for 2^{13} revoked elements and in 6 seconds when 2^{16} have been revoked.

For the smallest level of post-quantum security (40 bits and $|\mathsf{DGM.pos}| = 80$ bits), our results demonstrate that the overhead of our revocation technique is acceptable until 2^{14} revoked elements. Indeed, the overhead caused by the revocation is less than one second. If less than 2^{17} elements have been revoked, then the overhead of revocation is less than 10 seconds. With this level of security, we recommend using SPE when less than 2^{17} elements are revoked. For a higher number of revoked elements, a public list would most likely offer a better option.

One of the advantages of our revocation technique comes from the save that we made in terms of public element sizes. A revocation list can be considered as part of the public key if we consider that we need to go through the revocation list to verify that the signer has not been revoked. A revocation list grows linearly with the number of revoked members. In our case, each revoked key is associated with a leaf of the GGM tree. When a member is revoked, the manager erases the path

to the leaf corresponding to the $\mathsf{DGM.pos}$. This means that, at most, our public element $\mathsf{DGM.rvk}_1$ would be composed of $2^{|\mathsf{DGM.pos}|-1}$ of 2λ bits, where λ is the level of post-quantum security.

The major advantage of our revocation procedure through SPE is that the anonymity of the revoked members is assured. In the traditional approach of using revocation lists, the secret key of the revoked members is published in a public list, which means that all their previously signatures are no longer anonymous. Indeed, because of their publicly available secret keys, we can generate their signature by identifying any matches with the already generated signatures.

3.5 Chapter conclusion

In this chapter, we presented a fully dynamic post-quantum group signature (called DGM), which extends the static G-Merkle approach, which cannot handle large group (see Table 3.5), to allow new users to join the group at any time. Similar to G-Merkle, our scheme relies only on symmetric primitives. Additionally, DGM members can sign an unlimited number of messages. This is in contrast to G-Merkle members who can perform a maximum of 4096 signatures. While other dynamic post-quantum groups are often criticized for generating signatures of large signature sizes, as illustrated in Figure 3.5, our proposal suggests that the size of a signature can be reduced by using only symmetric primitives. Moreover, our competitive signature sizes do not rely on the number of group members outperforming other state-of-the-art group signatures as [KKW18] at the time of DMG's design. This can be considered as the benchmark for applications of post-quantum dynamic group signature constructed with symmetric primitives. DGM also provides a new approach for signature revocation with the use of symmetric puncturable encryption (SPE). A practical analysis was conducted to show and evaluate the applicability of our constructed DGM. Table 3.5 situates DGM with the state-of-the-art.

Even though DGM improves the state-of-the-art at the time of its publications, they are also some limitations to DGM. The first one is its interactive verification procedure which narrows its range of applications. Indeed, the central authority could experience, in practice, difficulty in handling large groups as each verification requires $\mathcal{M}$'s approval. The second limitation is the stateful nature of DGM, which means that each member needs to update a state after every single signature, and this means that a mistake from a member could lead to a security breach. The last issue of DGM is the level of trust in the central authority. As the DGM's manager is responsible for generating users' secret keys, a corrupted manager would be able to frame any group members, in other words $\mathcal{M}$ can generate a signature on behalf of a honest group members.

Therefore, we target solving these issues in the other contributions related to the Research Objective O1.

TABLE 3.5: Comparison of different post-quantum group signatures for 128-bits of post-quantum security. N denotes the number of members in the group, N.A.=Non Applicable, B=Maximum possible signatures per member, low= The group manager $\mathcal{M}$ cannot frame a signature, high=$\mathcal{M}$ can frame a signature.

Schemes		[DPLS18]	[EZS+19]	[BEF19]	[KKW18]	[EBM18]	DGM
$\|\sigma\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$
$\|\sigma\|$ (KB)	$N = 2^7$	581	39	1370	315	2.72	2.82
	$N = 2^{10}$	581	54	1850	418	N.A.	2.82
	$N = 2^{20}$	581	140	3450	770	N.A.	2.82
$\|\mathsf{usk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log B)$
$\|\mathsf{usk}\|$ (KB)		146	0.059	$0.032 \log N$	$0.032 \log N$	$2.2 \log B$	$2.8 \log B$
$\|\mathsf{gpk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\|\mathsf{gpk}\|$ (KB)		123	$N \cdot 11.1$	0.032	0.032	0.032	0.032
Sign (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$
Sign (ms)[a]		450	No data	No data	3000	N.A.	1
Verify (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$
Verify (ms)[a]		169	No data	No data	3000	N.A.	Interactive
PQ canditate		Lattice	Lattice	Symmetric	Symmetric	Symmetric	Symmetric
Stateless		Yes	Yes	Yes	Yes	No	No
Dynamic		Static	Partially	Static	Fully	Static	Fully
Traceable		Yes	Yes	No	Yes	Yes	Yes
Non-interactive		Yes	Yes	Yes	Yes	Yes	No
Level of Trust in $\mathcal{M}$		Low	Low	Low	Low	High	High

[a]The times provided are for $N = 2^{10}$ and taken from the original papers and were implemented in different programming languages and executed on different types of machines

Chapter 4

(T-)SGS: A Short and Stateless Dynamic Post-Quantum Group Signature based on Symmetric Primitives

This chapter presents the construction of two fully dynamic group signatures SGS and T-SGS. Both constructions were designed to solve the issue of an interactive verification procedure and the stateful feature met in the design of DGM presented in Chapter 3.

As stated in the previous chapters, group signatures [CVH91] are considered as a promising privacy-preserving instrument of real-world applications. A group signature allows members of a group to sign a digital message on behalf of the group while remaining anonymous. Due to this great promise, the research community has invested a significant amount of effort to develop and improve this primitive. In this primitive, group members are managed by a central authority called the manager. This authority is responsible for initiating the group and every prospective member needs to interact with his in order to become a group member. This authority is also able to trace a signature and to reveal the identity of the signer. This property is called traceability. However, if the application of group signatures is for Enhanced Privacy ID (EPID)[Int20], [BL09], similar to the one designed by Intel, this property is not desired. Similar to Chapter 3, we focus in this chapter only on fully dynamic group signatures which allow new members to join and leave the group at any time.

The first theoretical definition of group signature was written by Chaum et al. [CVH91]. However, Bellare et al. introduced the most commonly used definition for static [BMW03] and dynamic [BSZ05] group signature. More recently, Bootle et al. [BCC+16] advanced the definition of a fully dynamic group signature on which the work presented in this chapter is based.

The design of group signatures which can resist to the attack of an adversary holding a quantum computer became important, especially after the launch of the NIST post-quantum standardization process [AASA+20] which has attracted a lot of attention. There exist different post-quantum candidates to construct quantum-safe group signature. Lattice-based cryptography is the most commonly used one as the works of Gordon et al. [GKV10] Libert et al. [LLNW16],[LLM+16], Ling et al. [LNW15],[LNWX18], Del Pino et al. [DPLS18] and more recently Esgin et al. [EZS+19] demonstrate. Another promising post-quantum candidate is the construction based on symmetric primitive such as cryptographic hash function or block cipher. Only four group signatures have been constructed with symmetric primitives [EBM18],[KKW18],[BEF19] and [BLS+19].

The group signatures in [KKW18] and [BEF19] are both stateless and took advantage of the creation of zero-knowledge proof system [GMW91] constructed from symmetric primitives. A dynamic group signature constructed with symmetric primitives was first proposed by Boneh et al. [BEF19] and designed based on ZKB++ [CDG+17]. In 2018, a new draft of post-quantum group signature has been proposed by Katz et al. [KKW18], and constructed with an improved zero-knowledge proof reducing the signature size. However, as can be seen in Table 4.4, the first one does not provide the traceability property and the second one has been only described as a static group signature. These constructions are outperformed in terms of signature size by DGM, presented in Chapter 3, proposed by Buser et al. [BLS+19], which is also the only group signature providing both traceability and full dynamicity at the same time. However, the DGM scheme is a stateful group signature that requires signers to keep a state locally.

This chapter focuses on the design of fully dynamic group signature based on symmetric primitives to fill a significant research gap compared to the lattice-based group signature and it aims to solve the issue of our first group signature DGM presented in Chapter 3. We specifically targeted a **stateless** group signature with a **non-interactive** verification procedure.

4.0.1 Detailed contributions

Starting from state-of-the-art stateless SPHINCS+ signature [BHK+19], we constructed two post-quantum group signatures. The contributions of Chapter 4 can be summarized as follows:

- **SPHINCS Group signature (SGS)**: We present a naive transformation of the digital signature SPHINCS+ to a stateless and dynamic group signature called SGS (see Section 4.2.2). SGS is limited because traceability is not provided, which means the central authority cannot trace signatures and its revocation process is non-practical. SGS is built to facilitate the understanding of the construction of the fully functional group signature. Parts specific to SGS are presented in blue from here on.

- **Traceable SPHINCS Group signature (T-SGS)**: We extend our SGS construction to a fully functional SPHINCS group signature scheme with traceability (T-SGS) by adding a Merkle accumulator [KKW18] (see Section 4.2.3), solving the limitations of SGS. Thanks to the traceability, T-SGS provides an applicable revocation process. Parts specific to T-SGS are presented in red in the rest of the chapter.

- **Competitive signature size**: Both constructions achieve short signature sizes for a post-quantum security level of 128-bits. The naive SGS scheme enjoys a constant signature size of 33KB, regardless of the group size. The T-SGS construction achieves a signature size growing $\mathcal{O}(\log N)$, where N is the number of users in the group. It is only 501KB for $N = 2^{10}$ and $1,302$KB when N is increased to 2^{30}. When compared to the state-of-the-art post-quantum group signature schemes with the same features (or even less), our constructions match or outperform their signature size (see Section 4.3). The current state-of-the-art of post-quantum group signatures is summarized in Table 4.4

- **Algorithms efficiency**: Both constructions are efficient. SGS respects the great promises of SPHINCS+ with a signing algorithm performed in less than 1300ms and verifying processes executed in less than 2ms. The efficiency of

T-SGS depends on the number of group members and can be expressed as $\mathcal{O}(\log N)$, where N is the number of users in the group. The signing and verification can both be completed in respectively less than 12.1 and 7.1 seconds for a group of $N = 2^{20}$ members (see Section 4.3).

4.0.2 Techniques

Our constructed group signatures are driven by the digital signature named SPHINCS+ [BHK+19] and its strong guaranties in terms of post-quantum security as reported lately by NIST [AASA+20]. SPHINCS+ is a modification of the renowned Merkle signature [Mer89]. Recall that the Merkle signature was constructed over a one-time signature scheme (OTS) and a binary Merkle tree constructed with cryptographic hash function (See section 4.1.1). The security of this signature scheme relies on the properties of the underlying cryptographic hash function including one-wayness and collision resistance (see Definition 2.2.1). Our schemes will use the same properties to ensure security. The Merkle signature is known to be efficient and easy to implement but the signer needs to keep a state that has to be updated after each signature otherwise, it results in a break in the scheme's security. Moreover, the number of possible signatures is limited. SPHINCS+ signature suppresses the state issue by using a hyper-tree composed of multiple levels of Merkle trees instead of just one. To sign, the user computes the paths thanks to a pseudorandom function which takes as input the message and the public key of the scheme. In this chapter, we first present a naive transformation of SPHINCS+ into a group signature to facilitate the understanding of the construction of the fully functional group signature. To achieve that, the signer generates the path based on the signer's secret information (secret key). Its anonymity relies on the one-wayness property of the pseudorandom function and its membership to the group is proven by the fact only group members will be able to generate a valid hyper-tree

However, to ensure the traceability property to one of our group signature schemes, we need to prove that the input of a pseudorandom function was coming from a valid secret key. Therefore, in our second construction we use an accumulator [BEF19] constructed from symmetric primitives and zero-knowledge membership proof to insure the validity of the key. The idea is to generate the path in the Merkle tree depending on a keyed pseudorandom function with the group member's secret key and the signer will prove that he is the owner of the secret key that generated the path. This can be achieved by an accumulator and a non-interactive zero-knowledge proof system.

4.0.3 Outline of chapter 4

This chapter starts with Section 4.1 which define the additional cryptographic primitives required to construct (T-)SGS as the tweakable hash function, the few-time signature scheme FORS and the digital signature SPHINCS+. Section 4.2 presents our designed group signature SGS and T-SGS. It first introduce our non-traceable construction SGS in section 4.2.2 to then explain the transformation to the fully functional group signature T-SGS in section 4.2.3. All parts written in red in this chapter are specific to T-SGS construction. Then, we discuss the performance of both constructions in section 4.3. This chapter concludes with the discussion of the issues of these constructions and the future research direction of our research in section 4.4.

4.1 Additional definitions

(T-)SGS is constructed with cryptographic hash functions defined in Definition 2.2.1, pseudorandom functions (PRF) defined in Definition 2.6.1. It also relies on tweakable hash function and on the SPHINCS+ digital signature which are both defined below.

Definition 4.1.1 (Tweakable Hash Function)**.** *A tweakable hash function [BHK+19] is a function defined defined by the public parameter space $\mathcal{P}$ and tweak space $\mathcal{T}$:*

$$\mathsf{TH} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{\alpha} \rightarrow \{0,1\}^{1\lambda}, id \leftarrow \mathsf{TH}(P, T, m) \tag{4.1}$$

that takes as input a message m, a public parameters P belonging to a public parameters space $\mathcal{P}$ (i.e. $P \in \mathcal{P}$) and a tweak T from a tweak space $\mathcal{T}$ (i.e. $T \in \mathcal{T}$). It outputs a 2λ-bits hash value id.

Tweakable hash functions are used in SPHINCS+ to select the authentication path from the bottom to the top of the hyper-tree (see Figure 4.3) based on public parameters, an element that will change for each call (tweak) and the message to sign. Tweakable hash functions have the following security properties: post-quantum single function, multi-target-collision resistance for distinct tweaks and post-quantum single function, multi-target decisional second pre-image resistance for distinct tweaks. Both properties of tweakable hash functions are formally presented in [BHK+19].

4.1.1 SPHINCS+ digital signature

This section reviews the post-quantum stateless digital signature (see Definition 2.4.1) named SPHINCS+ [BHK+19]. SPHINCS+ is based on a One-time signature (OTS) scheme, the Merkle signature scheme and the FORS signature scheme. SPHINCS+ is based on a hyper-tree of height H, which consists of d layers of Merkle signature and one layer of FORS signature at the bottom. Figure 4.3 shows an example of a hyper-tree with $h = 9$ and $d = 3$. The SPHINCS+ public key is the Merkle root of the top layer of the hyper-tree.

Merkle signature

A Merkle signature (MS) [Mer89] is a signature scheme constructed on a Merkle tree and its properties. This scheme is presented for d levels in the SPHINCS+ hyper-tree (Figure 4.3). MS uses n One-Time Signature (OTS) (see Definition 2.5.1) key pairs and builds a Merkle tree over the leaves, where each leaf corresponds to a OTS public key. A node of the Merkle tree is the hash value of the concatenation of both its children nodes (see Figure 4.1). The MS public key $\mathsf{MS.pk}$ is the root of the tree, while the secret key is the set of all OTS secret keys. The next definition of MS is presented in the context of an hyper-tree (see Figure 4.3) and an example of MS is presented in Figure 4.1. MS was already defined in Chapter 3, but, in this chapter, we define its algorithms to facilitate the understanding of our construction introduced in Section 4.2.

Definition 4.1.2 (Merkle Signature)**.** *A Merkle signature* MS *[BDH11] is defined by the three following PPT algorithms:*

$(\mathsf{MS.pk}, \{(\mathsf{OTS.pk}_i, \mathsf{OTS.sk}_i)\}_{i=1}^{2^{height}}) \leftarrow \mathsf{MS.Gen}(seed, t, level, height)$*: This generates a Merkle tree (see Figure 4.1) and takes as inputs a seed $seed \in \{0,1\}^{2\lambda}$, an index t which is the tree number, level which indicates the level of the Merkle tree in the hyper-tree (Figure 4.3) and a height, which indicates height of the*

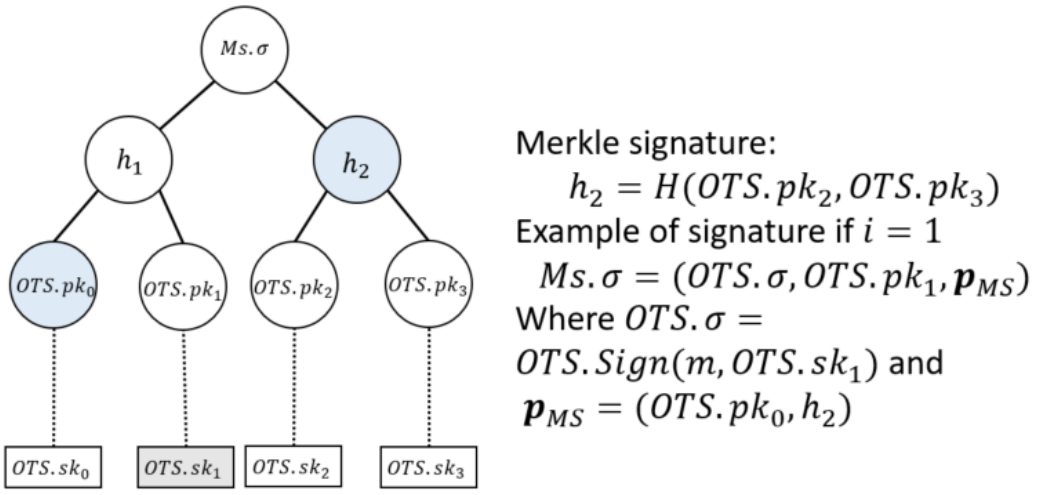


FIGURE 4.1: Merkle signature

tree. This algorithm outputs a set of OTS key pair generated with seed and the Merkle root MS.pk*. The algorithm constructs each OTS key pairs with a keyed PRF.*

MS.$\sigma \leftarrow$ MS.Sign$(m, \text{OTS.sk}_i, level)$*: This takes a message* m *to be signed, a OTS secret key* OTS.sk_i *where* i *correspond to the position of the signing OTS key pair in the Merkle tree and an element level which indicates the level in the hyper-tree. It outputs a signature* $\text{MS.}\sigma = (\text{OTS.}\sigma, \text{OTS.pk}, \text{auth})$ *composed of a OTS signature* $\text{OTS.}\sigma$*, the OTS public key* OTS.pk_i *and the path* auth *from* OTS.pk_i *to the root* MS.pk *(see Figure 4.1).*

$r \leftarrow$ MS.Verify$(m, \text{MS.}\sigma, index)$*: This takes as input a message* m*, a signature* $\text{MS.}\sigma$ *and an index which indicates the position of the of the OTS public key in the Merkle tree (for example in Figure 4.1* $index = \{0, 1\}$*). This verifies the OTS signature and then outputs the Merkle root* r *from index and* $\text{MS.}\sigma$*. In the context of SPHINCS+, only* $\text{MS.}\sigma$ *of the top layer is verified.*

FORS

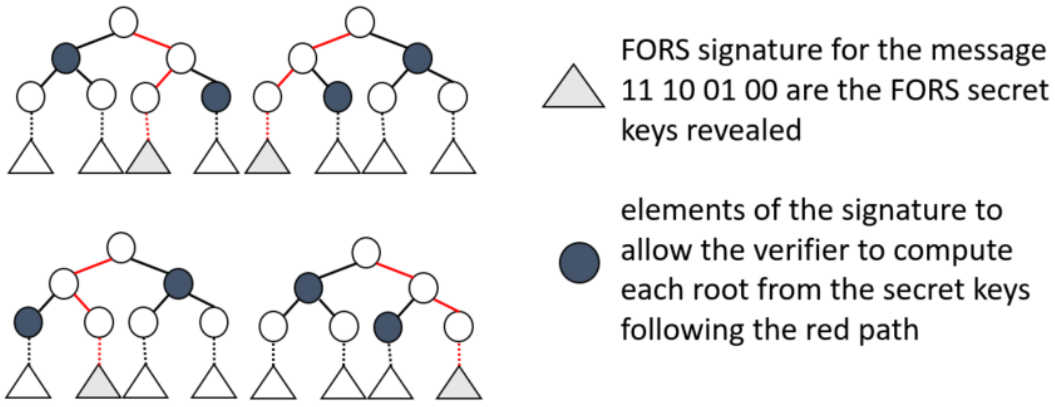


FIGURE 4.2: FORS

FORS [BHK+19] is a few-time signature scheme designed specifically for the SPHINCS+ signature. It is presented at the bottom of the SPHINCS+ hyper-tree (Figure 4.3). FORS has three parameters k, t and a (i.e. $t = 2^a$), where k is the number of trees with a height equal to a and t represents the number of leaves for each of the k trees. The trees are constructed similar to a Merkle tree but in contrast to the Merkle signature, the leaves are not OTS public key but random elements that are part of the secret key. Figure 4.2 illustrates an example of FORS signature with $k = 4$ and $a = 2$.

Definition 4.1.3 (FORS)**.** *FORS is defined by three PPT algorithms:*

$(\mathsf{FORS.pk}, \mathsf{FORS.sk}) \leftarrow \mathsf{FORS.KeyGen}(pos, seed)$*: This algorithm generates a secret key* FORS.sk *which is composed of* $(k \cdot t)$ λ*-bits values computed from a keyed PRF. The PRF takes as a key seed and as input the position pos in the hyper-tree of the FORS signature scheme (see Figure 4.3). The public key* FORS.pk *consists of* k *elements of* λ*-bits which are the Merkle roots constructed from the* $k \cdot t$ *element of the secret key used as leaves for the* k *trees of height* a *(see Figure 4.2).*

$\mathsf{FORS}.\sigma \leftarrow \mathsf{FORS.Sign}(m, \mathsf{FORS.sk})$*: The signing algorithm takes as input a message* m *to sign. It starts by dividing the message into* k *parts of* a *bits which give the address in each tree of which* k *secret elements (one per tree) will be part of the signature* FORS.σ*. The rest of* FORS.σ *is composed of* k *authentication paths of the* k *secret elements. An example of* FORS.σ *is illustrated in Figure 4.2.*

$\mathsf{FORS.pk} \leftarrow \mathsf{FORS.Verify}(m, \mathsf{FORS}.\sigma)$*: This algorithm recomputes the public key* FORS.pk *based on the message* m *and the signature* FORS.σ*. In the context of SPHINCS+,* FORS.σ *is not verified, it just need to recompute* FORS.pk*, to be able to verifies the full path on the hyper-tree.*

SPHINCS+ algorithms

The only Merkle tree generated during the key generation procedure is the one at the top of the Merkle Tree and its root will be the part of the public key. The full hyper-tree (see Figure 4.3) is never generated. The signer selects the path based on the public parameters and the message m. The signer then generates dynamically each Merkle tree needed based on the selected path. Each leaf of all internal trees is generated with a PRF function and a secret seed *seed*, which gives the possibility of generating the needed Merkle trees only (i.e the ones in full black in Figure 4.3). a OTS key pair is selected at each level of the hyper-tree and it is used to sign the root of the tree from the previous level. The verification of the signature starts with the generation of the path and follows with the computation of the FORS public key (see Figure 4.2 and definition 4.1.3). The path verification is performed similar to a Merkle signature and between each layer, we have a verification of a OTS signature. A signature is valid if the root of the hyper-tree can be computed from it.

4.2 Our group signatures: SGS and T-SGS

This section presents both constructions of our group signatures. We propose two different constructions: a naive derivation of SPHINCS+ named SGS without the traceability property; and another non-trivial construction named T-SGS providing

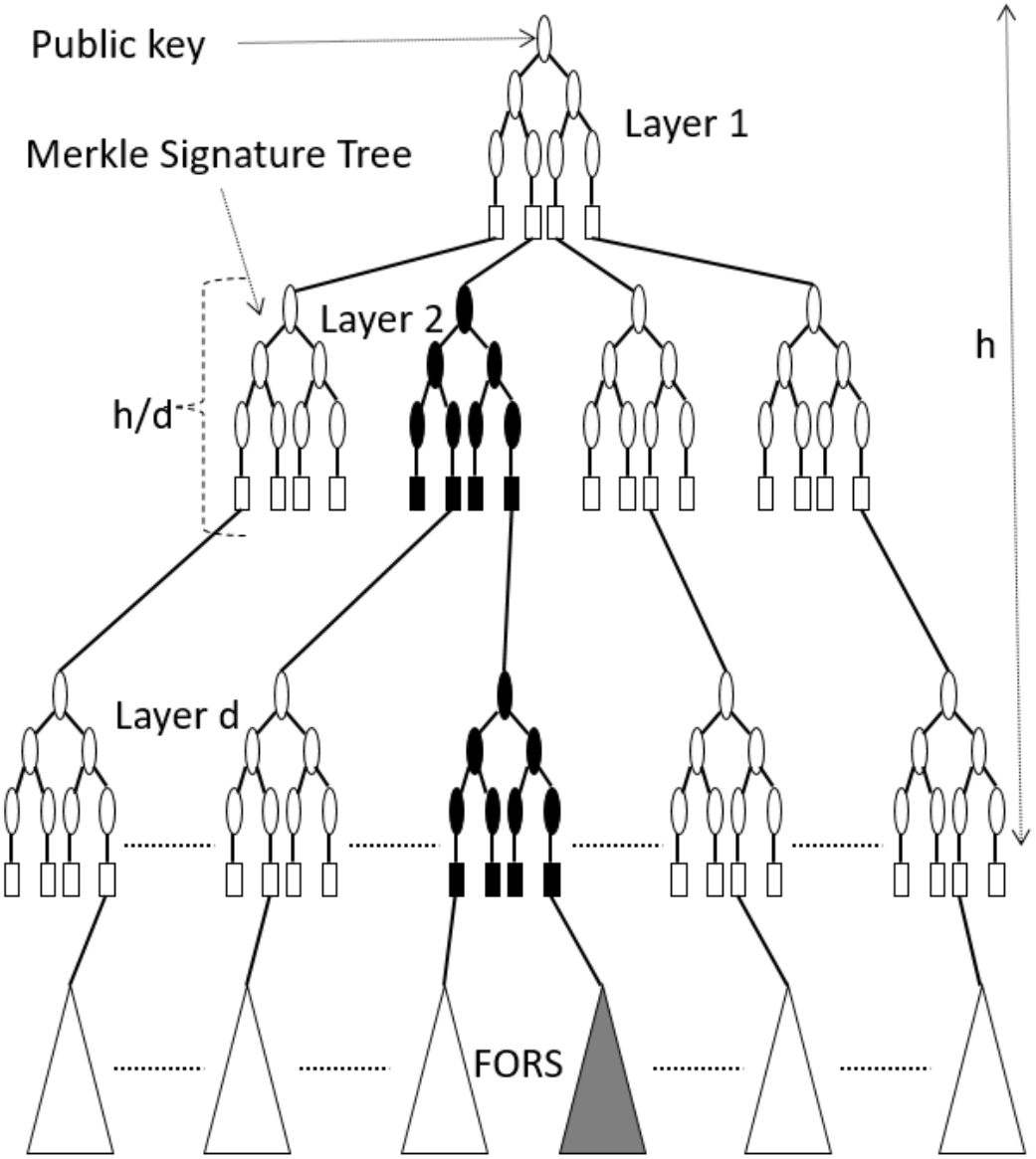


FIGURE 4.3: hyper-tree structure

traceability (see Definitions 5.1.5). After a high-level description of our group signature, this section presents SGS, the naive transformation of SPHINCS+ to a group signature. This will facilitate the understanding of T-SGS, the fully functional group signature. Both schemes are constructed with a cryptographic hash function H (see Definition 2.2.1), a Merkle signature scheme MS (see Definition 4.1.2), the FORS signature scheme FORS (see Definition 4.1.3), a digital signature scheme DS (see Definition 2.4.1), a PRF denoted by PRF (see Definition 2.6.1) and a tweakable hash function TH. For the construction of T-SGS, we also used a non-interactive zero-knowledge proof system NIZK and an accumulator A. All primitives are defined in Chapter 2 or previously in this chapter in Section 4.1. The detailed algorithms of both constructions are presented in Algorithm 4 and 6. All parameters are summarized in Table 4.1.

4.2.1 High level description

Both group signatures follow the SPHINCS+ hyper-tree structure presented in Section 4.1.1. However, to convert into a group signature, some changes are required. Similar to the SPHINCS+ signature scheme, an element of group public key gpk is the Merkle root of the top layer of the hyper-tree, denoted by r_{HT}. The other element of the public key is the manager $\mathcal{M}$'s public key mpk.

The group manager $\mathcal{M}$ starts the group by initializing all lists and his public and secret keys to communicate with the (prospective) group members. He also initializes two seeds $\mathsf{SK.shared}$, which is used to compute the trees of layer d to 2 of the hyper-tree and will be transmitted to the members after the completion of the joining process, and $\mathsf{SK.seed}$, which is used to generate the top layer (layer 1) of hyper-tree and so r_{HT}. It is important that $\mathsf{SK.seed}$ remains secret and is not shared with the group members. This avoids that a corrupted member using the OTS keys of layer 1 of the hyper-tree to sign different messages than the roots of layer 2 of the hyper-tree. According to Definition 2.5.1 on page 19, a OTS scheme is secure if and only if a OTS secret key is used only once, or in other words, is used to sign only one message. Therefore, if a corrupted member uses a OTS secret key belonging to layer 1 to sign a random message, then an outsider would be able to recover the OTS secret key. This would allow the outsider to generate a valid path to the hyper-tree root and so, compute a valid group signature without going through the joining process.

Each group member $\mathcal{U}$, owns a key pair $(\mathsf{upk}, \mathsf{usk})$. The member public key upk is composed of a DS public key, $\mathsf{DS.pk}$, which allows $\mathcal{M}$ to verify the origin of $\mathcal{U}$'s requests. While the secret key $\mathsf{usk} = (\mathsf{SK.prf}, \mathsf{SK.shared}, \mathsf{DS.sk}, \{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}, \mathsf{w})$ is composed of four elements for SGS and five for T-SGS. The first element is $\mathsf{SK.prf}$ that will allow the member to generate the path from the FORS signature to the Merkle root. The second element of the secret key is $\mathsf{SK.shared}$, which is a shared secret key among all group members, which allows the signer to dynamically generate the $d-1$ layers of Merkle tree needed for the signature, similar to SPHINCS+ (see Figure 4.3 and 4.4). Each new member receives $\mathsf{SK.shared}$ after the completion of the joining process. The third component of usk is the secret key of a DS scheme $\mathsf{DS.sk}$ which is used only to authenticate any requests sent to the central authority M. The last element is a witness w that will allow the member to generate the zero-knowledge membership proof and ensures traceability.

To sign a message, a group member computes a pseudorandom element y from a PRF and its personal secret $\mathsf{SK.prf}$. Then, it needs to compute the path with the help of the message and y, and so the path from the bottom to the top of the hyper-tree will differ from a member to another. Then, depending on the selected

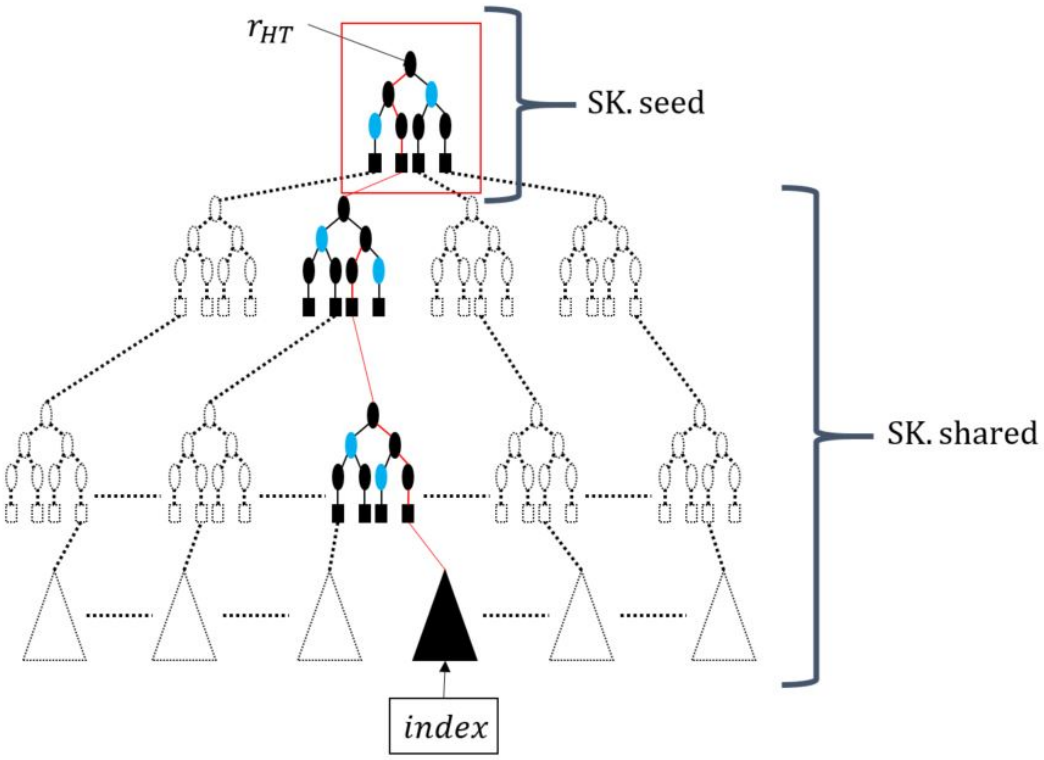


FIGURE 4.4: (T-)SGS hyper-tree

path (see Figure 4.4), it will generate the FORS signature with the help of the shared secret key SK.shared and then it will generate the internal nodes of the Merkle tree with the shared key SK.shared. A SGS group signature is composed of one FORS signature FORS.σ, and $d-1$ Merkle signatures MS.σ, one per level of the hyper-tree and the pseudorandom element y. The verification process recomputes FORS public key and recompute the hyper-tree root r'_1 thanks to the d Merkle signatures and accept if $\mathsf{r}_{HT} = r'_1$, because it means that the signer knows SK.shared and the Merkle signatures from layer 1 of Merkle tree, so the signature belongs to a group member, and it respected the path selected from y and the message.

4.2.2 SPHINCS group signature (SGS)

Our naive SGS construction (see Algorithm 4, 5 and 6) is a non-traceable group signature, which means that the manager cannot trace any signature. The blue parts in Algorithm 4, 5 and 6 are specific to SGS, which follows Definition 2.9.1.

SGS algorithms

SGS is defined by the following tuple of PPT algorithms:

- $(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{SGS.Setup}(1^\lambda)$: This algorithm takes as input the security parameter λ and initializes the group public key gpk composed of the root of top layer of the hyper-tree r_{HT}, (i.e. the tree in the red rectangular in Figure 4.4), the manager public key mpk. The group secret key gsk is composed of the manager secret key msk, the shared secret key SK.shared used to generate the hyper-tree from layers d to 2, the seed SK.seed to generate layer 1 of the

hyper-tree and a secret list of member secret key $priv\mathcal{L}$. Layer 1 of the hyper-tree is not generated with the shared secret key $\mathsf{SK.shared}$, because it avoids corrupted member to compute different OTS signature for the top layer and so break the security of the scheme. The parameters param, used to construct the hyper-tree (See Figure 4.3), and an empty list of members **members** are initialized.

- $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{SGS.MKeyGen}(1^\lambda)$: This algorithm generates the member's $\mathcal{U}$ key pair $(\mathsf{upk}, \mathsf{usk})$ depending on the security parameter λ. After its execution, the public key upk is composed of the public key of a digital signature scheme $\mathsf{DS.pk}$. The secret key usk is composed of $\mathsf{DS.sk}$. Both keys will be updated after the joining process.

- $(\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{msk}, \mathbf{members} \rangle_{\mathcal{M}}) \leftarrow \mathsf{SGS.Join}(\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}})$: This is an interactive process between a prospective group member $\mathcal{U}$ and the group manager $\mathcal{M}$. At the end of this procedure, the new group member receives the shared secret key $\mathsf{SK.shared}$, $2^{h/d}$ Merkle signatures $\{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}$, which are the Merkle signature of the roots of tree belonging to level 2 and its personal key $\mathsf{SK.prf}$ which are added to usk. $\mathsf{SK.prf}$ is generated by $\mathcal{M}$ during the process and shared with $\mathcal{U}$. $\mathcal{U}$ is added to the list **members** and the tuple $(\mathcal{U}, \mathsf{SK.prf})$ will be stored by $\mathcal{M}$ in the secret list $priv\mathcal{L}$.

- $\sigma \leftarrow \mathsf{SGS.Sign}(m, \mathsf{usk}, \mathsf{gpk}, \mathsf{param})$: This algorithm takes as input the message to be signed m, the member secret key usk, the group public key gpk and the public parameters param. The signing procedure starts by selecting the path to follow from the bottom to the top by computing $index \leftarrow \mathsf{TH}(\mathsf{gpk}, y, \mathsf{H}(m))$ where $y \leftarrow \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$[1]. The $index$ gives the position of the Merkle Tree at each layer and also which OTS key pairs to use to generate a Merkle signature for all of the d layers. The $index[i]$ indicates the tree number and the OTS key pair to use to generate the Merkle signature at level i of the hyper-tree. At each level, the signer needs to regenerate the corresponding Merkle tree. Figure 4.4 illustrates an example of a signature, only the trees on the path (e. i. the black ones) are generated during the signing process. At the end of the procedure, the signature σ is composed of an element y generated from the keyed PRF PRF, of one FORS is signature $\mathsf{FORS}.\sigma$ and d Merkle signature $\mathsf{MS}.\sigma$ (see Definition 4.1.2), each composed of a OTS signature and the authentication path (the blue nodes in Figure 4.4). The Merkle signature of layer d to 2 are generated thanks to the shared secret key $\mathsf{SK.shared}$, while the top layer is generated with the $2^{h/d}$ Merkle signatures given by $\mathcal{M}$ during the joining procedure.

- $0/1 \leftarrow \mathsf{SGS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param})$: This deterministic algorithm takes as inputs the signed message m, the SGS signature σ, the group public key gpk and the public parameters param and verifies the validity of a signature σ. It starts by computing the path that the signer had to follow ($index$) with the help of σ's element y and then computes the FORS public key from the FORS signature and accepts if the hyper-tree r_{HT} can be recomputed from σ following the path indicated by $index$ and the d Merkle signature.

- $(\mathsf{gpk}, \mathsf{gpk}, \mathbf{members}) \leftarrow \mathsf{SGS.Revoke}(\mathcal{U}, \mathsf{gsk}, \mathsf{gpk}, \mathbf{members}, \mathsf{param})$: This algorithm takes as input the member to be revoked $\mathcal{U}$, the group secret key gsk, the group public key gpk, the public list of members **members** and the public

[1] In SGS, y could be selected randomly

parameters param. This procedure deletes the member $\mathcal{U}$'s ability to sign. As in this construction, the traceability property is not provided. $\mathcal{M}$ needs to generate a new shared secret key $\mathsf{SK.shared}$ and a new seed $\mathsf{SK.seed}$. Then it regenerates the root of the hyper-tree r_{HT} which will result in an update of the group public and secret key gpk and gsk. $\mathsf{SK.shared}$ and the $2^{h/d}$ Merkle signature of layer 1 are shared to the remaining group member. $\mathcal{M}$ also updates **members** by removing $\mathcal{U}$ from it.

4.2.3 Traceable SPHINCS group signature (T-SGS)

We now present our non-trivial construction T-SGS which provides traceability and tracing soundness (see Definition 2.9.5 and 2.9.6). The red parts in Algorithm 4, 5 and 6 are specific to T-SGS.

From SGS to T-SGS

Our idea is still to exploit the promises of the SPHINCS+ digital signature [BHK+19]. However, to have a group signature respecting the most traditional definition of dynamic group signatures [BCC+16], we need to make more modifications to the SPHINCS+ signature. We need to integrate a zero-knowledge membership proof to the SGS. The signer proves that $y = \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$ and it has been generated by $\mathsf{SK.prf}$ belonging to a group member. Then, $\mathcal{M}$ can trace/open the signature by simply checking which $\mathsf{SK.prf}$ generated y. To achieve this, we use a non-interactive zero-knowledge proof system NIZK and an accumulator A which accumulates the set $priv\mathcal{L}$ containing all $\mathsf{SK.prf}$s of the group members.

Merkle accumulator and membership proof

As explained previously, in order to provide traceability, we need to ensure that y (lines 7 of the algorithm $\mathsf{T\text{-}SGS.Sign}$ presented in Algorithm 5) is the output of a PRF keyed with $\mathsf{SK.prf}$ belonging to one of the group members. Therefore we use an accumulator [BEF19] and a non-interactive zero-knowledge proof system (NIZK). Because of our desire to achieve post-quantum security using only symmetric primitives, we have the choice to work with ZKB++ [CDG+17] or KKW [KKW18]. Both these NIZKs rely on a one-way and collision resistant circuit C (i.e. a keyed PRF). The prover proves the knowledge of secret x such that $C(x) = y$ where y is public. For the sake of achieving better performances, we constructed the accumulator in the same way presented in [KKW18], which means that the accumulator has a Merkle tree structure (See Section 4.1.1) where each leaf corresponds to $\mathsf{H}(\mathsf{SK.prf})$ belonging to a member. There is a leaf for each group member. The signer has to prove with a NIZK proof system that it knows the path from his $\mathsf{SK.prf}$ to the Merkle root, which is the accumulator public key $\mathsf{A.pk}$ as illustrated in Figure 4.5. Therefore, the accumulator is a Merkle tree of depth $\log N$, where N is the number of members. The main difference between a "traditional" Merkle tree and a Merkle accumulator is the fact that we cannot perform a standard Merkle proof-like in the Merkle signature (see Section 4.1.1), because the path needs to be kept hidden to ensure anonymity of the members. The membership proof from the bottom to the top of the Merkle tree can be seen as a circuit $C(\mathsf{SK.prf}) = \mathsf{A.pk}$ and the signer shows that he knows $\mathsf{SK.prf}$ such that $C(\mathsf{SK.prf}) = \mathsf{A.pk}$, where $\mathsf{A.pk}$ is the Merkle root. In the context of T-SGS, Figure 4.5 illustrates the circuit for a group of N members where the signer proves the knowledge of $\mathsf{SK.prf}$ and the path (i.e. the witness $(w_1, w_2, \ldots, w_{\log n}) \in w_{\mathcal{U}}$) to the accumulator public key $\mathsf{A.pk}$ as illustrated in

TABLE 4.1: A summary of (T-)SGS parameters.

d	Number of internal layers in the hyper-tree
h	Total height of the hyper-tree
k, a	FORS parameters indicating in how many parts of length a bits the signed message m needs to be divided (see definition 4.1.3)
param	group parameters of the hyper-tree (d, h, k, a)
λ	post-quantum security parameter
$index$	Output of the the tweakable hash function indicating the Merkle tree number at each level. $index[i]$ indicate the selected tree at level i and $index[0]$ indicates the position of the FORS tree in the hyper-tree
members	List of group members
$\mathcal{R}$	List containing all SK.prf of revoked members
gpk	group public key which is composed of $\mathcal{M}$'s public key mpk,the root of the hyper-tree r_{HT}. T-SGS, additionally requires, the list of revoked members $\mathcal{R}$ and the accumulator's public key A.pk
(mpk, msk)	Manager's key pair used to authenticate the communication with group member
r_{HT}	Merkle root of the hyper-tree
gsk	Group secret key composed of $\mathcal{M}$'s secret key msk, the common secret key SK.shared, which is used to generate all trees from level 1 to $d-1$, SK.seed, which used to generated the top Merkle tree of the hyper-tree and the private list that stores members secret keys $priv\mathcal{L}$
usk	Member $\mathsf{id}_{\mathcal{U}}$ secret key composed SK.prf used to generate the path and the membership proof the path, SK.shared which is the share secret key, $2^{h/d}$ Merkle signatures $\{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}$, which correspond to the Merkle signature of layer 1, DS.sk which is used authenticate his request sent to $\mathcal{M}$ and the witness w used to performed the membership proof
upk	Member $\mathsf{id}_{\mathcal{U}}$ public key composed of a digital signature scheme' public key DS.pk
$priv\mathcal{L}$	Private list of $\mathcal{M}$ which stores the tuple $(\mathsf{id}_{\mathcal{U}}, \mathsf{SK.prf})$ for all group members. This list allows $\mathcal{M}$ to trace any signatures

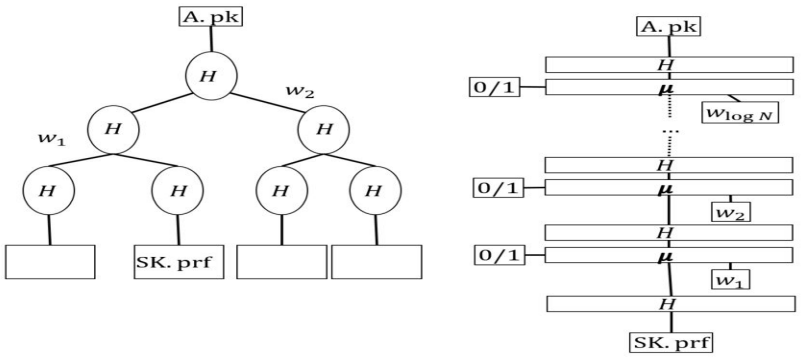


FIGURE 4.5: Accumulator example

Figure 4.5, where $C(x) = \mathsf{H}(\mu(\mathsf{H}(\mu(\mathsf{H}(\mathsf{SK.prf}), \mathsf{w}_1)), \mathsf{w}_2))$. H is a one-way and collision-resistant function and μ is a gate that orders the inputs of H depending if the path is coming from left or right in the tree. A more detailed explanation of the Merkle accumulator is done in Chapter 6.3.1.

T-SGS algorithms

T-SGS is defined by the following tuple of PPT algorithms:

- $(\mathsf{gpk}, \mathsf{gsk}, \mathbf{members}, \mathsf{param}) \leftarrow \mathsf{T\text{-}SGS.Setup}(1^\lambda)$: This algorithm follows the same procedure as $\mathsf{SGS.Setup}$ but additionally initializes the accumulator Acc and an initially empty list of revoked members $\mathcal{R}$ is added to gpk.

- $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{T\text{-}SGS.MKeyGen}(1^\lambda)$: This is similar to $\mathsf{SGS.MKeyGen}$.

- $(\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{msk}, \mathbf{members} \rangle_{\mathcal{M}}) \leftarrow \mathsf{T\text{-}SGS.Join}(\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}})$: This differs from $\mathsf{SGS.Join}$ by the fact that $\mathsf{SK.prf}$ is added to the accumulator Acc and the corresponding witness w will be transmited to $\mathcal{U}$.

- $\sigma \leftarrow \mathsf{T\text{-}SGS.Sign}(m, \mathsf{usk}, \mathsf{gpk}, \mathsf{upk}, \mathsf{param})$: This follows the same procedure as $\mathsf{SGS.Sign}$ but additionally a membership proof π will be generated where $\pi \leftarrow \mathsf{NIZK.Prove}((y, \mathsf{A.pk}), (\mathsf{SK.prf}, \mathsf{w}))$ (see Remark 1 on page 21), for the relation R such that $((y, \mathsf{A.pk}), (\mathsf{SK.prf}, \mathsf{w})) \in R$ if and only if the following statement holds: $y = \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$ and $\mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_{priv\mathcal{L}}, \mathsf{w}, \mathsf{SK.prf}) = 1$. After the generation of π, the path to follow in the hyper-tree based on the message m, gpk, y (as $\mathsf{SGS.Sign}$) and π. Moreover, this procedure starts by verifying the validity of the witness w. If w appears invalid, then the signer calls an internal interactive procedure $\langle \mathsf{usk} \rangle_{\mathcal{U}} \leftarrow \mathsf{T\text{-}SGS.WitUpdate}\langle \mathsf{upk}, \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{msk}, \mathsf{gpk}, \mathbf{members} \rangle_{\mathcal{M}})$. In this internal procedure, the signer requests a new witness from $\mathcal{M}$. This procedure needs to be called after new member joined the group. The rest of the procedure is similar to $\mathsf{SGS.Sign}$

- $0/1 \leftarrow \mathsf{T\text{-}SGS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param})$: This algorithm is similar to $\mathsf{SGS.Verify}$ with the exception that the membership proof π needs to be verified and it also checks if the element y of σ has not been generated by a secret key of a revoked member.

- $\mathcal{U}/\perp \leftarrow \mathsf{T\text{-}SGS.Open}(m, \sigma, \mathsf{gsk})$: This reveals the identity of the signer by the element $y \leftarrow \mathsf{PRF}(\mathsf{SK.prf}, H(m))$ of σ. $\mathcal{M}$ searches the corresponding $\mathsf{SK.prf}$ that generated y in his private list $priv\mathcal{L}$ and returns the signer's identity.

- $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{T\text{-}SGS.Revoke}(\mathcal{U}, \mathsf{gsk}, \mathsf{gpk}, \mathsf{param})$: This adds $\mathsf{SK.prf}$ belonging to $\mathcal{U}$ to the list of revoked members $\mathcal{R}$. Contrary to SGS, the T-SGS does not need to re-generate the shared secret parameters, thanks to the traceability property. Therefore, gsk remains unchanged and therefore, $\mathsf{SK.shared}$ and the Merkle signatures of layer 1 do not need to be shared, which makes this process more effective than $\mathsf{SGS.Revoke}$. It is important to notice that we choose to work with a public revocation list instead of the revocation through symmetric puncturable encryption (SPE) that we proposed in Chapter 3.3. The SPE-based revocation procedure is not adapted to the T-SGS construction as we do not have an element of the signature which belongs only to a members.

4.2.4 Security analysis

Theorem 7 (Correctness)**.** *Let* (T-)SGS *be the construction provided in Algorithm 4, 5 and 6 with a secure cryptographic hash function* H*, a Merkle signature scheme* MS*, a FORS signature scheme* FORS*, a* EU-CMA *secure digital signature scheme* DS*, a secure pseudorandom function* PRF*, a secure tweakable hash function* TH*, a secure Accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then* (T-)SGS *achieves correctness based on Definition 2.9.2.*

Correctness. The correctness of $(T\text{-})SGS$ ensues from the correctness of all symmetric primitives used. The Theorem 7 assumes that $(T\text{-})SGS$ uses only secure primitives, so primitives that are achieving correctness. Therefore, $(T\text{-})SGS$ achieves correctness. □

Theorem 8 (Non-frameability)**.** *Let* (T-)SGS *be the construction provided in Algorithm 4, 5 and 6 with a secure cryptographic hash function* H*, a Merkle signature scheme* MS*, a FORS signature scheme* FORS*, a* EU-CMA *secure digital signature scheme* DS*, a secure pseudorandom function* PRF*, a secure tweakable hash function* TH*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then* (T-)SGS *achieves non-frameability based on Definition 2.9.3.*

Non-Frameability. We use a game-based approach to show that both schemes SGS and T-SGS provide non-frameability. Let V_i be the event that $\mathcal{A}$ wins the GAME_i and elements denoted with $*$ is a forgery generated by $\mathcal{A}$. Because, SGS does not provide traceability, this proof shows that it is infeasible for outsiders to forge a signature. In both constructions, we assume that the central authority $\mathcal{M}$ is trusted.
GAME_1: The original non-frameability experiment $\mathbf{Exp}^{Fame}_{\mathcal{A},\mathsf{GS}}$ (Figure 2.3) executed with an adversary $\mathcal{A}$.
GAME_2: Same as the previous game but the adversary $\mathcal{A}$ replaces the signature of $\mathcal{M}$ with his forgery $\mathsf{DS}.\sigma^*$ to interact with honest group members. This is indistinguishable from the previous game because of $\mathcal{A}$ to win thanks to the EU-CMA security of DS (see definition 2.4.1) used during the interaction with $\mathcal{M}$. Indeed, $|\Pr[V_2] - \Pr[V_1]| \leq \mathbf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) < \mathsf{negl}(\lambda)$.
GAME_3: Same as GAME_2 but $\mathcal{A}$ creates his own OTS key pairs from the top layer of the hyper-tree. This advantage of $\mathcal{A}$ only decreased because of the collision resistance property of H. Indeed, H is a secure cryptographic hash function which makes it computationally infeasible to find a collision or to inverse the H according to Definition 2.2.1, therefore it is indistinguishable from the previous game: $|\Pr[V_3] - \Pr[V_2]| < \mathsf{negl}(\lambda)$.
GAME_4: Same as GAME_3 but $\mathcal{A}$ generates his own OTS secret keys from the Merkle signatures $\{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}$ of layer 1 of the hyper-tree that was received during the joining

Algorithm 4 (T-)SGS algorithms (part 1), for parameters see Table 4.1

(T-)SGS.Setup

Input: 1^λ
Output: gpk, gsk, **members**, param

1: param $\leftarrow (h, d, k, a)$
2: SK.shared, SK.seed $\xleftarrow{\$} \{0,1\}^{2\lambda}$
3: $(\mathsf{r}_{HT}, \{\mathsf{OTS.pk}_i, \mathsf{OTS.sk}_i\}_{i=1}^{2^{h/d}}) \leftarrow$ MS.Gen(SK.seed, 1, 1, h/d)
4: (mpk, msk) $\leftarrow$ DS.KeyGen(1^λ)
5: **members**, $priv\mathcal{L}$, $\mathcal{R} \leftarrow \emptyset$
6: A.pk $\leftarrow$ A.Gen(1^λ)
7: gpk $\leftarrow$ (mpk, r_{HT}, $\mathcal{R}$, A.pk)
8: gsk $\leftarrow$ (msk, SK.shared, SK.seed, $priv\mathcal{L}$)
9: **return** (gpk, gsk, **members**, param)

(T-)SGS.MKeyGen

Input: 1^λ
Output: upk, usk

1: (DS.pk, DS.sk) $\leftarrow$ DS.KeyGen(1^λ)
2: upk $\leftarrow$ (DS.pk)
3: usk $\leftarrow$ (DS.sk)
4: **return** (usk, upk)

SGS.Revoke

Input: $\mathcal{U}$, gsk, gpk, **members**, param
Output: gpk, gsk, **members**

1: Parse gsk $\rightarrow$ (msk, SK.shared, SK.seed, $priv\mathcal{L}$)
2: Parse gpk $\rightarrow$ (mpk, r_{HT}, $\mathcal{R}$)
3: Parse param $\rightarrow (h, d, k, a)$
4: Remove $\mathcal{U}$ from **members**
5: SK.shared, SK.seed $\xleftarrow{\$} \{0,1\}^{2\lambda}$
6: $(\mathsf{r}_{HT}, \{\mathsf{OTS.pk}_i, \mathsf{OTS.sk}_i\}_{i=1}^{h/d}) \leftarrow$ MS.Gen(SK.seed, 1, 1, h/d)
7: gpk $\leftarrow$ (mpk, r_{HT}, $\mathcal{R}$)
8: Sends the new SK.shared and the $2^{h/d}$ Merkle signature of layer 1 from the hyper-tree to all $id \in$ **members**
9: gsk $\leftarrow$ (msk, SK.shared, SK.seed, $priv\mathcal{L}$)
10: **return** (gpk, gsk, **members**)

T-SGS.Revoke

Input: $\mathcal{U}$, gsk, gpk, param
Output: gpk, gsk

1: Parse gsk $\rightarrow$ (msk, SK.shared, SK.seed, $priv\mathcal{L}$)
2: Parse gpk $\rightarrow$ (mpk, r_{HT}, $\mathcal{R}$)
3: Parse param $\rightarrow (h, d, k, a)$
4: Search $\mathcal{U}$ in $priv\mathcal{L}$ and return the corresponding SK.prf
5: $\mathcal{R} \leftarrow \mathcal{R} \cup$ SK.prf
6: gpk $\leftarrow$ (mpk, r_{HT}, $\mathcal{R}$)
7: **return** (gpk, gsk)

Algorithm 5 (T-)SGS algorithm (part 2), for parameters see Table 4.1

(T-)SGS.Sign

Input: m, usk, upk, gpk, param
Output: σ

Parse usk $\rightarrow$ (SK.prf, SK.shared, DS.sk, $\{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}$, w)
Parse upk $\rightarrow$ (DS.SK.prf)
Parse gpk $\rightarrow$ (mpk, r_{HT}, $\mathcal{R}$, A.pk)
Parse param $\rightarrow$ (h, d, k, a)
if A.Verify(A.pk, $\mathsf{A}_{\mathsf{priv}\mathcal{L}}$, w, SK.prf) $= 0$ **then**
  $\langle \mathsf{usk} \rangle_{\mathcal{U}} \leftarrow$ T-SGS.WitUpdate($\langle \mathsf{upk}, \mathsf{usk}, \rangle_{\mathcal{U}}, \langle \mathsf{msk}, \mathsf{gpk}, \mathbf{members} \rangle_{\mathcal{M}}$) ▷see Algorithm 6
end if
$y \leftarrow \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$ ▷refer to footnote 1 on page 60
$\pi \leftarrow$ NIZK.Prove($(y, \mathsf{A.pk}), (\mathsf{SK.prf}, \mathsf{w})$) ▷see Remark 1 on page 21
Prove of knowledge of SK.prf and $((y, \mathsf{A.pk}), (\mathsf{SK.prf}, \mathsf{w})) \in R$ iff:

(i) $y = \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$

(ii) $1 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_{\mathsf{priv}\mathcal{L}}, \mathsf{w}, \mathsf{SK.prf})$

$index \leftarrow \mathsf{TH}(\mathsf{gpk}, (y, \pi), m)$
$index \leftarrow \mathsf{TH}(\mathsf{gpk}, y, m)$
$(\mathsf{FORS.pk}, \mathsf{FORS.sk}) \leftarrow \mathsf{FORS.KeyGen}(index, \mathsf{SK.shared})$
$\mathsf{FORS}.\sigma \leftarrow \mathsf{FORS.Sign}(\mathsf{H}(m), \mathsf{FORS.sk})$ ▷see Definition 4.1.3
$(r_d, \{(\mathsf{OTS.pk}_i, \mathsf{OTS.sk}_i)\}_{i=1}^{2^{h/d}}) \leftarrow \mathsf{MS.Gen}(\mathsf{SK.shared}, index[d], d, h/d)$
$\mathsf{MS}.\sigma_d \leftarrow \mathsf{MS.Sign}(\mathsf{FORS.pk}, \mathsf{OTS.sk}_{index[d]}, d)$
for $j = d - 1$ to 2 **do**
  $(r_j, \{(\mathsf{OTS.pk}_i, \mathsf{OTS.sk}_i)\}_{i=1}^{2^{h/d}}) \leftarrow \mathsf{MS.Gen}(\mathsf{SK.shared}, index[j], j, h/d)$
  $\mathsf{MS}.\sigma_j \leftarrow \mathsf{MS.Sign}(r_{j+1}, \mathsf{OTS.sk}_{index[j]}, j)$
end for
$\sigma \leftarrow (y, \pi, \mathsf{FORS}.\sigma, \{\mathsf{MS}.\sigma_i\}_{i=1}^{d})$
return σ

(T-)SGS.Verify

Input: m, σ, gpk, param
Output: $0/1$

Parse $\sigma \rightarrow (y, \pi, \mathsf{FORS}.\sigma, \{\mathsf{MS}.\sigma_i\}_{i=1}^{d})$
Parse gpk $\rightarrow$ (mpk, r_{HT}, $\mathcal{R}$, A.pk)
Parse param $\rightarrow$ (h, d, k, a)
if $0 =$ NIZK.Verify($(y, \mathsf{A.pk}), \pi$) **then**
  return 0
end if
for all SK.prf $\in \mathcal{R}$ **do**
  if $y = \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$ **then**
    return 0
  end if
end for
$index \leftarrow \mathsf{TH}(\mathsf{gpk}, (y, \pi), m)$
$index \leftarrow \mathsf{TH}(\mathsf{gpk}, y, m)$
$\mathsf{FORS.pk}' \leftarrow \mathsf{FORS.Verify}(\mathsf{H}(m), \mathsf{FORS}.\sigma)$
$r_d \leftarrow \mathsf{MS.Verify}(\mathsf{FORS.pk}', \mathsf{MS}.\sigma_d, index[d])$
for $i = d - 1$ to 1 **do**
  $r_i \leftarrow \mathsf{MS.Verify}(r_{i+1}, \mathsf{MS}.\sigma_i, index[i])$
end for
return $r_1 == \mathsf{r}_{HT}$

Algorithm 6 (T-)SGS Algorithm (part 3), for parameters see Table 4.1

(T-)SGS.Join

Input: $\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}}$
Output: $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}}$

1: $\mathcal{U}$: Parse $\mathsf{usk} \rightarrow (\mathsf{DS.sk})$
2: $\mathcal{U}$: Parse $\mathsf{upk} \rightarrow (\mathsf{DS.pk})$
3: $\mathcal{M}$: Parse $\mathsf{gpk} \rightarrow (\mathsf{mpk}, \mathsf{r}_{HT}, \mathcal{R}, \mathsf{A.pk})$
4: $\mathcal{M}$: $\mathsf{gsk} \rightarrow (\mathsf{msk}, \mathsf{SK.shared}, \mathsf{SK.seed}, privL)$
5: $\mathcal{U}$: $\mathsf{DS}.\sigma \leftarrow \mathsf{DS.Sign}(join, \mathsf{DS.sk})$
6: $\mathcal{U}$ sends $(\mathsf{DS}.\sigma, join, \mathsf{upk})$ to $\mathcal{M}$
7: $\mathcal{M}$: if $1 = \mathsf{DS.Verify}(join, \mathsf{DS}.\sigma, \mathsf{DS.pk})$ then $\mathbf{members} \leftarrow \mathbf{members} \cup \mathcal{U}$, $\mathsf{SK.prf} \xleftarrow{\$} \{0,1\}^{2\lambda}, privL \leftarrow privL \cup (\mathcal{U}, \mathsf{SK.prf})$
8: $\mathcal{M}$: $\mathsf{gsk} \leftarrow (\mathsf{msk}, \mathsf{SK.shared}, privL)$
9: $\mathcal{M}$: $(\mathsf{A}_{privL}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, privL)$
10: $\mathcal{M}$: $\mathsf{w} \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_{privL}, privL, \mathsf{SK.prf})$
11: $\mathcal{M}$: $sk \leftarrow (\mathsf{SK.shared}, \mathsf{SK.prf}, \{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}, \mathsf{w})$
12: $\mathcal{M}$: $\mathsf{DS}.\sigma \leftarrow \mathsf{DS.Sign}(sk, \mathsf{msk})$
13: $\mathcal{M}$ sends $(\mathsf{DS}.\sigma, sk)$ to $\mathcal{U}$.
14: $\mathcal{U}$: if $1 = \mathsf{DS.Verify}(sk, \mathsf{DS}.\sigma, \mathsf{mpk})$ then Parse $sk \rightarrow (\mathsf{SK.shared}, \mathsf{SK.prf}, \{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}, \mathsf{w})$, $\mathsf{usk} \leftarrow (\mathsf{SK.prf}, \mathsf{SK.shared}, \mathsf{DS.sk}, \{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}, \mathsf{w})$,
15: **return** $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk}, \mathbf{members} \rangle_{\mathcal{M}}$

T-SGS.Open

Input: m, σ, gsk
Output: $\mathcal{U} / \perp$

1: Parse $\sigma \rightarrow (y, \pi, \mathsf{FORS}.\sigma, \{\mathsf{MS}.\sigma_i\}_{i=1}^{d})$
2: Parse $\mathsf{gsk} \rightarrow (\mathsf{msk}, \mathsf{SK.shared}, \mathsf{SK.seed}, privL)$
3: Return the set of all $(\mathcal{U}, \mathsf{SK.prf}) \in privL$ such that $y = \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$ or $\perp$
4: **return** $\mathcal{U} / \perp$

T-SGS.WitUpdate

Input: $\langle \mathsf{upk}, \mathsf{usk}, \rangle_{\mathcal{U}}, \langle \mathsf{msk}, \mathsf{gpk}, \mathbf{members} \rangle_{\mathcal{M}}$
Output: $\langle \mathsf{usk} \rangle_{\mathcal{U}}$

1: $\mathcal{U}$: Parse $\mathsf{upk} \rightarrow (\mathsf{DS.pk})$
2: $\mathcal{U}$: Parse $\mathsf{usk} \rightarrow (\mathsf{SK.prf}, \mathsf{DS.sk}, \mathsf{SK.shared}, \{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}, \mathsf{w})$
3: $\mathcal{M}$: Parse $\mathsf{gpk} \rightarrow (\mathsf{mpk}, \mathsf{r}_{HT}, \mathcal{R}, \mathsf{A.pk})$
4: $\mathcal{U}$: $\mathsf{DS}.\sigma \leftarrow \mathsf{DS.Sign}(wit, \mathsf{DS.sk})$
5: $\mathcal{U}$ sends $(\mathsf{DS}.\sigma, wit, \mathsf{DS.pk})$ to $\mathcal{M}$
6: $\mathcal{M}$: if $1 = \mathsf{DS.Verify}(wit, \mathsf{DS}.\sigma, \mathsf{DS.pk})$ if it outputs 1, $\mathcal{M}$ searches the tuple $(\mathcal{U}, \mathsf{SK.prf}) \in privL$ and runs $\mathsf{w} \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_{privL}, privL, \mathsf{SK.prf})$, abort otherwise
7: $\mathcal{M}$: $\mathsf{DS}.\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{w}, \mathsf{msk})$
8: $\mathcal{M}$ sends $(\mathsf{w}, \mathsf{DS}.\sigma)$ to $\mathcal{U}$
9: $\mathcal{U}$: if $1 = \mathsf{DS.Verify}(\mathsf{w}, \mathsf{DS}.\sigma, \mathsf{mpk})$ then $\mathsf{usk} \leftarrow (\mathsf{SK.prf}, \mathsf{DS.sk}, \mathsf{SK.shared}, \{\mathsf{MS}.\sigma_i\}_{i=1}^{2^{h/d}}, \mathsf{w})$
10: **return** $\langle \mathsf{usk} \rangle_{\mathcal{U}}$

process by a corrupted member. This change only reduces the probability of winning for $\mathcal{A}$ because of the EU-CMA security of OTS. Therefore, it is indistinguishable from the previous game and we have $|\Pr[V_4] - \Pr[V_3]| \leq \mathbf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{OTS}}(\lambda) < \mathsf{negl}(\lambda)$.
GAME_5: Same as GAME_4 but $\mathcal{A}$ generates his own $index^*$ for an honest user. The advantage of the adversary is only reducing thanks to the security of TH because based on the security definition of a tweakable hash function (see Definition 4.1.1) which shares the same properties as a cryptographic hash function, therefore $|\Pr[V_5] - \Pr[V_4]| < \mathsf{negl}(\lambda)$. This ends the proof of non-freamability of SGS
GAME_5: Same as GAME_4, but $\mathcal{A}$ tries to forge the membership proof π^*. $\mathcal{A}$ can win only if he extracts the witness $w^* = (\mathsf{SK.prf}^*, \mathsf{w}^*)$ for $\mathsf{SK.prf}^* \notin priv\mathcal{L}$, so for $\mathsf{SK.prf}^*$ which has not been accumulated in A. This only reduce the advantage of $\mathcal{A}$ because security properties of NIZK (see Definition 2.8.1), this has only a negligible probability to happen. This means that $|\Pr[V_5] - \Pr[V_4]| \leq \mathsf{negl}(\lambda)$.
GAME_6: Same as GAME_6, but $\mathcal{A}$ replaces y in the signature σ by a random element y^*. Thanks to the security of PRF, this is only decreasing the probability of $\mathcal{A}$ of winning and therefore $|\Pr[V_6] - \Pr[V_5]| \leq \mathsf{Adv}^{PRF}_{\mathcal{A},\mathsf{PRF}}(\lambda) < \mathsf{negl}(\lambda)$
GAME_7: $\mathcal{A}$ wins this game if the signature σ^* generated by $\mathcal{A}$ contain a valid evaluation of y for a group member, but this will happen with a negligible probability thanks to the unpredictability of PRF on new inputs as the input is $\mathsf{H}(m)$, H is a cryptographic hash function and so, provides collision resistance and therefore the input will always change.Therefore, we can conclude that $|\Pr[V_7] - \Pr[V_6]| \leq \mathsf{negl}(\lambda)$

This shows that $\mathbf{Adv}^{Frame}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

□

Theorem 9 (Anonymity)**.** *Let* (T-)SGS *be the construction provided in Algorithm 4, 5 and 6 with a secure cryptographic hash function* H*, a Merkle signature scheme* MS*, a FORS signature scheme* FORS*, a* EU-CMA *secure digital signature scheme* DS*, a secure pseudorandom function* PRF*, a secure tweakable hash function* TH*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then* (T-)SGS *achieves anonymity based on Definition 2.9.4.*

Anonymity. To proof, the anonymity of (T-)SGS, we will use the game base approach. In the proof, V_i is the event that $\mathcal{A}$ wins the anonymity experiment GAME_i. The elements denoted with $*$ are the forgeries generated by $\mathcal{A}$. Elements of the proof in red, are valid only for T-SGS.
GAME_1: The real anonymity experiment $\mathbf{Exp}^{Anon-b}_{\mathcal{A},\mathsf{GS}}$ (see Figure 2.3) runs with an adversary $\mathcal{A}$.
GAME_2: Same as GAME_1 but for $\mathsf{id}^0_{\mathcal{U}}$ the value $y = PRF(\mathsf{SK.prf}_0, \mathsf{H}(m))$ is replaced by a random string. This is reduce the advantage of $\mathcal{A}$ because of the security of PRF (see Definition 2.6.1), which reduces the chance of the adversary to win. Indeed, to win this game, $\mathcal{A}$ needs to breaks PRF, and this has only a negligible chance to happen. Therefore, we have $|\Pr[V_2] - \Pr[V_1]| \leq \mathsf{Adv}^{PRF}_{\mathcal{A},\mathsf{PRF}}(\lambda) < \mathsf{negl}(\lambda)$.
GAME_3: Same as GAME_2 but for $\mathsf{id}^1_{\mathcal{U}}$ the value $y = \mathsf{PRF}(\mathsf{SK.prf}_1, \mathsf{H}(m))$ is replaced by a random string. The advantage for $\mathcal{A}$ is decreasing with this change because of the security of PRF (see Definition 2.6.1). After calling the oracle $Chal_b$, it will output a signature σ, where the signature is coming from $\mathsf{id}^0_{\mathcal{U}}$ or $\mathsf{id}^1_{\mathcal{U}}$. For an adversary to win GAME_3, $\mathcal{A}$ needs to break the security of PRF. Therefore $|\Pr[V_3] - \Pr[V_2]| \leq \mathsf{Adv}^{PRF}_{\mathcal{A},\mathsf{PRF}}(\lambda) < \mathsf{negl}(\lambda)$.
GAME_4: Same as GAME_3 but $\mathcal{A}$ tries to breaks the anonymity through the membership proof π. But thank to the zero-knowledge property of NIZK (see Definition 2.8.1 and 2.8.4) this reduces the advantage of $\mathcal{A}$ and therefore, we have $|\Pr[V_4] - \Pr[V_3]| \leq \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) < \mathsf{negl}(\lambda)$.

This shows that $\mathbf{Adv}_{\mathcal{A}}^{Anon}(\lambda) < \mathsf{negl}(\lambda)$. □

Theorem 10 (Traceability). *Let* T-SGS *be the construction provided in Algorithm 4, 5 and 6 with a secure cryptographic hash function* H*, a Merkle signature scheme* MS*, a FORS signature scheme* FORS*, a* EU-CMA *secure digital signature scheme* DS*, a secure pseudorandom function* PRF*, a secure tweakable hash function* TH*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then* T-SGS *achieves traceability based on Definition 2.9.5.*

Traceability. To prove the traceability of T-SGS, we will use again a game-based approach. In the proof, V_i is the event that the adversary $\mathcal{A}$ the traceability experiment $\mathbf{Exp}_{\mathcal{A},\mathsf{GS}}^{Trace}$ presented in Figure 2.3. The elements denoted with $*$ are forgeries generated by $\mathcal{A}$.
GAME_1: The real traceability experiment $\mathbf{Exp}_{\mathcal{A},\mathsf{GS}}^{Trace}$ (see Figure 2.3) runs with an adversary $\mathcal{A}$.
GAME_2: Same as GAME_1 but A uses his own $\mathsf{SK.prf}^*$ to generate y. This change only decreases the advantages of $\mathcal{A}$ because of the security of PRF (see Definition 2.6.1). The only chance for A to win the experiment is to generate $y^* \leftarrow \mathsf{PRF}(\mathsf{SK.prf}^*, \mathsf{H}(m))$ which is equal to the original $y \leftarrow \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$. This is computationally infeasible due to collision-resistance and the one-wayness provided by PRF. Therefore, this game is is indistinguishable from the previous game: $|\Pr[V_2] - \Pr[V_1]| \leq \mathsf{negl}(\lambda)$.
GAME_3: Same as GAME_2 but $\mathcal{A}$ forges the membership proof related to the path y. In another words, $\mathcal{A}$ generates a proof when $((y, \mathsf{A.pk}), (\mathsf{SK.prf}^*, \mathsf{w})) \notin R$. By achieving this forgery, it ensues that $\mathsf{T\text{-}SGS.Open}(m, \sigma, \mathsf{gsk}) = \perp$ but $\mathsf{T\text{-}SGS.Verify}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{param}) = 1$. The probability of success for $\mathcal{A}$ is negligible because of the soundness property of the NIZK used in our scheme (see Definition 2.8.3). Therefore, it is indistinguishable from the previous game $|\Pr[V_3] - \Pr[V_2]| \leq \mathsf{Adv}_{\mathcal{A},\mathsf{NIZK}}^{\mathsf{Sound}}(\lambda) < \mathsf{negl}(\lambda)$.
This concludes the proof and shows that $\mathbf{Adv}_{\mathcal{A}}^{Trace}(\lambda) < \mathsf{negl}(\lambda)$. □

Theorem 11 (Tracing soundness). *Let* T-SGS *be the construction provided in Algorithm 4, 5 and 6 with a secure cryptographic hash function* H*, a Merkle signature scheme* MS*, a FORS signature scheme* FORS*, a* EU-CMA *secure digital signature scheme* DS*, a secure pseudorandom function* PRF*, a secure tweakable hash function* TH*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then* T-SGS *achieves tracing soundness based on Definition 2.9.6.*

Tracing soundness. To prove, the tracing soundness of T-SGS, we will use a game-based approach again. In the proof, V_i is the event that the adversary $\mathcal{A}$ the traceability experiment $\mathbf{Exp}_{\mathcal{A},\mathsf{GS}}^{Trace\text{-}Sound}$ presented in Figure 2.3. The elements denoted with $*$ be forgery generated by $\mathcal{A}$. $\mathcal{A}$ wins the game if a signature can be assigned to two different honest group members. We assume that $\mathsf{SK.prf}_0$ is an element of usk of the member $\mathsf{id}_{\mathcal{U}}^0$ and $\mathsf{SK.prf}_1$ is an element of the secret usk belonging to the honest member $\mathsf{id}_{\mathcal{U}}^1$.
GAME_1: The original tracing soundness experiment $\mathbf{Exp}_{\mathcal{A},\mathsf{GS}}^{Trace\text{-}Sound}$ (see Figure 2.3) executed with an adversary $\mathcal{A}$.
GAME_2: Same as GAME_1 but A uses his own $\mathsf{SK.prf}^*$ to generate y. This only decreases the advantage of $\mathcal{A}$ thanks to the security of PRF (see Section 2.6.1). The only chance for A to win the experiment is to generate y^* such that $\mathsf{PRF}(\mathsf{SK.prf}_1, \mathsf{H}(m)) = y^* = \mathsf{PRF}(\mathsf{SK.prf}_2, \mathsf{H}(m))$. The collision-resistance and one-wayness provided by

PRF implies that the wining probability is indistinguishable from the previous game: $|\Pr[V_2] - \Pr[V_1]| < \mathsf{negl}(\lambda)$.

This shows that $\mathbf{Adv}_{\mathcal{A}}^{Trace-Sound}(\lambda) < \mathsf{negl}(\lambda)$. □

Theorem 12 (Revocability)**.** *Let* T-SGS *be the construction provided in Algorithm 4, 5 and 6 with a secure cryptographic hash function* H*, a Merkle signature scheme* MS*, a FORS signature scheme* FORS*, a* EU-CMA *secure digital signature scheme* DS*, a secure pseudorandom function* PRF*, a secure tweakable hash function* TH*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then* (T-)SGS *achieves revocability based on Definition 2.9.7.*

Revocability. This prove shows that it is computationally infeasible for a revoked member to generate a valid signature. The part in blue are valid for SGS and the part in red are valid for T-SGS. We assume that $\mathsf{id}_{\mathcal{U}}$ is a revoked member and therefore the element SK.prf of his secret key usk is in the list $\mathcal{R}$. Let $\sigma = (y, \pi, \mathsf{FORS}.\sigma, \{\mathsf{MS}.\sigma\}_{i=1}^{d})$ be the output of the algorithm (T-)SGS.Sign$(m, \mathsf{usk}, \mathsf{upk}, \mathsf{gpk}, \mathsf{param})$.

The proof of revocability for SGS is similar to the proof of non-freamability for outsiders, because $\mathsf{id}_{\mathcal{U}}$ cannot access to the current shared secret key SK.shared, it is computationally infeasible for $\mathsf{id}_{\mathcal{U}}$ to generate a valid path to the hyper-tree root r_{HT}. Therefore, the value r_1 outputted after the d verification of the Merkle signature will not be equal to r_{HT} thanks to the collision resistance property of H. Therefore, the signature will be rejected.

If $\mathsf{id}_{\mathcal{U}}$ tries to generate a signature following the T-SGS.Sign algorithm, the signature will be rejected because of line 8 of the T-SGS.Verify algorithm (see Algorithm 5), as $y \leftarrow \mathsf{PRF}(\mathsf{SK.prf}, \mathsf{H}(m))$ and $\mathsf{usk} \in \mathcal{R}$. To avoid the rejection when the verification procedure checks if $\mathsf{SK.prf} \in \mathcal{R}$, $\mathsf{id}_{\mathcal{U}}$ needs to generate $y \leftarrow \mathsf{PRF}(\mathsf{SK.prf}', \mathsf{H}(m))$ where $\mathsf{SK.prf}' \notin \mathcal{R}$, this verification process will still abort because of the soundness property of NIZK. Indeed, $\mathsf{id}_{\mathcal{U}}$ needs to prove that he knows $\mathsf{SK.prf}'$ which has generated y and has been accumulated in A, which is not the case. Therefore, we can conclude that $\Pr[\sigma \leftarrow (\text{T-})\mathsf{SGS.Sign}(m, \mathsf{usk}, \mathsf{upk}, \mathsf{gpk}, \mathsf{param}) : (\text{T-})\mathsf{SGS.Verify}(\sigma, \mathsf{gpk}, \mathsf{param}) = 1] < \mathsf{negl}(\lambda)$.

And therefore it shows that $\mathbf{Adv}_{\mathcal{A}}^{Revoke}(\lambda) < \mathsf{negl}(\lambda)$.

□

4.3 Evaluation

In this section, we evaluate the expected performances of both SGS and multiple instances of T-SGS. We first explain the different setup that we use to perform the evaluation and then we discuss the (T-)SGS performance and compare it with the state-of-art at the time of its design.

4.3.1 Evaluation setup

We evaluate the applicability of both constructions for 128-bits of post-quantum security. Our simulation is based on the C implementation provided in [BHK+] for the SPHINCS+ part of our protocol. The SPHINCS+ signature relies the OTS scheme is instantiated with W-OTS+ [Hül13] and FORS as described in Section 4.1.1. The PRF has been constructed with the block cipher LowMC [ARS+15] and we use SHA-256 for all cryptographic hash purposes.

For the traceable version, T-SGS, we have the choice between two zero-knowledge (ZK) proofs ZKB++ [CDG+17] and KKW [KKW18], which can be made non-interactive by two different transforms: the Fiat-Shamir transform or the Unruh transform [Unr15]. ZKB++ and KKW have been implemented from the Picnic digital signature C implementation [Pic]. We constructed an accumulator and then simulated a ZK proof of membership with Picnic implementation and we added it to the SPHINCS+ implementation to have a better understanding of our algorithms' performances. T-SGS has been primarily evaluated with KKW as a ZK proof system made non-interactive with the Fiat-Shamir transform in our goal to optimize the signature size. We also evaluate T-SGS with ZKB++ as ZK proof once with the Fiat-Shamir transform (T-SGS-ZKB++) and once with the Unruh transform (T-SGS-Unruh++). The simulations have been done on a machine with a processor Intel(R) Core i7-7500U CPU @ 2.7GHz 2.90GHz and the results are summarized in Table 4.2 and 4.3.

TABLE 4.2: SGS evaluation: SGS.Sign and SGS.Verify performance are presented in seconds, with $d = 9, h = 63, k = 29$. The signature size $|\sigma|$ is presented in KB

SGS		
SGS.Sign	SGS.Verify	$\|\sigma\|$
1.244	0.0016	33

TABLE 4.3: T-SGS evaluation: Sign and Verify performance are presented in seconds, with $d = 9, h = 63, k = 29$. The signature size $|\sigma|$ is presented in KB

	T-SGS			T-SGS-ZKB++			T-SGS-Unruh++		
Members	.Sign	.Verify	$\|\sigma\|$	.Sign	.Verify	$\|\sigma\|$	.Sign	.Verify	$\|\sigma\|$
2^{10}	7.383	3.886	501	3.526	1.546	1287	3.592	1.880	2076
2^{12}	8.114	4.544	595	3.910	1.744	1538	4.000	2.000	2485
2^{14}	9.025	5.104	688	4.009	2.004	1789	4.161	2.302	2895
2^{16}	10.001	5.709	783	4.149	2.23	2040	4.568	2.587	3303
2^{18}	11.042	6.428	876	4.518	2.502	2291	4.993	2.868	3713
2^{20}	12.091	7.093	970	4.883	2.874	2543	5.405	3.272	4122

4.3.2 Evaluation results

(T-)SGS signature size can be represented as follow: $|\sigma| = h \cdot |\mathsf{Hash}| + d \cdot |\mathsf{MS}.\sigma| + |y| + |\mathsf{FORS}.\sigma| + |\pi|$, where $|\mathsf{Hash}|$ is the size of the output of the cryptographic hash function H, h the height of the hyper-tree, $|\mathsf{MS}.\sigma|$ is the size of a Merkle signature, $|\mathsf{OTS}.\sigma|$ is the size of the OTS signature, $|\mathsf{FORS}.\sigma|$ is the size of the FORS signature scheme and $|y|$ is the size of the output of the PRF function which are all elements of the SGS signature which has the advantage of not depending on the number of group members. The T-SGS signature has additionally the membership proof π of size $|\pi|$, which increase logarithmically with the ring size on $\mathcal{O}(\log N)$, where N is the number of group members, because of the accumulator that has a Merkle tree structure (see Figure 4.5). The signature size of SGS is shown in Table 4.2 and 4.4. Similar to SGS, the scheme presented by Boneh et al. [BEF19] does not provide traceability. Table 4.4 shows that SGS has two significant advantages compared to [BEF19]. The main one is that the group size does not affect the size of the signature because the signature size does only depend on the parameters of the hyper-tree. Moreover, the practical size achieved by SGS is significantly shorter than the minimum one that can be achieved by [BEF19].

Table 4.3 and 4.4 demonstrates the practical size achieved by the several instances of our traceable group signature (T-SGS) for different group size, while Table 4.4

shows the best theoretical signature size for a post-quantum security level 128-bits. For T-SGS, Table 4.3 highlights that if the objective is to optimize the signature size, KKW is the best option as ZK proof coupled with the Fiat-Shamir transform. The sizes were computed based on the formula available in [KKW18] with the parameters suggested by the Picnic team in [Pic]. However, if the goal is algorithm efficiency, T-SGS implemented with ZKB++ and the Fiat-Shamir transform is the best alternative. Finally, suppose the main objective is to provide the highest level of post-quantum security. In that case, we need to use the Unruh transform [Unr15], which is provably post-quantum secure but this comes with an increase in signature size. In terms of signature size, the work of Katz et. al. [KKW18] and our T-SGS are quite close, but we additionally provide full dynamicity (see Table 4.4) and the difference of signature size is mainly due to the choice of parameters of KWW. Their works implemented the parameters to optimize the signature size, while we choose to work with the KKW parameters submitted at NIST post-quantum standardization process [KZ20],[AASA+20].

When it comes to dynamic group signatures, DGM [BLS+19] appears to be competitive in terms of signature size (See Table 4.4). However, it has two main limitations when it comes to implementation. It is stateful, while (T-)SGS remains stateless and furthermore, DGM's verification process is interactive while our scheme is non-interactive. Table 4.4 and 4.3 demonstrate that T-SGS achieves better signature size than the other dynamic stateless group signature [BEF19] and additionally provides traceability which allow the manager to revoke corrupted members, while in [BEF19] the manager can only revoke members that have their key leaked.

A disadvantage of (T-)SGS is also the large secret key size usk, which is composed of a minimum of 275KB as illustrated in Figure 4.4. This large size comes from the requirement for each group member to store $2^{h/d}$ Merkle signatures. These Merkle signature correspond to layer 1 of the hyper-tree (see Figure 4.4).

4.3.3 Cost of traceability

This subsection discusses the impact of the integration of an accumulator in the hyper-tree. As explained in Section 4.2.3, the goal of the accumulator is to ensure the tracebility property and so transform SGS to a group signature, T-SGS, which has all the required properties. We studied other technique to provide this traceability property, but it appears that the use of an accumulator is the only possibility to achieve this aim. We require to ensure two statements: first, that the element y (the element to generate the path) has been generated by a secret key $\mathsf{SK.prf}$ and second that this secret key belongs to one of the group member. To best of our knowledge, this second statement cannot be done without requiring the use of an accumulator while working only with symmetric primitives. We investigate other approach including the use of verifiable random function (VRF), but we could not achieve the traceability without compromising the anonymity of the signer.

The problem coming from the integration of the accumulator is that path verification procedure becomes redundant because giving a zero-knowledge proof π that demonstrates that y was computed by a secret key belonging to a group member. This observation forces us the abandon the approach to work with SPHINCS+ in order to design privacy-preserving primitives.

4.4 Chapter conclusion

This paper presented two dynamic post-quantum group signatures constructed with symmetric primitives only. The first one (SGS) is a naive transformation of SPHINCS+ [BHK+19]. The second one, named T-SGS, is a non-trivial extension of SGS to a fully functional group signature. When it comes to post-quantum group signatures based on symmetric primitives, the current state-of-the-art is dealing with issues of either large signature sizes or lack of flexibility of their schemes. While the current benchmark in terms of signature size for dynamic group signature (DGM [BLS+19]) is stateful and has an interactive verification process, our schemes avoid the use of the state and are non-interactive. To the best of our knowledge, T-SGS is the first fully dynamic stateless group signature constructed from symmetric primitives. Both constructions were theoretically and practically analyzed, which demonstrated that they produce a valuable compromise between flexibility and signature size. The results and the contributions of (T-)SGS is presented in Table 4.4, which is an updated Table 1.4 available on page 8.

The construction of T-SGS demonstrates the difficulty to design a stateless group signature from symmetric primitives. Indeed, there is a default at the heart of T-SGS construction. The only way to provide traceability to a group signature constructed from SPHINCS+ is to include a accumulator as explained in Section 4.2.3. The problem ensuing from this integration is that the membership of proven twice in the signature. First, it is proven by demonstrating the knowledge of a valid authentication path from the bottom to the top of the SPHINCS+ hyper-tree. This is how the non-traceable group signature SGS was build. Secondly, the membership to the group is proven through the membership proof of the accumulator. The second part is necessary to ensure traceability and therefore makes the first part redundant and useless.

Our desire to achieve the stateless feature lead us to this problem. In DGM, traceability was provided because each leaf of the tree belongs to one member only. In T-SGS, the leaves are shared among the group members, which means that a leaf could not be associated with a member and therefore, the only way to provide traceability was to integrated the accumulator. This chapter demonstrates that, currently, the only possibility to achieve the stateless feature when working with symmetric primitives, is to associate a non-interactive zero-knowledge proof and an accumulator.

This chapter orientates the future works related to symmetric-based stateless group signature. The research community requires to focus on improving accumulators and non-interactive zero-knowledge proof, both constructed with symmetric primitives, to compete with the stateless lattice-based constructions, which are stateless and achieve better signature size (see Table 4.4). For this reason, we recommend the use of Katz et al. [KKW18]. group signature over T-SGS, when the stateless feature is desired.

TABLE 4.4: Comparison of different post-quantum group signatures for 128-bits of post-quantum security. N denotes the number of members in the group, N.A.=Non Applicable, B=Maximum possible signatures per member, low= The group manager $\mathcal{M}$ cannot frame a signature, high=$\mathcal{M}$ can frame a signature

		[DPLS18]	[EZS+19]	[BEF19]	[KKW18]	[EBM18]	DGM	SGS	T-SGS
$\|\sigma\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$
$\|\sigma\|$ (KB)	$N = 2^7$	581	39	1370	315	2.72	2.82	33	359
	$N = 2^{10}$	581	54	1850	418	N.A.	2.82	33	501
	$N = 2^{20}$	581	140	3450	770	N.A.	2.82	33	970
$\|\mathsf{usk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log B)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$
$\|\mathsf{usk}\|$ (KB)		146	0.059	$0.032 \log N$	$0.032 \log N$	$2.2 \log B$	$2.8 \log B$	275	$275 + 0.032 \log N$
$\|\mathsf{gpk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$
$\|\mathsf{gpk}\|$ (KB)		123	$N \cdot 11.1$	0.032	0.032	0.032	0.032	0.032	0.032
Sign (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$
Sign (ms)[a]		450	No data	No data	3000	N.A.	1	1244	7383
Verify (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$
Verify (ms)[a]		169	No data	No data	3000	N.A.	Interactive	0.1	3886
PQ canditate		Lattice	Lattice	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric
Stateless		Yes	Yes	Yes	Yes	No	No	Yes	Yes
Dynamic		Static	Partially	Static	Fully	Static	Fully	Fully	Fully
Traceable		Yes	Yes	No	Yes	Yes	Yes	No	Yes
Non-interactive		Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Level of Trust in $\mathcal{M}$		Low	Low	Low	Low	High	High	High	High

[a]The times provided are for $N = 2^{10}$ and taken from the original papers and were implemented in different programming languages and executed on different types of machines

Chapter 5

xGS: Efficient Post-Quantum Group Signature from Symmetric Primitives with User-Controlled Linkability

This chapter is dedicated to the presentation of the last group signature designed in this thesis. As stated in the previous chapters, group signatures are promising **privacy-preserving** primitives as they allow members of a group to sign a digital message on behalf of the group while remaining anonymous. Consequently, a significant amount of research has investigated and developed this primitive. In this primitive, group members are managed by a central authority called the manager and represented by $\mathcal{M}$. This authority is responsible for initiating the group and every prospective member needs to interact with him in order to become a group member. The manager is also able to trace a signature and reveal the identity of the signer. In this chapter, similar to the rest of the thesis, we focus on fully dynamic constructions which allow new members to join and leave at any time in the group life, this increases the range of application compared to static group signature, where the set of members remains unchanged after the setup, and partially dynamic group signature, where new members can join the group after the completion of the setup phase but not leave. This chapter includes the content of our paper "Efficient Post-Quantum Group Signature from Symmetric Primitives with User-Controlled Linkability" submitted at the conference ACM ASIACCS 2022 [ACMa]. This chapter targets Research Objective O1 (see Chapter 1.3.2).

5.0.1 Group signature and quantum security

The launch of the NIST post-quantum standardization process shed light on the need for practical primitives which can resist attacks from quantum computers. Group signatures are no different. The appearance of quantum computers will require post-quantum group signature and changes to post-quantum security need to happen before this appearance. Therefore, we require **practical** post-quantum group signatures which can work on classical computers. Different post-quantum candidates exist to construct quantum-safe group signatures, with lattice-based cryptography being the most common. This trend is illustrated in the works of Gordon et al. [GKV10], Libert et al. [LLNW16],[LLM+16], Ling et al. [LNW15],[LNWX18], Del Pino et al. [DPLS18] and more recently Esgin et al. [EZS+19]. Another promising post-quantum candidate is the construction based on symmetric primitives such as cryptographic hash function or block cipher. Only four group signatures have been constructed with

symmetric primitives [EBM18],[KKW18],[BEF19] and [BLS+19]. Symmetric primitives have algorithms that appear to be efficient on classical computers in terms of running time. The security of those primitives is well understood among the research community and an asset for the security of symmetric-based protocols as it depends only on the symmetric primitive used. In this framework, if a primitive has been broken (e.g. DES), the protocol just needs to replace it by a new primitive (e.g. AES) and the protocol would be secure again. This contrasts with lattice-based primitives whose security depends on specific hardness assumptions, which means that if the hardness assumption is compromised, all primitives constructed based on that become obsolete. Symmetric primitives provide post-quantum security and numerous advantages despite their large signature size and their tendency to be stateful, which means that users need to update a state after each signature and any failure in the update could break the security of the scheme.

5.0.2 Post-quantum group signature from symmetric primitive

At the time of the design of the group signature presented in this chapter, the literature was composed of a number group signatures. Boneh et al. [BEF19] proposed a dynamic group signature constructed with symmetric primitives which is based on the zero-knowledge proof system named ZKB++ [CDG+17]. In 2018, Katz et al. [KKW18] proposed a new draft of post-quantum group signature constructed with an improved zero-knowledge proof reducing the signature size. However, as shown in Table 5.4, the scheme in [BEF19] does provide the traceability property and [KKW18] has only been described as a static group signature. These constructions are outperformed in terms of signature size by the DGM scheme presented in Chapter 3, which extends the static G-Merkle [EBM18] and is the only group signature that provides both traceability and full dynamicity at the same time. However, DGM requires the verifier to interact with the group manager during the verification process, which is not a standard requirement for group signatures and may create an obstacle in practical deployments.

5.0.3 Group signature and linkability

By definition, a group signature is unlinkable, which means that it is infeasible to determine if two signatures have been generated by the same group member with the group public key. Yet linkability is desired in certain applications as e-voting or cryptocurrencies, which are of great interest to the research community. The linkability in e-voting or cryptocurrency is **compulsory linkability** (that is, users cannot choose to be unlinked) when another relaxed level of linkability called **user-controlled linkability** may be preferred in other applications. User-controlled linkability allows a group member to be *voluntarily* linked while it remains anonymous, which means that a user can decide if its signature can be linked or not without the intervention of a trusted party leaving the power to the users. User-controlled linkability is a desired property in applications similar to Direct Anonymous Attestation (DAA) [BFG+13, CDL16], but we also believe that online survey systems are favorable to have this feature as the additional linkability may provide extra information to the organization (e.g. the same customer goes for shopping in different stores or different dates). Recent work presented by Diaz et al. [DL21] highlights the interest of user-controlled linkability with a possible application for smart vehicles in intelligent transportation systems where data (i.e. messages) are sent and used to detect anomalies. A driver could choose to make its data linkable to help detect any vehicle

problem on his vehicle while remaining anonymous. Another driver could also choose to keep its data unlinkable to have a better level of privacy. Nonetheless, to the best of our knowledge, the existing constructions that provide any kind of user-controlled linkability are not resistant to quantum attacks.

5.0.4 Challenges

The challenges that we tried to solve in this chapter follow from the issues of DGM presented in Chapter 3 and from the issued of (T-SGS) presented in 4. Our challenge is thus to provide the features of dynamicity and a short signature size without any interaction with the group manager in the verification process. Additionally, we design the first post-quantum group signature with user-controlled linkability. This chapter focuses on symmetric primitives for the aforementioned reasons and fills a significant research gap compared to the lattice-based group signature constructions.

5.0.5 Detailed contributions

Motivated by the state-of-the-art stateful XMSS signature [HBGM15], we construct a post-quantum dynamic group signature. We summarize our contributions of this chapter as follows:

- **xGS: a group signature with user-controlled linkability**: We present the first post-quantum dynamic group signature (called xGS) with user-controlled linkability based on XMSS digital signature (see Section 5.4). Our xGS allows members to generate x unlinkable group signatures and $2^h - \mathsf{x}$ linkable ones for a user-chosen value x and the group parameter h.

- **Competitive signature size**: Our construction achieves a very short signature size for a post-quantum security of 128-bits. Our xGS enjoys a constant signature size of 16.14KB, regardless of the group size. When compared to the state-of-the-art post-quantum group signature schemes with the same features (or even less), our construction matches or outperforms their signature size (see Section 5.5 for analysis and Table 5.4 for comparison results)

- **Algorithms efficiency**: Our construction features efficient algorithms as our signing algorithm can be executed in 1496ms (including online computations of only 1ms) and verifying processes can be executed in less than 22ms (see Section 5.5). Similar to the signature size, the algorithm efficiency does not depend on the group size, which is an advantage compared to the state-of-the-art (see Section 5.5).

5.0.6 Related works

This part presents the related works when the group signature presented in the chapter was designed. Quantum resistant group signature schemes are mostly constructed from lattices. This trend is illustrated by the works of Libert et al. [LLM+16] [LLNW16], Ling et al. [LNW15],[LNWX17], Del Pino et al. [DPLS18], and Esgin et al. [EZS+19].

There are also various post-quantum signature schemes based on symmetric primitives including the old work of Lamport et al. [Lam79] and the more recent [Hül13]. Nowadays, we count four group signature based on symmetric primitives only. Two of them [EBM18] and [BLS+19] are following the construction of the Merkle signature [Mer89]. However, the main weakness with this both group signatures is that the

group members need to maintain a state and any member's errors are subject to a leak of anonymity.

At the heart of Katz et al. group signature zero-knowledge proof such as ZKB++[CDG+17], and KKW [KKW18]. The state-of-the-art post-quantum signatures based on symmetric primitives are constructed by Katz et al., who built digital signature schemes by improving the zero-knowledge proof ZKB++ [CDG+17] named Picnic. Alongside their zero-knowledge proofs, either Fiat-Shamir Transform [FS86] or Unruh Transform [Unr15] were employed to construct an Non-Interactive zero-knowledge proof (NIZK) [BG92] as well.

Berntstein et al. [BHK+19], create SPHINCS+ and they succeed to avoid the use of the state. Their approach is really interesting because they do not use zero-knowledge proof and they succeed to remove the state which results in an increase of the signature size compared to the stateful Merkle signature [Mer89], but it outperforms the other stateless signature based on symmetric primitives Picnic [CDG+17] in terms of signature size and algorithms computation efficiency.

5.0.7 Chapter organization

This chapter uses a different definition of group signature presented in Chapter 2 as we have the "user-controlled" linkability. Therefore, Chapter 5 starts with the foundation of group signatures with user-controlled linkability in Section 5.1. Section 5.3 presents the digital signature schemes XMSS and XMSS^{MT} on which our scheme is based. Section 5.4 discusses our construction named xGS and its security. The chapter finishes on the evaluation of our schemes in Section 5.5, including a comparative summary of our scheme with the current state-of-the-art. It also summarizes the final results related to Research Objective O1 by updating for the last time Table 5.4.

5.1 Foundation of group signature with user-controlled linkability

Similar to traditional group signature presented in Chapter 2, we distinguish three identities in a group signature with user-controlled linkability: Group Manager $\mathcal{M}$, the member and verifier. Their detailed presentations are illustrated in Section 2.9 on page 22.

Definition 5.1.1 (Fully dynamic group signature with user-controlled linkability)**.** *A dynamic group signature with user-controlled linkability* GS *is defined by the following polynomial-time (PPT) algorithms:*

$(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{GS.Setup}(1^\lambda, \mathsf{param})$*: This procedure is executed by the central authority* $\mathcal{M}$*. It takes as inputs the security parameter* λ *and the group parameters and outputs the group public key* gpk*, the group secret key* gsk*.*

$(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{GS.UKeyGen}(1^\lambda, \mathsf{x}, \mathsf{param})$*: This procedure is executed by all prospective members. It takes as input the security parameter* λ*, the integer* x*, which indicates the desired number of unlinkable signatures, and the group parameters* param*. This outputs the user's public and secret key* upk, usk*.*

$\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gsk} \rangle_{\mathcal{M}} \leftarrow \mathsf{GS.Join}(\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk} \rangle_{\mathcal{M}})$*: This is an interactive protocol between* $\mathcal{M}$ *and an individual user* $\mathcal{U}$ *who wants to join the group. The user takes as inputs a user secret key and a user public key while* $\mathcal{M}$ *takes as inputs* gpk*,* gsk*. At the end of this protocol, the user updates his* usk *while* $\mathcal{M}$ *updates the group secret key* gsk*.*

$\perp /(\sigma, \mathsf{usk}) \leftarrow \mathsf{GS.Sign}(m, \mathsf{usk}, \mathsf{param}, \mathsf{link}, \mathsf{x})$: *A group member* $\mathcal{U}$ *executes the signing procedure with its secret key* usk*, a message* m *parameters* param*, a boolean* link*, which indicates if the signature is meant to be linkable, and the number of unlinkable signatures* x*. It outputs a valid group signature* σ *and an updated* usk *or* $\perp$ *if* 2^h *signatures have been previously generated from* usk*.*

$0/1 \leftarrow \mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param})$: *This is a deterministic and public algorithm that verifies the validity of the signature* σ *for a message* m *with the group public key* gpk *and parameters* param*. It outputs* 1 *if* σ *is valid,* 0 *otherwise.*

$\mathcal{U}/\perp \leftarrow \mathsf{GS.Open}(\sigma, \mathsf{gpk}, \mathsf{gsk})$: $\mathcal{M}$ *executes this procedure on a valid signature* σ*, the group public key* gpk *and the group secret key* gsk*. It outputs the identity* $\mathcal{U}$ *that generated* σ *or* $\perp$ *if* σ *cannot be associated with a group member.*

$\mathsf{gpk} \leftarrow \mathsf{GS.Revoke}(\mathcal{U}, \mathsf{gsk}, \mathsf{gpk})$: *This algorithm revokes the ability of the user* $\mathcal{U}$ *to generate valid signatures. It takes as input the member* $\mathcal{U}$ *to be revoked, the group secret key* gsk *and the group public key* gpk*. It outputs an updated group public key* gpk *(including an updated list of all revoked members* $\mathcal{R}$*).*

$0/1 \leftarrow \mathsf{GS.Link}(\mathsf{gpk}, (\sigma_1, m_1), (\sigma_2, m_2), \mathsf{param})$: *This algorithm takes as input the group public key* gpk*, two tuples composed of a signature* σ_i *and a message* m_i *with* $i = 1, 2$*. It outputs* 1 *if* σ_1 *and* σ_2 *have been generated by the same member, or* 0 *otherwise.*

We follow the security definition presented in Section 2.9, with an additional *user-controlled linkability* requirement. In this important to notice, that we omit the properties of tracing-soundness and revocability. The tracing soundness follows directly from the traceability and the revocability is studied in the non-fraemability experiment. All security experiments **Exp** are described in Figure 5.1 on page 81.

Definition 5.1.2 (Correctness)**.** *A* GS *achieves correctness if and only if for a security parameter* λ *the advantage* $\mathbf{Adv}^{Corr}_{\mathcal{A}}$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}^{Corr}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{Corr}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{5.1}$$

where the correctness experiment $\boldsymbol{Exp}^{Corr}_{\mathcal{A},\mathsf{GS}}(\lambda)$ *is defined in Figure 5.1.*

Definition 5.1.3 (Non-frameability)**.** *A* GS *provides non-frameability if a coalition of group members and/or outsiders is not able to forge a valid signature* σ' *which can be linked to an honest member* $\mathcal{U}$ *outside the coalition. A* GS *achieves non-frameability if and only if, for a security parameter* λ*, the advantage* $\mathbf{Adv}^{Frame}_{\mathcal{A}}(\lambda)$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}^{Frame}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{Frame}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1] < \mathsf{negl}(\lambda). \tag{5.2}$$

where the non-frameability experiment $\boldsymbol{Exp}^{Frame}_{\mathcal{A},\mathsf{GS}}(\lambda)$ *is described in Figure 5.1 (page 81).*

Definition 5.1.4 (Anonymity)**.** *A* GS *provides anonymity if and only if a signature* σ *does not leak any information on the identity of the signer. A* GS *achieves anonymity if and only if, for a security parameter* λ*, the advantage* $\mathbf{Adv}^{Anon}_{\mathcal{A}}$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}^{Anon}_{\mathcal{A}}(\lambda) = |\Pr[\boldsymbol{Exp}^{Anon-0}_{\mathcal{A},\mathsf{GS}}(\lambda) = 0] - \Pr[\boldsymbol{Exp}^{Anon-1}_{\mathcal{A},\mathsf{GS}}(\lambda) = 1]| < \mathsf{negl}(\lambda). \tag{5.3}$$

where the anonymity experiment $\boldsymbol{Exp}^{Anon-b}_{\mathcal{A},\mathsf{GS}}(\lambda)$ *is presented in Figure 5.1 (page 81).*

Definition 5.1.5 (Traceability). *A* GS *achieves traceability if and only if it is computationally infeasible for a PPT adversary to generate a valid signature* σ *which cannot be linked with a group member (i.e* $\perp \leftarrow \mathsf{GS.Open}(\sigma, \mathsf{gpk}, \mathsf{gsk})$*). A* GS *achieves traceability if and only if, for a security parameter* λ*, the advantage* $\mathbf{Adv}_{\mathcal{A}}^{Trace}(\lambda)$ *of an PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}_{\mathcal{A}}^{Trace}(\lambda) = \Pr[\boldsymbol{Exp}_{\mathcal{A},\mathsf{GS}}^{Trace}(\lambda) = 1] < \mathsf{negl}(\lambda). \tag{5.4}$$

where the traceability experiment $\boldsymbol{Exp}_{\mathcal{A},\mathsf{GS}}^{Trace}(\lambda)$ *is illustrated in Figure 5.1 (page 81).*

Definition 5.1.6 (User-Controlled Linkability). *A* GS *achieves user-controlled linkability if and only if it is computationally infeasible for a PPT adversary to generate* $\mathsf{x}+1$ *valid signatures such that all signature pairs* (σ_i, σ_j) *are not linked together, where* $i, j \in [1, \mathsf{x}+1]$*. A* GS *achieves user-controlled linkability if and only if, for a security parameter* λ*, the advantage* $\mathbf{Adv}_{\mathcal{A}}^{Link}(\lambda)$ *of a PPT adversary* $\mathcal{A}$ *satisfies*

$$\mathbf{Adv}_{\mathcal{A}}^{Link}(\lambda) = \Pr[\boldsymbol{Exp}_{\mathcal{A},\mathsf{GS}}^{Link}(\lambda) = 1] < \mathsf{negl}(\lambda). \tag{5.5}$$

where the linkability experiment $\boldsymbol{Exp}_{\mathcal{A},\mathsf{GS}}^{Link}(\lambda)$ *is illustrated in Figure 5.1 (page 81).*

5.1.1 Security Oracles

A secure group signature with user-controlled linkability achieves the security properties of Correctness, Non-frameability, Anonymity, Traceability and User-Controlled Linkability. To prove these requirements, we use a set of games, in which a PPT adversary $\mathcal{A}$ can access oracles. There are also a number of different global sets: $\mathcal{S}$ is list of signatures obtained from the *Sign* oracle, $\mathcal{H}$ is the the list of honest group members, $\mathcal{CO}$ is the list of corrupted members and $\mathcal{CH}$ is the list of outputs of the challenge oracle $Chal_b$.

- $\sigma \leftarrow Sign(\mathcal{U}, m)$: It returns a signature for the message m by the group member $\mathcal{U}$ ($\sigma \leftarrow \mathsf{GS.Sign}(m, \mathsf{usk}, \mathsf{param}, \mathsf{link}, \mathsf{x})$) if and only if $\mathcal{U} \in \mathcal{H}$ and $\mathcal{U} \notin \mathcal{CO}$. All queries performed with this oracle are inserted in a set $\mathcal{S}$ ($\mathcal{S} \leftarrow \mathcal{S} \bigcup \{(m, \sigma, \mathcal{U})\}$).
- $\mathcal{U} \leftarrow Open(m, \sigma)$: It returns the identity of the generator of the signature σ for the message m. More formally, it returns $\mathsf{GS.Open}(\sigma, \mathsf{gpk}, \mathsf{gsk})$ if and only if $\mathsf{GS.Verify}(m, \sigma, \mathsf{gpk}, \mathsf{param}) = 1$ and $(m, \sigma) \notin \mathcal{CH}$ or, in other words, if the signature is valid and was not obtained from the $Chal_b$ oracle.
- $\mathsf{usk} \leftarrow RevealM(\mathcal{U})$: It returns the personal secret key usk of an honest group member $\mathcal{U}$. $\mathcal{U}$ is added to the list of corrupted members $\mathcal{CO}$, $\mathcal{CO} \leftarrow \mathcal{CO} \bigcup \{\mathcal{U}\}$
- $\mathsf{upk} \leftarrow AddM(\mathcal{U})$: It adds a new honest member $\mathcal{U}$ to the group by generating its public and secret keys (upk, usk) and initiates the joining process with $\mathcal{M}$. It returns the public key upk and updates the set of honest members: $\mathcal{H} \leftarrow \mathcal{H} \bigcup \{\mathcal{U}\}$
- $\mathsf{usk} \leftarrow CorruptM(\mathcal{U})$: It creates a new corrupted member $\mathcal{U}$ and initiates the joining process with $\mathcal{M}$. $\mathcal{U}$ is added to the list of corrupted members $\mathcal{CO}$, $\mathcal{CO} \leftarrow \mathcal{CO} \bigcup \{\mathcal{U}\}$.
- $\sigma \leftarrow Chal_b(\mathcal{U}^0; \mathcal{U}^1, m)$: It returns a signature generated by the member $\mathcal{U}^b$ only if both $\mathcal{U}^0$ and $\mathcal{U}^1$ are honest members ($\mathcal{U}^0 \in \mathcal{H}$ and $\mathcal{U}^0 \notin \mathcal{CO}$ and $\mathcal{U}^1 \in \mathcal{H}$ and $\mathcal{U}^1 \notin \mathcal{CO}$). This oracle can be called only once in the anonymity experiments (see Figure 5.1). The message and the generated signature are added to the set $\mathcal{CH}$, $\mathcal{CH} \leftarrow \mathcal{CH} \bigcup \{(m, \sigma)\}$.

$\underline{\mathbf{Exp}^{Corr}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk},\mathsf{gsk}) \leftarrow \mathsf{GS.Setup}(1^\lambda,\mathsf{param})$
$(\mathsf{id}_\mathcal{U}, m) \leftarrow \mathcal{A}^{AddM}(\mathsf{gpk},\mathbf{members})$
if $\mathcal{A}$'s outputs are not valid
  then return 0
$\sigma,\mathsf{usk} \leftarrow\!\leftarrow \mathsf{GS.Sign}(m,\mathsf{usk},\mathsf{param},\mathsf{link},\mathsf{x})$
if $\mathsf{id}_\mathcal{U} \notin$ **members** or $\mathsf{upk} \in \mathcal{R}$
  then return 1
if $\mathsf{GS.Verify}(\mathsf{m},\sigma,\mathsf{gpk},\mathsf{param}) = 0$
  then return 1
then return 0

$\underline{\mathbf{Exp}^{Frame}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk},\mathsf{gsk}) \leftarrow \mathsf{GS.Setup}(1^\lambda,\mathsf{param})$,
$\mathcal{H},\mathcal{CO},\mathcal{S} \leftarrow \emptyset$
$(\mathcal{U}, m, \sigma) \leftarrow$
$\mathcal{A}\left\{\begin{array}{cc} AddM & CorruptM \\ Sign & RevealM \end{array}\right\}(play : \mathsf{gpk})$
if $\mathsf{GS.Verify}(m,\sigma,\mathsf{gpk},\mathsf{param}) = 0$ or $(m,\mathcal{U}) \in \mathcal{Q}$
  then return 0
$\mathcal{U} \leftarrow \mathsf{GS.Open}(\sigma,\mathsf{gpk},\mathsf{gsk})$
if $\mathcal{U} \notin \mathcal{R}$
  then return 1
return 0

$\underline{\mathbf{Exp}^{Anon-b}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk},\mathsf{gsk}) \leftarrow \mathsf{GS.Setup}(1^\lambda,\mathsf{param})$,
$\mathcal{H},\mathcal{CO},\mathcal{S},\mathcal{CH} \leftarrow \emptyset$
$d \leftarrow \mathcal{A}\left\{\begin{array}{cc} AddM & Chall_b \\ RevealM & Open \\ CorruptM & Sign \end{array}\right\}(play$: $\mathsf{gpk})$
Return d

$\underline{\mathbf{Exp}^{Trace}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk},\mathsf{gsk}) \leftarrow \mathsf{GS.Setup}(1^\lambda,\mathsf{param})$,
$\mathcal{H},\mathcal{CO},\mathcal{S} \leftarrow \emptyset$
$(m,\sigma,\mathsf{gpk}) \leftarrow$
$\mathcal{A}\left\{\begin{array}{cc} AddM & Sign \\ CorruptM & RevealM \end{array}\right\}(play$: $\mathsf{gpk})$
if $\mathsf{GS.Verify}(m,\sigma,\mathsf{gpk},\mathsf{param}) = 0$
  then return 0
$\mathcal{U} \leftarrow \mathsf{GS.Open}(\sigma,\mathsf{gpk},\mathsf{gsk})$
if $\mathcal{U} = \perp$
  then return 1
then return 0

$\underline{\mathbf{Exp}^{Link}_{\mathcal{A},\mathsf{GS}}(\lambda)}$:
$(\mathsf{gpk},\mathsf{gsk}) \leftarrow \mathsf{GS.Setup}(1^\lambda,\mathsf{param})$,
$\mathcal{H},\mathcal{CO},\mathcal{S} \leftarrow \emptyset$
$(\mathsf{usk},\mathsf{upk},(m_i,\sigma_i)_{i=1}^{\mathsf{x}}) \leftarrow$
$\mathcal{A}^{Sign,Corrupt}(play : \mathsf{gpk})$
$(m_{\mathsf{x}+1},\sigma_{\mathsf{x}+1}) \leftarrow \mathcal{A}^{Sign}(play : \mathsf{gpk})$
if $\mathsf{GS.Link}(\mathsf{gpk},(m_1,\sigma_1),(m_1,\sigma_1)) = 0$ & $\mathsf{GS.Open}(\sigma_1,\mathsf{gpk},\mathsf{gsk}) =$ $\mathsf{GS.Open}(\sigma_2,\mathsf{gpk},\mathsf{gsk})$
  then return 1
Return 0

FIGURE 5.1: Security Experiments

5.2 Additional definitions

This section defines the additional primitives used in this chapter. The other global cryptographic parameters are presented in Chapter 2.

Definition 5.2.1 (Pseudorandom permutation (PRP))**.** *A Pseudorandom permutation*

$$P : \{0,1\}^n \times \{0,1\}^{2\lambda} \rightarrow \{0,1\}^n, y \leftarrow P(x, \mathsf{sk}) \tag{5.6}$$

is a function which takes as input an element of length n and a key sk of length 2λ and outputs the pseudorandom value b of length n bits. A PRP is defined by three properties:

- *P is one-to-one from $\{0,1\}^n$ to $\{0,1\}^n$ for any $\mathsf{sk} \in \{0,1\}^{2\lambda}$.*
- *P can be inverted for any secret key $\mathsf{sk} \in \{0,1\}^{2\lambda}$.*
- *P over sk is indistinguishable from a random permutation.*

Definition 5.2.2 (WOTS^+)**.** *WOTS^+ is a one-time signature scheme defined by the following algorithms and which is built with a cryptographic hash function only:*

$(\mathsf{WOTS}^+.\mathsf{pk}, \mathsf{WOTS}^+.\mathsf{sk}) \leftarrow \mathsf{WOTS}^+.\mathsf{KeyGen}(1^\lambda)$*: This generates a WOTS^+ key pair.*

$\mathsf{WOTS}^+.\sigma \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(m, \mathsf{WOTS}^+.\mathsf{sk})$*: This takes as input a message m and a secret key $\mathsf{WOTS}^+.\mathsf{sk}$ and outputs a signature $\mathsf{WOTS}^+.\sigma$.*

$0/1 \leftarrow \mathsf{WOTS}^+.\mathsf{Verify}(m, \mathsf{WOTS}^+.\sigma, \mathsf{WOTS}^+.\mathsf{pk})$*: This takes as input a message m, a signature $\mathsf{WOTS}^+.\sigma$ and a public key $\mathsf{WOTS}^+.\mathsf{pk}$ and outputs 1 if $\mathsf{WOTS}^+.\sigma$ is valid, or 0 otherwise.*

WOTS^+ provides existential unforgeability under adaptive chosen message attacks EU-CMA security (see Definition 2.4.2) if a key pair $\mathsf{WOTS}^+.\mathsf{pk}, \mathsf{WOTS}^+.\mathsf{sk}$ is used only **once**. In this chapter, we use WOTS^+ as a black box and therefore we only provide the high level description here. Readers may refer to [Hül13] for the details.

Remark 2. *In this chapter, we are using WOTS^+ in the same context of XMSS. This means that WOTS^+ signatures are never formally verified. The verification procedure takes as input only a message and a signature $\mathsf{WOTS}^+.\sigma$ and outputs the valid public key: $\mathsf{WOTS}^+.\mathsf{pk} \leftarrow \mathsf{WOTS}^+.\mathsf{Verify}(m, \mathsf{WOTS}^+.\sigma)$.*

5.3 XMSS and XMSS^{MT}

As previously stated, our construction is based on same idea as the stateful digital scheme XMSS [HBG+18]. This section introduces XMSS and its extension XMSS^{MT} [HBG+18]. Both are stateful schemes and therefore, after each signature, the secret key needs to be updated. This is not the case for stateless schemes. They are based on the idea of Merkle trees [Mer89], which are binary trees in which each nodes is the hash value (output of crytpographic hash function H, see Definition 2.2.1) of both its children nodes. XMSS constructs the tree using the keys of a one-time signature scheme named WOTS^+ (see Definition 5.2.2) as leaves. XMSS and XMSS^{MT} are digital signature schemes (see Definition 2.4.1) which have been proven EU-CMA secure (see Definition 2.4.2) in [HBG+18].

5.3.1 XMSS

This sections presents a definition of XMSS [HBG+18] whose main aim is to facilitate the comprehension of our scheme.

Definition 5.3.1 (XMSS). XMSS *[BDH11] is a stateful digital signature defined by the three following PPT algorithms:*

$\mathsf{XMSS.pk} \leftarrow \mathsf{XMSS.KeyGen}\ (\{\mathsf{WOTS^+.pk}_j, \mathsf{WOTS^+.sk}_j\}_{j=0}^{2^h-1}, b)$*: This takes as inputs* 2^h WOTS$^+$ *key pairs and the public bitmask* $b = \{(b_{l,j}||b_{r,j})\}_{j=1}^{h}$ *to construct the tree. It constructs a binary tree of height* h *and outputs the root* XMSS.pk *(see Figure 5.2).*

$\mathsf{XMSS}.\sigma \leftarrow \mathsf{XMSS.Sign}(m, \{\mathsf{WOTS^+.pk}_j, \mathsf{WOTS^+.sk}_j\}_{j=0}^{2^h-1}, \mathsf{idx})$*: This takes as input a message* m *to be signed, a set of* WOTS$^+$ *keys* $\{\mathsf{WOTS^+.pk}_j, \mathsf{WOTS^+.sk}_j\}_{j=0}^{2^h-1}$ *and the state* idx*. This executes the following steps:*

1. $\mathsf{WOTS^+}.\sigma \leftarrow \mathsf{WOTS^+.Sign}(m, \mathsf{WOTS^+.sk_{idx}})$
2. $\mathsf{XMSS}.\sigma = (\mathsf{WOTS^+}.\sigma, \mathsf{idx}, \mathsf{auth})$*: The index* idx*, which is a* h*-bits string, indicates the position of the* WOTS$^+$ *key in the tree and* auth *is the authentication path from the* WOTS$^+$ *key to the root. In the example in Figure 5.2, we have* $\mathsf{idx} = (101)_2 = 5$ *and* $\mathsf{auth} = (n1, n2)$*.*
3. $\mathsf{idx} = \mathsf{idx} + 1$

$1/0 \leftarrow \mathsf{XMSS.Verify}(m, \mathsf{XMSS}.\sigma, \mathsf{XMSS.pk})$*: This takes as input a message* m*, a signature* XMSS.σ *and the* XMSS *public key* XMSS.pk*. It outputs* 1 *if* XMSS.σ *is valid or* 0 *otherwise. This executes the following steps:*

1. $\mathsf{WOTS^+.pk'} = \mathsf{WOTS^+.Verify}(m, \mathsf{WOTS^+}.\sigma)$ *(See Remark 2)*
2. *Compute* XMSS.pk′ *from* WOTS$^+$.pk′ *using the authentication path* auth *and the index* idx*, which indicates the position of* WOTS$^+$.pk′ *in the tree. (In Figure 5.2,* $\mathsf{XMSS.pk'} = \mathsf{H}(n2 \oplus b_{l,3}, \mathsf{H}(\mathsf{H}(\mathsf{WOTS^+.pk'} \oplus b_{l,1}, \mathsf{WOTS^+.pk}_5 \oplus b_{r,1}) \oplus \oplus b_{l,2}, n1 \oplus b_{r,2})) \oplus b_{r,3}))$
3. *Return* XMSS.pk == XMSS.pk′

Remark 3. *When* XMSS *is used in the context of* XMSS^{MT}*, the verification procedure takes as input only a signature* WOTS$^+$.σ *and outputs the valid root of the tree:* $\mathsf{XMSS.pk} \leftarrow \mathsf{XMSS.Verify}(m, \mathsf{XMSS}.\sigma)$.

Remark 4. *Our construction (see Section 5.4) is defining* XMSS *algorithms of with two additional inputs* R_i *and* r_i *for the key Generation and signing algorithms (For example:* $\mathsf{KeyGen}(\{\mathsf{WOTS^+.pk}_j, \mathsf{WOTS^+.sk}_j\}_{j=0}^{2^h-1}, R_i, r_i)$*. And one additional input* R *for the verification procedure. This adds* R_i *in the computation of every internal nodes of the* XMSS *tree (See Figure 5.5) and so is necessary to the signing and verification procedures*

5.3.2 XMSS^{MT}

XMSS^{MT} differs from the original XMSS by using a hyper-tree which is composed of multiple trees instead of a single one (see Figure 5.3). The hyper-tree has a height of h_{MT} and is divided in d levels of XMSS trees of height h_{MT}/d each. The advantage of XMSS^{MT} is that the generation of the WOTS$^+$ key pairs can be delayed over

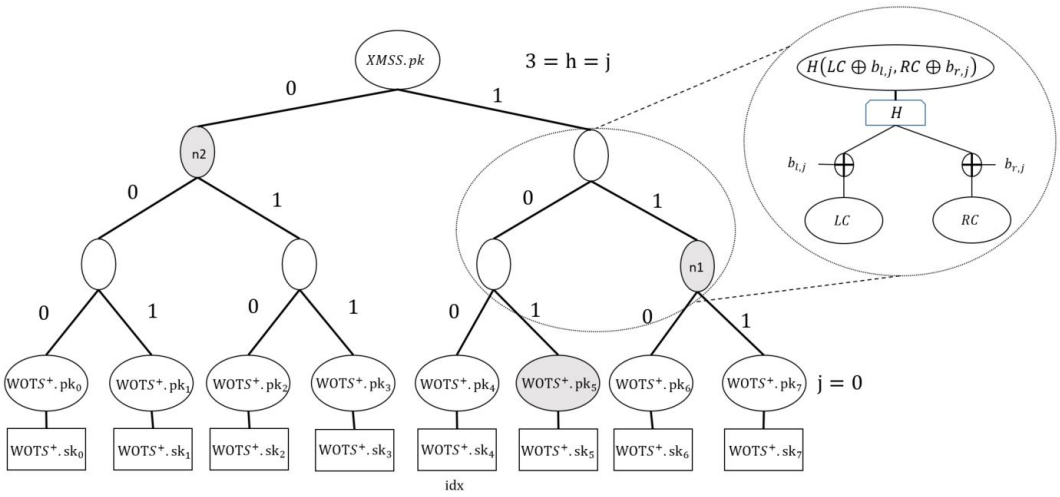


FIGURE 5.2: XMSS tree

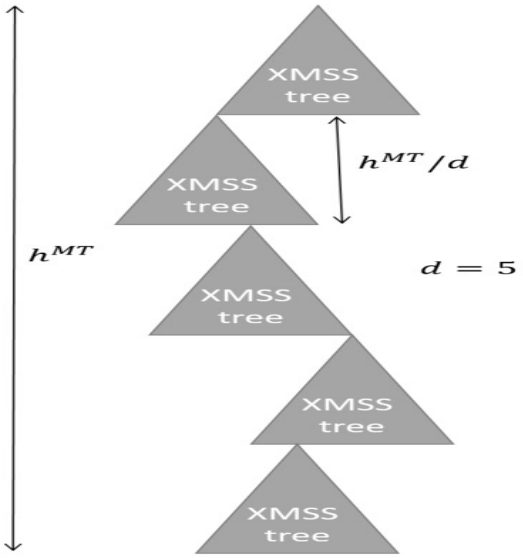


FIGURE 5.3: XMSS^{MT} hyper-tree

time. During the key generation, XMSS needs to generate 2^h WOTS$^+$ key pairs, while XMSS^{MT} only needs to generate $2^{h_{MT}/d} * d$ keys. This allows to have a higher tree height h_{MT} with much shorter key generation time and, consequently, significantly increases the maximum number of possible signatures. A signature XMSS$^{MT}.\sigma$ is composed of d WOTS$^+$ signatures $\{\text{WOTS}^+.\sigma\}_{i=1}^{d}$, d authentication paths $\{\mathsf{auth}\}_{i=1}^{d}$ and one h_{MT}-bit index idx_{MT}, which indicates the path to follow in the hyper-tree.

Definition 5.3.2 (XMSS^{MT})**.** *For the sake of our construction, we defined the* XMSS^{MT} *algorithm with the following algorithms:*

$(\text{XMSS}^{MT}.pk, \text{XMSS}^{MT}.sk) \leftarrow \text{XMSS}^{MT}.\mathsf{KeyGen}(1^\lambda)$*: This only needs to generate the top tree and outputs the root* XMSS$^{MT}.pk$ *(see Figure 5.3).*

$\text{XMSS}^{MT}.\sigma \leftarrow \text{XMSS}^{MT}.\mathsf{Sign}(m, \text{XMSS}^{MT}.sk, \mathsf{idx}_{MT})$*: This takes as input a message m to be signed, the secret key* XMSS$^{MT}.sk$ *and an index* idx_{MT} *to indicate the leaf used to do the* WOTS$^+$ *signature. It outputs a signature* $\text{XMSS}^{MT}.\sigma = (\{\text{WOTS}^+.\sigma_i, \mathsf{auth}\}_{i=1}^{d}, \mathsf{idx}_{MT})$*, which is composed of d* XMSS *signatures, where d is the number of levels in the hyper-tree. The tree at level i signs the root of the tree at level $i-1$.*

$1/0 \leftarrow \text{XMSS}^{MT}.\mathsf{Verify}(m, \text{XMSS}^{MT}.\sigma, \text{XMSS}^{MT}.pk)$*: This takes as input a message m, a signature* XMSS$^{MT}.\sigma$ *and the* XMSS^{MT} *public key* XMSS$^{MT}.pk$*. It outputs 1 if* XMSS$^{MT}.\sigma$ *is valid, which means that it was possible to reconstruct the root of the top layer starting from the first* WOTS$^+$ *signature, or 0 otherwise.*

5.4 xGS: group signature with user-controlled Linkability

This section presents the construction of our xGS following Definition 5.1.1. Our scheme is constructed with a cryptographic hash function H, the WOTS$^+$ scheme, a pseudorandom permutation scheme P, and a pseudorandom function PRF. All primitives are defined in Chapter 2.

5.4.1 High-level idea

To demystify our scheme, we introduce xGS with a basic example of a group made of two members who desire to generate three unlinkable signatures (see Figure 5.4). In xGS, the manager $\mathcal{M}$ starts the group by executing the XMSS^{MT} key generation algorithm. This will generate the XMSS^{MT} public key, which will be the group public key gpk, and the XMSS^{MT} secret key, which is the group secret key gsk. The members' side is presented in Section 5.4.2. Each member constructs x (number of desired unlinkable signatures) modified XMSS trees (see below in Section 5.4.2) but on the same set of 2^h WOTS$^+$ keys. Then, each member registers the x roots $\{\mathsf{mt}_i\}_{i=0}^{\mathsf{x}-1}$ and their associated random element $\{R_i\}_{i=0}^{\mathsf{x}-1}$ (presented in Section 5.4.2) to the central authority $\mathcal{M}$. Figure 5.4 illustrates an example for $\mathsf{x} = 3$, in which a member generates 3 modified XMSS trees. $\mathcal{M}$ signs these x roots and shares them with the member. After the joining procedure, each member has x $\mathcal{M}$'s signatures, in our case XMSS^{MT} signatures, which are integrated into the member's secret key usk. The role of $\mathcal{M}$ and his signature is detailed in Section 5.4.3.

A signature can be divided into two parts: a modified XMSS signature and $\mathcal{M}$'s signature, which authenticates the root of the modified XMSS tree used in the first part. To sign a message, each member has two choices: generating a linkable or an unlinkable signature. If he chooses to generate a linkable signature, he will use

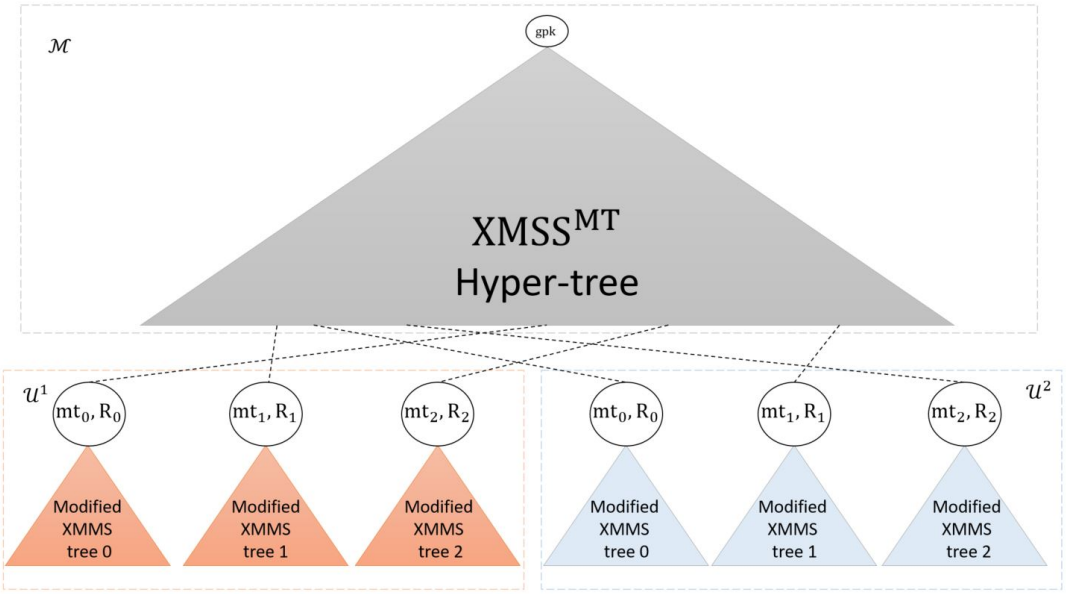


FIGURE 5.4: Example of structure for a group of two members and $\mathsf{x} = 3$

the modified XMSS tree number $\mathsf{x} - 1$ (2 in the example in Figure 5.4) to generate every linkable signature and add $\mathcal{M}$'s signature that authenticates the tree number $\mathsf{x} - 1$. This signature is unlinkable and becomes linkable after the second signature is generated by tree 2. When an unlinkable signature is desired, the member will use the modified trees 0 to $\mathsf{x} - 2$ starting with tree 0 for the first signature and then using the trees in order for the next signatures. Trees 0 to $\mathsf{x} - 2$ are used only once while the tree $\mathsf{x} - 1$ can be used multiple times. In Figure 5.4, the trees 0 and 1 are used to generate unlinkable signatures. The detailed algorithms of the constructions are presented in Algorithms 7 and 9. All parameters are summarized in Table 5.1.

5.4.2 Members' modified trees

This part presents the elements used by members (orange and blue squares in Figure 5.4). As previously stated, each xGS group member can generate a chosen number x of unlinkable signatures. Then, each group member generates x different modified XMSS trees on top of the same set of 2^h WOTS$^+$ key pairs but depending on 2x different random elements $\{R_i\}_{i=0}^{\mathsf{x}-1}$ and $\{r_i\}_{i=0}^{\mathsf{x}-1}$, which gives x different roots $\{\mathsf{mt}_i\}_{i=0}^{\mathsf{x}-1}$. The tuples $\{\mathsf{mt}_i, R_i\}$ will be authenticated by $\mathcal{M}$. All x tuples $\{\mathsf{mt}_i, R_i\}_{i=0}^{\mathsf{x}-1}$ will be registered to $\mathcal{M}$, who provides x signatures $\{\mathcal{M}.\sigma_i\}_{i=0}^{\mathsf{x}-1}$ of these tuples.

Modified XMSS

A member who desires to generate x unlinkable signatures will generate x modified XMSS trees. Our modified XMSS trees differ from the "traditional" XMSS trees in that it includes an additional level between the WOTS$^+$ public keys and the rest of the tree, two nonces are also used to compute the tree and the leaves are permuted. The reason behind these modifications is the anonymity desired for the group signature, which is not the case for simple digital signatures similar to XMSS. The modified XMSS algorithms are defined as follows:

- $\mathsf{mt}_i \leftarrow$ Mod.KeyGen$(\{\mathsf{WOTS}^+.\mathsf{pk}_j, \mathsf{WOTS}^+.\mathsf{sk}_j\}_{j=0}^{2^h-1}, i, R_i, r_i,\ sd, b, \mathsf{x})$: This algorithm takes as inputs a set of WOTS$^+$ keys, the tree number i, two nonces R_i to

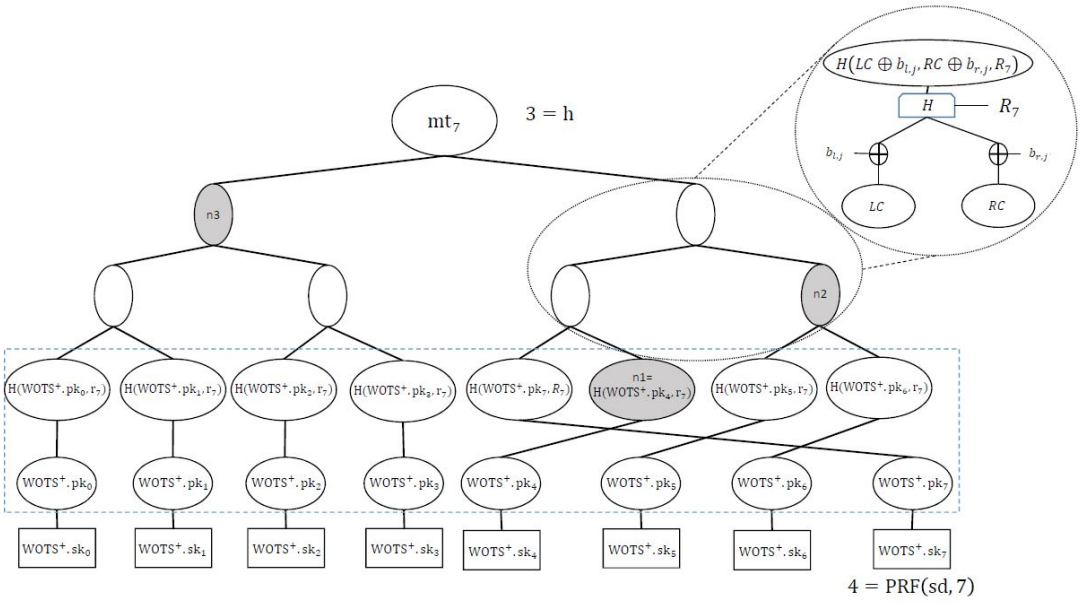


FIGURE 5.5: Modified XMSS tree example for $i = 7$.

ensure traceability and r_i to ensure unlinkability (see Remark 5), a secret seed sd, which is part of the member secret key usk, the public bitmask b, which belongs to the public parameters and therefore is common to all members, and the number of unlinkable signatures x. This outputs a root mt_i of the modified XMSS tree i, where $i \in [0, \mathsf{x}-1]$. This procedure starts by computing the leaves $\{leaf_j\}_{j=0}^{2^h-1}$, which are not simply the WOTS$^+$ public key like XMSS. The leaves are computed as follows:

- If $i < \mathsf{x} - 1$, then the leaves are computed as follows: $leaf_j \leftarrow \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_j, r_i)$ if $j \neq i$ and $leaf_j \leftarrow \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_j, R_i)$ if $j = i$. These trees are used only once to generate one unlinkable signature each.
- If $i = \mathsf{x} - 1$, then the leaves are computed as follows: $leaf_j \leftarrow \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_j, r_i)$ if $j < i$ and $leaf_j \leftarrow \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_j, R_i)$ if $j \geq i$. This tree is used to generate linkable signatures, therefore the construction is different.

Before the generation of the binary tree i, the position of leaf i in the tree should be randomized. (For example, we randomize the position of leaf #7 in binary tree #7. Also note that we only need to randomize the position of the first i leaf instead of all 2^h leaves, in i trees respectively.) The position of leaf i in the tree i is computed as follows: First, we compute $y \leftarrow \mathsf{PRF}(sd, i)$ (See Definition 2.6.1). We only take the first h bits of y to set the position (ranging from 0 to $2^h - 1$) of the leaf i in the tree. In the tree i, only the leaf i is re-positioned while the rest stays in sequential order. Figure 5.5 demonstrates an example for $i = 7$ (see Remark 6). Then, the modified XMSS tree is constructed with a similar procedure to a traditional XMSS, except that tree nodes are computed as follows: $\mathsf{H}(LC \oplus b_{l,j}, RC \oplus b_{r,j}, R_i)$ (LC=left child and RC=right child) instead of $\mathsf{H}(LC \oplus b_{l,j}, RC \oplus b_{r,j})$ in XMSS. This procedure ends by returning the root mt_i.

- $(\mathsf{WOTS}^+.\sigma, \mathsf{auth}, R_i, \mathsf{idx}) \leftarrow \mathsf{Mod.Sign}(m, R_i, r_i, i, j, sd, \mathsf{x}, b, \{\mathsf{WOTS}^+.\mathsf{pk}_{j'}, \mathsf{WOTS}^+.\mathsf{sk}_{j'}\}_{j'=0}^{2^h-1})$: This algorithm takes as inputs a message m, nonces R_i and r_i, an index i which indicates the i-th tree to be used, an index

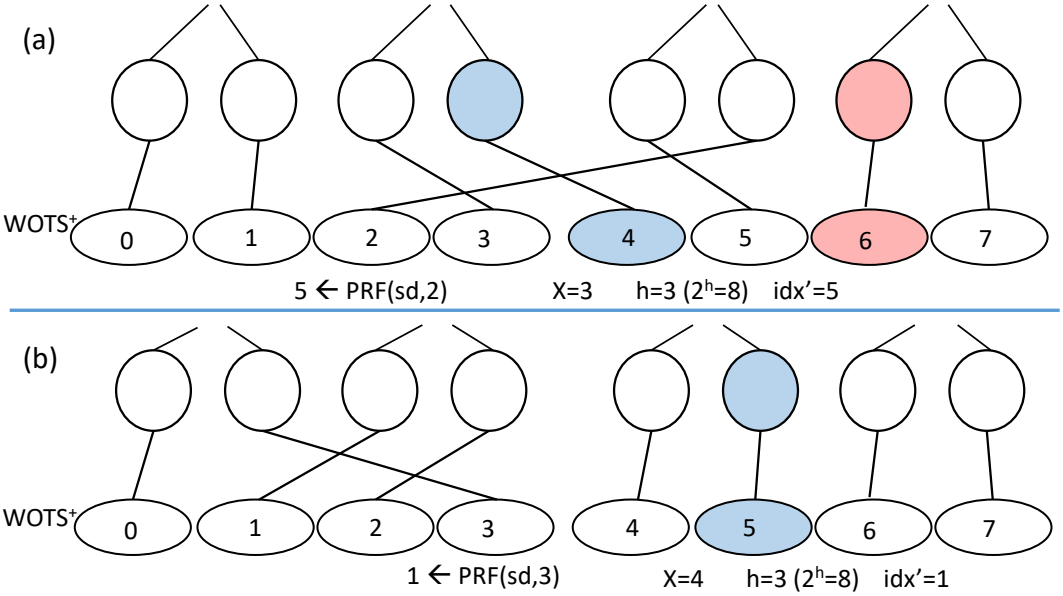


FIGURE 5.6: Leaf position (idx) in XMSS tree for $j \geq \mathsf{x}$. We distinguish two cases (a) $\mathsf{idx}' \geq \mathsf{x}$ and (b) $\mathsf{idx}' < \mathsf{x}$.

$0 \leq j \leq 2^h - 1$ which indicates the $\mathsf{WOTS}^+.\mathsf{sk}$ to be used, a secret seed sd, the number of unlinkable signatures x, the public bitmask b and a set of WOTS^+ keys. Then, this outputs a WOTS^+ signature $\mathsf{WOTS}^+.\sigma \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(m, \mathsf{WOTS}^+.\mathsf{sk}_i)$, the authentication path auth (the grey nodes in Figure 5.5), the nonce R_i used to construct the tree i if $i < \mathsf{x} - 1$ or $\mathsf{x} - 1$ and the index idx which indicates the position of the $\mathsf{WOTS}^+.\mathsf{sk}_j$ used in the modified XMSS tree, which is computed as follows:

- if $j < \mathsf{x}$: In this case, it is an unlinkable signature. j must be equal to i. In other words, idx is the first XMSS bits of $y \leftarrow \mathsf{PRF}(sd, j)$ (e.g. in Figure 5.5, $j = 7$, $\mathsf{x} = 8$, $\mathsf{idx} = 4$), as discussed above in Mod.KeyGen.
- if $j \geq \mathsf{x}$: In this case, it is a linkable signature. First, compute idx' to be the first H bits of $y \leftarrow \mathsf{PRF}(sd, \mathsf{x} - 1)$. If $\mathsf{idx}' < \mathsf{x}$, then set $\mathsf{idx} \leftarrow j$ (e.g. in Figure 5.6 (b), $j = 5$, $\mathsf{x} = 4$, $\mathsf{idx}' = 1$, $\mathsf{idx} = 5$). If $\mathsf{idx}' \geq \mathsf{x}$, we further divide into two cases: (i) $\mathsf{idx} \leftarrow j - 1$ if $j \leq \mathsf{idx}'$ (e.g. in Figure 5.6 (a), $j = 4$, $\mathsf{x} = 3$, $\mathsf{idx}' = 5$, $\mathsf{idx} = 3$); or (ii) $\mathsf{idx} \leftarrow j$ if $j > \mathsf{idx}'$ (e.g. in Figure 5.6 (a), $j = 6$, $\mathsf{x} = 3$, $\mathsf{idx}' = 5$, $\mathsf{idx} = 6$).

- $\mathsf{mt}'_i \leftarrow \mathsf{Mod.Verify}(m, \mathsf{WOTS}^+.\sigma, \mathsf{auth}, R_i, \mathsf{idx}, b)$: This algorithm takes as inputs the message m, the WOTS^+ signature $\mathsf{WOTS}^+.\sigma$, the authentication path auth, the nonce R_i, the index idx and the public bitmask b. This outputs a root mt'_i. This works exactly with the same procedure as the XMSS verification process except that R_i is included at each step of the path verification. From the example shown in Figure 5.5, the procedure starts by computing $\mathsf{WOTS}^+.\mathsf{pk}'_7 \leftarrow \mathsf{WOTS}^+.\mathsf{Verify}(m, \mathsf{WOTS}^+.\sigma)$ (See Remark 2) and then computing $\mathsf{mt}'_7 = \mathsf{H}(n3 \oplus b_{l,3}, \mathsf{H}(\mathsf{H}(\mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}'_7, R_7) \oplus b_{l,1}, n1 \oplus b_{r,1}, R_7) \oplus b_{l,2}, n2 \oplus b_{r,2}, R_7) \oplus b_{r,3}, R_7)$.

Remark 5. *The purpose of r_i is to hide WOTS^+ keys and to ensure unlinkability when desired. The example in Figure 5.5 has $n1 = \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_4, r_7)$. If R_i was used for all leaves of the tree i, we would have $n1 = \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_4, R_7)$. Now, assuming*

that the second nonce r_i is not used, we would have $n1 = \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_4, R_7)$ for a signature with the index $i = 7$. So, if we take the signature σ' generated with tree 4 (index $i = 4$) and composed of R_4 and $\mathsf{WOTS}^+.\sigma$ generated from $\mathsf{WOTS}^+.\mathsf{sk}_4$. An adversary can extract $\mathsf{WOTS}^+.\mathsf{pk}_4$ from σ' and then check if $n1 = \mathsf{H}(\mathsf{WOTS}^+.\mathsf{pk}_4, R_7)$ from the signature generated with the index $i = 7$. If the equality stands, it implies that both signatures have been generated by the same member, so it is possible to link any signatures, even unlinkable ones. Our scheme thus requires the use of a second nonce r_i which is kept secret to avoid this issue. In this example, as r_7 remains secret, the only way to link $n1$ to $\mathsf{WOTS}^+.\mathsf{sk}_4$ is to invert the cryptographic hash function H, which is by definition computationally infeasible (see Definition 2.2.1).

Remark 6. *For tree i, all leaves are in order except the leaf i. This position swap of the leaf i in tree i has two purposes. First, it avoids that the position of WOTS^+ keys appears in order similar to the original XMSS, which ensures unlinkability between two consecutive signatures. Second, we use the first h bits of the PRF output value to allocate a new position for leaf i in tree i. This means that it is possible for a member to generate two signatures with different WOTS^+ keys but in the same index even in two different modified trees. It is therefore infeasible for an adversary to affirm that two signatures have been generated by different members if the WOTS^+ keys are in the same position in the tree.*

After the completion of the joining process, each group member has registered his x nonces $\{R_i\}$ and tree roots $\{\mathsf{mt}_i\}$ with $\mathcal{M}$ which stores these to trace any group signatures. $\mathcal{M}$ also signs these tuples and shares its signature with the member who will then use it to generate a signature. The group signature of a message m by a member is composed of the outputs of the procedure Mod.Sign together with $\mathcal{M}$'s signature $\mathcal{M}.\sigma$ on the tuple $\{\mathsf{mt}_i, R_i\}$. The verification procedure starts with reconstructing mt'_i from $\mathsf{WOTS}^+.\sigma$, idx and R_i thanks to Mod.Verify and finishes with verifying $\mathcal{M}$'s signature for $\{\mathsf{mt}'_i, R_i\}$.

5.4.3 The manager's signature

We now describe the content of the grey square in our example in Figure 5.4. XMSS^{MT} (See Section 5.3) is used for the two following reasons. First, it allows us to sign up to 2^{80} signatures while XMSS is limited due to its inefficient key generation algorithm. Secondly, it produces shorter signature compared to other stateless post-quantum signatures from symmetric primitives such as SPHINCS+ [BHK+19] or Picnic [CDG+17], which is significant as members needs to store x of them. However, we cannot use XMSS^{MT} directly as with XMSS^{MT}, $\mathcal{M}$'s signature has the following format $\mathcal{M}.\sigma_i = \mathsf{XMSS}^{MT}.\sigma = (\{\mathsf{WOTS}^+.\sigma_i, \mathsf{auth}_i\}_{i=1}^{d}, \mathsf{idx}_{MT})$, which means that the element idx_{MT} could link two signatures between them. For example, when a member registers two tuples $\{\mathsf{mt}_1, R_1\}$ and $\{\mathsf{mt}_2, R_2\}$ for which he receives $\mathcal{M}.\sigma_1$ and $\mathcal{M}.\sigma_2$, and if the elements idx^1_{MT} and idx^2_{MT} appear to be in order, then we could guess that $\{\mathsf{mt}_1, R_1\}$ and $\{\mathsf{mt}_2, R_2\}$ belongs to the same member.

Similar to our modification of XMSS (that ensured the anonymity of the signer), the manager also needs to randomly select indexes or two following indexes may belong to the same member such that the signature would be linkable. Given the large number of leaves (up to 2^{80}), it is not efficient to have a database of 2^{80} bits to search which index was used. $\mathcal{M}$ thus performs a pseudorandom permutation (PRP) of its indexes, which means that the first WOTS^+ key would not be associated with the first leaf. Similar to XMSS^{MT}, $\mathcal{M}$ uses a state idx_{MT} set at 0 initially and then updated after each signature ($\mathsf{idx}_{MT} = \mathsf{idx}_{MT} + 1$). But, instead of using $\mathsf{WOTS}^+.\mathsf{sk}_{\mathsf{idx}_{MT}}$ to

TABLE 5.1: A summary of parameters

param	xGS parameters formed of a the height of the modified XMSS tree (h), the height of the XMSS^{MT} hyper-tree h_{MT} and d the number of layers in the hyper-tree for $\mathcal{M}$'s signatures
gpk	group public key which is composed of $\mathcal{M}$'s public key mpk and the list of the revoked members $\mathcal{R}$.
gsk	Group secret key composed of a state Mst, $\mathcal{M}$'s secret key msk, which is XMSS^{MT} secret key, a secret key sk to perform a random permutation, the public list of members **members** and the private database $priv\mathcal{L}$.
usk	Member's secret key composed of a secret seed sd used to determine the position of the leaves, 2^h WOTS$^+$ key pairs (WOTS$^+$.pk, WOTS$^+$.sk) which are the leaves of the modified XMSS trees, x triples $\{\mathsf{mt}_i, R_i, r_i\}$ where mt_i is the root of the modified XMSS tree constructed on the WOTS$^+$ key pairs, R_i and r_i (See Figure 5.5), and two states i for unlinkable signatures and $\mathsf{s_l}$ for linkable signatures which indicates a the WOTS$^+$ key number to use in the next signature. After the joining procedure, x $\mathcal{M}$'s signature $\mathcal{M}.\sigma_i$ is added to it.
upk	Member's public key
σ	The group signature composed WOTS$^+$ signature, the registered element R_i, the authentication path auth (the grey nodes in Figure 5.5), idx which is the direction to follow to construct in order to reconstruct mt_i and the manager signature, which is a XMSS^{MT} signature of the tuple mt_i, R_i.

perform the signature, $\mathcal{M}$ uses WOTS$^+$.sk$_y$, where $y = P(\mathsf{idx}_{MT}, \mathsf{sk})$ and P is a PRP (See Definition 5.2.1) and line 7 of xGS.Join in Algorithm 7. This permutation is illustrated in the simple example presented in Figure 5.4 where the positions of the modified XMSS trees in the hyper-tree do not respect than the order of the trees.

5.4.4 Algorithms presentation

We outline xGS's algorithms, which follow Definition 5.1.1 and whose details can be found in Algorithms 7, 8, 9 and 10 on pages 92, 93, 94 and 95 respectively. All parameters needed for xGS are summarized in Table 5.1 on page 90.

- $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{xGS.Setup}(1^\lambda, \mathsf{param})$: This algorithm takes as input the security parameters λ and the xGS public parameters $\mathsf{param} = (h, h_{MT}, d, b)$, where h is the height of the modified XMSS tree, h_{MT} is the height of the XMSS^{MT} hyper-tree and d is the number of layers in the hyper-tree for $\mathcal{M}$'s signatures and b the public bitmask to generate all trees (manager and members) with $b = \{(b_{l,j}||b_{r,j})\}_{j=1}^{(h+h_{MT})}$ and $b_{l,j}, b_{r,j} \in \{0,1\}^{2\lambda}$. This procedure outputs the group public key gpk, which is composed of $\mathcal{M}$'s public key mpk (the root of the XMSS^{MT} hyper-tree) and the list of the revoked members $\mathcal{R}$ (initially empty), and the group secret key gsk composed of a state Mst, $\mathcal{M}$'s secret key msk, which is XMSS^{MT} secret key, a secret key sk to perform a random permutation, and the private database $priv\mathcal{L}$ (initially empty), which is used to store members' information. This algorithm can be found in Algorithm 7 on page 92.

- $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{xGS.UKeyGen}(1^\lambda, \mathsf{x}, \mathsf{param})$: This procedure, run by a prospective member $\mathcal{U}$, outputs the user public key upk (an identification information), the secret key usk composed of a secret seed sd used to determine the position of

the leaves, 2^h WOTS$^+$ key pairs (WOTS$^+$.pk, WOTS$^+$.sk), which are the leaves of the modified XMSS trees, x triples $\{\mathsf{mt}_i, R_i, r_i\}$ where mt_i is the root of the modified XMSS tree constructed on the WOTS$^+$ key pairs (See Figure 5.5), and two states $\mathsf{s_u}$ for unlinkable signatures and $\mathsf{s_l}$ for linkable signatures, which indicates the number of WOTS$^+$ keys to be used in the next signature. This algorithm can be found in Algorithm 7 on page 92.

- $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gsk} \rangle_{\mathcal{M}} \leftarrow$ xGS.Join$(\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk} \rangle_{\mathcal{M}})$: This is an interactive process between a prospective group member $\mathcal{U}$ and the group manager $\mathcal{M}$ in which $\mathcal{U}$ registers x tuples $\{\mathsf{mt}_i, R_i\}_{i=0}^{\mathsf{x}-1}$ to $\mathcal{M}$, who authenticates these tuples by signing their hash values. This means that $\mathcal{M}$ signs $\mathsf{H}(\mathsf{mt}_i, R_i)$ for all tuples $\{\mathsf{mt}_i, R_i\}_{i=0}^{\mathsf{x}-1}$, which gives x signature $\{\mathcal{M}.\sigma_i\}_{i=0}^{\mathsf{x}-1}$. These signature $\{\mathcal{M}.\sigma_i\}_{i=0}^{\mathsf{x}-1}$ are sent to $\mathcal{U}$ who adds them to his secret key usk. $\mathcal{M}$ registers all $\{R_i\}_{i=0}^{\mathsf{x}-1}$ in his private database $priv\mathcal{L}$ at index $\mathcal{U}$. It is important to note that, in xGS, we omit the authentication process as it was done for the (T-)SGS joining procedure (see Algorithm 6 on page 67). We assume that the communication between $\mathcal{M}$ and any prospective member take place under a secure and authenticated channel. This algorithm can be found in Algorithm 8 on page 93.

- $\perp$ /$(\sigma, \mathsf{usk}) \leftarrow$ xGS.Sign$(m, \mathsf{usk}, \mathsf{param}, \mathsf{link}, \mathsf{x})$: The member $\mathcal{U}$ owns his secret key $\mathsf{usk} = (sd, \{\text{WOTS}^+.\mathsf{pk}_j, \text{WOTS}^+.\mathsf{sk}_j\}_{j=0}^{2^h-1}, \{\mathsf{mt}_i, R_i, r_i, \mathcal{M}.\sigma_i\}_{i=0}^{\mathsf{x}-1}, \mathsf{s_u}, \mathsf{s_l})$. If linkability is not desired (i.e. $\mathsf{link} = false$), the state $\mathsf{s_u}$ is selected and will run Mod.Sign with $R_{\mathsf{s_u}}$ and $r_{\mathsf{s_u}}$ and generate a WOTS$^+$ signature of m with the $\mathsf{s_u}^{th}$ WOTS$^+$ secret key. At the end of the procedure, it outputs a signature $\sigma = (\text{WOTS}^+.\sigma, \mathsf{auth}, R_{\mathsf{s_u}}, \mathsf{idx}, \mathcal{M}.\sigma_{\mathsf{s_u}})$, composed of WOTS$^+$ signature, the registered element $R_{\mathsf{s_u}}$, the authentication path auth (the grey nodes in Figure 5.5), idx, which indicates the position of WOTS$^+$ keys in the modified XMSS tree (to reconstruct $\mathsf{mt}_{\mathsf{s_u}}$), and the manager signature $\mathcal{M}.\sigma_{\mathsf{s_u}}$, which is the XMSS^{MT} signature of $\mathsf{H}(\mathsf{mt}_{\mathsf{s_u}}, R_{\mathsf{s_u}})$. It finishes by updating the state $\mathsf{s_u} \leftarrow \mathsf{s_u} + 1$ to prepare the next unlinkable signature. If $\mathsf{link} = true$, it executes Mod.Sign (see Section 5.4.2) with R_{x} and r_{x} and uses $\mathsf{s_l}$ (which is also incremented at the end of the procedure) as the index to select the $\mathsf{s_l}^{th}$ WOTS$^+$ secret key to perform the signing.

  This signing procedure outputs $\perp$ if either all unlinkable "quota" has been used up (when $\mathsf{link} = false$) or all linkable "quota" has been used up (when $\mathsf{link} = true$). This algorithm can be found in Algorithm 9 on page 94.

- $0/1 \leftarrow$ xGS.Verify$(m, \sigma, \mathsf{gpk}, \mathsf{param})$: This algorithm verifies the validity of a signature σ by reconstructing mt_i based on R_i, WOTS$^+$.σ, auth and idx in σ following the Mod.Verify procedure described in section 5.4.2 and then verifies the $\mathcal{M}$'s signature for $\mathsf{H}(\mathsf{mt}_i, R_i)$. This algorithm can be found in Algorithm 9 on page 94.

- $\mathcal{U}/\perp \leftarrow$ xGS.Open$(\sigma, \mathsf{gpk}, \mathsf{gsk})$: This algorithm, run by $\mathcal{M}$, returns the index $\mathcal{U}$ in the database $priv\mathcal{L}$, where element R_i of σ is stored or $\perp$ if R_i is not in $priv\mathcal{L}$. This algorithm can be found in Algorithm 8 on page 93.

- gpk $\leftarrow$ xGS.Revoke$(\mathcal{U}, \mathsf{gsk}, \mathsf{gpk})$: This algorithm, run by $\mathcal{M}$, adds all elements R_i associated with the revoked member $\mathcal{U}$ to the list of revoked members $\mathcal{R}$. This algorithm can be found in Algorithm 10 on page 95.

- $0/1 \leftarrow \mathsf{xGS.Link}(\mathsf{gpk}, (\sigma_1, m_1), (\sigma_2, m_2), \mathsf{param})$: This algorithm returns 1 if $R_1 = R_2$ belonging to σ_1 and σ_2 respectively and both signatures are valid and 0 otherwise. This algorithm can be found in Algorithm 10 on page 95.

Algorithm 7 xGS algorithms (Part 1)

xGS.Setup

Input: $1^\lambda, \mathsf{param}$
Output: $\mathsf{gpk}, \mathsf{gsk}$

Parse $\mathsf{param} \rightarrow (h, h_{MT}, d, b)$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{XMSS}^{MT}.\mathsf{KeyGen}(1^\lambda)$
$\mathsf{sk} \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\mathcal{R}, priv\mathcal{L} \leftarrow \emptyset$
$\mathsf{gpk} \leftarrow (\mathsf{mpk}, \mathcal{R})$
$\mathsf{Mst} \leftarrow 0$
$\mathsf{gsk} \leftarrow (\mathsf{Mst}, \mathsf{msk}, \mathsf{sk}, priv\mathcal{L})$
return $\mathsf{gpk}, \mathsf{gsk}$

xGS.UKeyGen

Input: $1^\lambda, \mathsf{x}, \mathsf{param}$
Output: $\mathsf{upk}, \mathsf{usk}$

$sd \xleftarrow{\$} \{0,1\}^{2\lambda}$
Parse $\mathsf{param} \rightarrow (h, h_{MT}, d, b)$
for $j = 0$ to $2^h - 1$ **do**
  $(\mathsf{WOTS}^+.\mathsf{pk}_j, \mathsf{WOTS}^+.\mathsf{sk}_j) \leftarrow \mathsf{WOTS.KeyGen}(1^\lambda)$
end for
for $i = 0$ to $\mathsf{x} - 1$ **do**
  $R_i \xleftarrow{\$} \{0,1\}^{2\lambda}$
  $r_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ ▷Use to hide the rest of WOTS^+ keys
  $\mathsf{mt}_i \leftarrow \mathsf{Mod.KeyGen}(\{\mathsf{WOTS}^+.\mathsf{pk}_j, \mathsf{WOTS}^+.\mathsf{sk}_j\}_{j=0}^{2^h-1}, i, R_i, r_i, sd, b, \mathsf{x})$ ▷see Section 5.4.2
end for
$\mathsf{s_l} \leftarrow \mathsf{x} - 1$
$\mathsf{s_u} \leftarrow 0$
$\mathsf{usk} \leftarrow (sd, \{\mathsf{WOTS}^+.\mathsf{pk}_j, \mathsf{WOTS}^+.\mathsf{sk}_j\}_{j=0}^{2^h-1}, \{\mathsf{mt}_i, R_i, r_i\}_{i=0}^{\mathsf{x}-1}, \mathsf{s_u}, \mathsf{s_l})$
$\mathsf{upk} \leftarrow \mathcal{U}$ ($\mathcal{U}$ is the user's identity)
return $\mathsf{upk}, \mathsf{usk}$

5.4.5 Security analysis

Theorem 13 (Correctness)**.** *Let* xGS *be the construction provided in Algorithm 7 and 9 constructed with a cryptographic hash function* H*, a PRF* PRF*, a PRP* P *and a* WOTS^+ *scheme. Then* xGS *achieves Correctness based on Definition 5.1.2.*

Correctness. The correctness xGS is provided directly from the correctness of the symmetric primitives used in the designed xGS. Because all used symmetric primitives are secure, then xGS achieves correctness. □

Algorithm 8 xGS algorithms (Part 2)

xGS.Join

Input: $\langle \mathsf{usk}, \mathsf{upk} \rangle_{\mathcal{U}}, \langle \mathsf{gpk}, \mathsf{gsk} \rangle_{\mathcal{M}}$
Output: $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gsk} \rangle_{\mathcal{M}}$

Parse $\mathsf{upk} \rightarrow \mathcal{U}$
Parse $\mathsf{usk} \rightarrow (sd, \{\mathsf{WOTS}^{+}.\mathsf{pk}_j, \mathsf{WOTS}^{+}.\mathsf{sk}_j\}_{j=0}^{2^h-1}, \{\mathsf{mt}_i, R_i, r_i\}_{i=0}^{\mathsf{x}-1}, \mathsf{s_u}, \mathsf{s_l})$
Parse $\mathsf{gpk} \rightarrow (\mathsf{mpk}, \mathcal{R})$
Parse $\mathsf{gsk} \rightarrow (\mathsf{Mst}, \mathsf{msk}, \mathsf{sk}, priv\mathcal{L})$
$\mathcal{U}$: sends x tuples $\{\mathsf{mt}_i, R_i\}_{i=0}^{\mathsf{x}-1}$ to $\mathcal{M}$
$\mathcal{M}$: adds $\{R_i\}_{i=0}^{\mathsf{x}-1}$ in $priv\mathcal{L}$ at index $\mathcal{U}$
$\mathcal{M}$:
for $i = 0$ to $\mathsf{x} - 1$ **do**
    $\mathsf{idx}_{MT} \leftarrow P(\mathsf{Mst}, \mathsf{sk})$ ▷see Section 5.4.3
    $\mathcal{M}.\sigma_i \leftarrow \mathsf{XMSS}^{MT}.\mathsf{Sign}(\mathsf{H}(R_i, \mathsf{mt}_i), \mathsf{msk}, \mathsf{idx}_{MT})$
    $\mathsf{Mst} \leftarrow \mathsf{Mst} + 1$
end for
$\mathcal{M}$ sends $\{\mathcal{M}.\sigma_i\}_{i=0}^{\mathsf{x}-1}$ to $\mathcal{U}$
$\mathcal{U}$: $\mathsf{usk} \leftarrow (\{\mathsf{WOTS}^{+}.\mathsf{pk}_j, \mathsf{WOTS}^{+}.\mathsf{sk}_j\}_{j=0}^{2^h-1}, \{\mathsf{mt}_i, R_i, r_i, \mathcal{M}.\sigma_i\}_{i=0}^{\mathsf{x}-1}, \mathsf{s_u}, \mathsf{s_l})$
$\mathcal{M}$: $\mathsf{gsk} \leftarrow (\mathsf{Mst}, \mathsf{msk}, \mathsf{sk}, priv\mathcal{L})$
return $\langle \mathsf{usk} \rangle_{\mathcal{U}}, \langle \mathsf{gsk} \rangle_{\mathcal{M}}$

xGS.Open

Input: $\sigma, \mathsf{gpk}, \mathsf{gsk}$
Output: $\mathcal{U}, \perp$

Parse $\sigma \rightarrow (\mathsf{WOTS}^{+}.\sigma, R_i, \mathsf{auth}, \mathsf{idx}, \mathcal{M}.\sigma_i)$
Parse $\mathsf{gpk} \rightarrow (\mathsf{mpk}, \mathcal{R})$
Parse $\mathsf{gsk} \rightarrow (\mathsf{Mst}, \mathsf{msk}, \mathsf{sk}, priv\mathcal{L})$
if $R_i \in priv\mathcal{L}$ **then**
    Search the index $\mathcal{U}$ associated with R_i in $priv\mathcal{L}$
    return $\mathcal{U}$
end if
return $\perp$

Algorithm 9 xGS algorithms (Part 3)

xGS.Sign

Input: $m, \mathsf{usk}, \mathsf{param}, \mathsf{link}, \mathsf{x}$
Output: $\perp / \sigma, \mathsf{usk}$

Parse $\mathsf{usk} \rightarrow (sd, \{\mathsf{WOTS^+.pk}_j, \mathsf{WOTS^+.sk}_j\}_{j=0}^{2^h-1}, \{\mathsf{mt}_i, R_i, r_i, \mathcal{M}.\sigma_i\}_{i=0}^{\mathsf{x}-1}, \mathsf{s_u}, \mathsf{s_l})$
Parse $\mathsf{param} \rightarrow (h, h_{MT}, d, b)$
if !link **then**
  if $\mathsf{s_u} == \mathsf{x} - 1$ **then**
    return $\perp$ ▷All possible unlinkable signatures have been done
  end if
  $(\mathsf{WOTS^+}.\sigma, \mathsf{auth}, R_{\mathsf{s_u}}, \mathsf{idx}) \leftarrow \mathsf{Mod.Sign}(m, R_{\mathsf{s_u}}, r_{\mathsf{s_u}}, \mathsf{s_u}, \mathsf{s_u}, sd, \mathsf{x}, b, \{\mathsf{WOTS^+.pk}_j, \mathsf{WOTS^+.sk}_j\}_{j=0}^{2^h-1})$ ▷We construct the $\mathsf{s_u}^{th}$ tree and use the $\mathsf{s_u}^{th}$ $\mathsf{WOTS^+}.sk$, Mod.Sign is presented in Section 5.4.2
  $\sigma \leftarrow (\mathsf{WOTS^+}.\sigma, R_{\mathsf{s_u}}, \mathsf{auth}, \mathsf{idx}, \mathcal{M}.\sigma_{\mathsf{s_u}})$
  $\mathsf{usk.s_u} \leftarrow \mathsf{s_u} + 1$
  return σ, usk
else
  if $\mathsf{s_l} == 2^h - 1$ **then**
    return $\perp$ ▷All $\mathsf{WOTS^+}$ keys have been used
  end if
  $(\mathsf{WOTS^+}.\sigma, \mathsf{auth}, R_{\mathsf{x}-1}, \mathsf{idx}) \leftarrow \mathsf{Mod.Sign}(m, R_{\mathsf{x}-1}, r_{\mathsf{x}-1}, \mathsf{x}-1, \mathsf{s_l}, sd, \mathsf{x}, b, \{\mathsf{WOTS^+.pk}_j, \mathsf{WOTS^+.sk}_j\}_{j=0}^{2^h-1})$ ▷We construct the $\mathsf{x}-1^{th}$ tree and use the $\mathsf{s_l}^{th}$ $\mathsf{WOTS^+}.sk$
  $\sigma \leftarrow (\mathsf{WOTS^+}.\sigma, R_{\mathsf{x}}, \mathsf{auth}, \mathsf{idx}, \mathcal{M}.\sigma_{\mathsf{x}})$
  $\mathsf{usk.s_l} \leftarrow \mathsf{s_l} + 1$
  return σ, usk
end if

xGS.Verify

Input: $m, \sigma, \mathsf{gpk}, \mathsf{param}$
Output: $0/1$

Parse $\sigma \rightarrow (\mathsf{WOTS^+}.\sigma, R_i, \mathsf{auth}, \mathsf{idx}, \mathcal{M}.\sigma_i)$
Parse $\mathsf{gpk} \rightarrow (\mathsf{mpk}, \mathcal{R})$
Parse $\mathsf{param} \rightarrow (h, h_{MT}, d, b)$
if $R_i \in \mathcal{R}$ **then**
  return 0
end if
$\mathsf{mt}'_i \leftarrow \mathsf{Mod.Verify}(m, \mathsf{WOTS^+}.\sigma, \mathsf{auth}, R_i, \mathsf{idx}, b)$
return $\mathsf{XMSS}^{MT}.\mathsf{Verify}(\mathsf{H}(\mathsf{mt}'_i, R_i), \mathcal{M}.\sigma_i, \mathsf{mpk})$

Algorithm 10 xGS algorithms (Part 4)

xGS.Revoke

Input: $\mathcal{U}$, gsk, gpk
Output: gpk

1: Parse gsk $\rightarrow$ (Mst, msk, sk, $priv\mathcal{L}$)
2: Parse gpk $\rightarrow$ (mpk, $\mathcal{R}$)
3: add all R_i's associated with $\mathcal{U}$ into $\mathcal{R}$
4: **return** gpk = (mpk, $\mathcal{R}$) ▷We remark that $\mathcal{R}$ should be authenticated by $\mathcal{M}$. It can be signed by $\mathcal{M}$, or put to some authenticated public accessible location

xGS.Link

Input: gpk, $(m_1, \sigma_1), (m_2, \sigma_2)$,
Output: 0/1

1: Parse gpk $\rightarrow$ (mpk, $\mathcal{R}$)
2: Parse $\sigma_1 \rightarrow$ (WOTS$^+$.σ_1, R_1, auth$_1$, idx$_1$, $\mathcal{M}.\sigma_1$)
3: Parse $\sigma_2 \rightarrow$ (WOTS$^+$.σ_2, R_2, auth$_2$, idx$_2$, $\mathcal{M}.\sigma_2$)
4: Parse param $\rightarrow (h, h_{MT}, d, b)$
5: **if** $0 =$ xGS.Verify$(m_1, \sigma_1,$ gpk, param$)$ or $0 =$ xGS.Verify$(m_2, \sigma_2,$ gpk, param$)$ **then**
6: **return** 0
7: **end if**
8: **return** $R_1 == R_2$

Theorem 14 (Non-frameability)**.** *Let* xGS *be the construction provided in Algorithm 7 and 9 constructed with a cryptographic hash function* H*, a PRF* PRF*, a PRP* P *and a* WOTS$^+$ *scheme. Then* xGS *achieves Non-frameability based on Definition 5.1.3.*

Non-frameability. We use a game-based approach to show that xGS provides non-frameability. Let V_i be the event that $\mathcal{A}$ wins the GAME$_i$ and all elements denoted with $*$ are forgeries generated by $\mathcal{A}$. Based on the game $\mathbf{Exp}^{Frame}_{\mathcal{A},\mathsf{GS}}(\lambda)$ presented in Figure 5.1), a PPT adversary $\mathcal{A}$ can access the private information of corrupted members, but $\mathcal{A}$ needs to frame an honest member $\mathcal{U}$ to win the non-frameability game. A signature σ is defined as follows (WOTS$^+$.σ, R_i, auth, idx, $\mathcal{M}.\sigma_i$).
GAME$_1$: The original non-frameability experiment $\mathbf{Exp}^{Frame}_{\mathcal{A},\mathsf{GS}}(\lambda)$ (Figure 5.1) executed with an adversary $\mathcal{A}$.
GAME$_2$: Same as the previous game but the adversary $\mathcal{A}$ replaces the XMSS^{MT} signature of $\mathcal{M}$ with its forgery $\mathcal{M}.\sigma^*_i$ in the group signature. This decreases the probability of $\mathcal{A}$ to win thanks to the EU-CMA security of XMSS^{MT} (see definition 5.3.2 and 2.4.2). Therefore, it is indistinguishable from the previous game and we have $|\Pr[V_2] - \Pr[V_1]| \leq \mathbf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) < \mathsf{negl}(\lambda)$.
GAME$_3$: Same as GAME$_2$ but $\mathcal{A}$ creates his own path idx* and R^*_i in order to generate mt$^*_i$ (modified XMSS root, line 7 of xGS.Sign in Algorithm 9) such that H(mt$_i, R_i$) = Hmt$^*_i, R^*_i$) signature for the bottom level of the $\mathcal{M}$ hyper-tree. This advantage of $\mathcal{A}$ only decreases because of the collision resistance property of H. Indeed, H is a secure cryptographic hash function which makes it computationally infeasible to find a collision or to inverse the H according to Definition 2.2.1. Therefore, this is indistinguishable from the previous game and we have $|\Pr[V_3] - \Pr[V_2]| < \mathsf{negl}(\lambda)$.

This concludes the proof and demonstrates that $\mathbf{Adv}^{Frame}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

□

Theorem 15 (Anonymity). *Let* xGS *be the construction provided in Algorithm 7, 8, 9 and 10 constructed with a cryptographic hash function* H*, a PRF* PRF*, a PRP P and a* WOTS$^+$ *scheme. Then,* xGS *achieves Anonymity based on Definition 5.1.4.*

Anonymity. We use a game-based approach to show that xGS provides anonymity (see Definition 5.1.4). Let V_i be the event that $\mathcal{A}$ wins the GAME_i and elements denoted with $*$ are forgeries generated by $\mathcal{A}$. Based on the game $\mathbf{Exp}^{Anon-b}_{\mathcal{A},\mathsf{GS}}(\lambda)$ presented in Figure 5.1, an adversary $\mathcal{A}$ can access the private information of corrupted members and signature generated by the *Sign* oracle game. A signature σ is defined as follows $(\mathsf{WOTS}^+.\sigma, R_i, \mathsf{auth}, \mathsf{idx}, \mathcal{M}.\sigma_i)$. The elements related to the signer's identity are the WOTS$^+$ public key which appear in the authentication path auth, R_i, idx and idx_{MT} in the manager signature $\mathcal{M}.\sigma_i$.
GAME_1: The original anonymity experiment $\mathbf{Exp}^{Anon-b}_{\mathcal{A},\mathsf{GS}}(\lambda)$ (Figure 5.1) executed with an adversary $\mathcal{A}$.
GAME_2: Same as the previous game but the adversary $\mathcal{A}$ takes the first element of the authentication auth and invert it to find if the corresponding WOTS$^+$ key has already been used by a user. This decreases the probability of $\mathcal{A}$ to win for the following reasons. We addressed this risk by adding an extra level between WOTS$^+$ public key coupled with a secret r_i (See Remark 5) and the element of auth, hence $\mathcal{A}$ needs to inverse the cryptographic hash function, which is infeasible (see Definition 2.2.1). Both idx_{MT} and idx could leak information about the signer, but xGS solves this issue by permuting the index idx_{MT} to generate $\mathcal{M}.\sigma$. Therefore, the order of the indexes of x $\mathcal{M}$'s signatures that each member possesses will not be in order and cannot leak any information, and therefore provides anonymity. This means that this is indistinguishable from the previous game and we have $|\Pr[V_2] - \Pr[V_1]| < \mathsf{negl}(\lambda)$.
This concludes the proof and shows that $\mathbf{Adv}^{Anon}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$. □

Theorem 16 (Traceability). *Let* xGS *be the construction provided in Algorithm 7, 8, 9 and 10 constructed with a cryptographic hash function* H*, a PRF* PRF*, a PRP P and a* WOTS$^+$ *scheme. Then,* xGS *achieves Traceability based on Definition 5.1.5.*

Traceability. We use a game-based approach to show that xGS provides traceability (see Definition 5.1.5). Let V_i be the event that $\mathcal{A}$ wins the GAME_i and elements denoted with $*$ are forgeries generated by $\mathcal{A}$. Based on the game $\mathbf{Exp}^{Trace}_{\mathcal{A},\mathsf{GS}}(\lambda)$ presented in Figure 5.1, an adversary $\mathcal{A}$ can access the private information of corrupted members, and signature generated by the *Sign* oracle game. A signature σ is defined as follows $(\mathsf{WOTS}^+.\sigma, R_i, \mathsf{auth}, \mathsf{idx}, \mathcal{M}.\sigma_i)$. In order to break the traceability, $\mathcal{A}$ needs to generate a valid signature with a different R_i^* which has not been registered by the central authority $\mathcal{M}$.
GAME_1: The original traceability experiment $\mathbf{Exp}^{Trace}_{\mathcal{A},\mathsf{GS}}(\lambda)$ (Figure 5.1) executed with an adversary $\mathcal{A}$.
GAME_2: Same as the previous game but the adversary $\mathcal{A}$ replaces the element R_i by a random element R_i^*. This decreases the probability of $\mathcal{A}$ to win thanks to the EU-CMA security of XMSS^{MT} (see definition 5.3.2 and 2.4.2). Indeed, The only way to win this game is to forge the XMSS^{MT} signature of $\mathcal{M}$ which was proven to be secure in [HBG+18]. Therefore, we have $|\Pr[V_2] - \Pr[V_1]| \leq \mathbf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) < \mathsf{negl}(\lambda)$.
GAME_3: Same as the previous game but the adversary $\mathcal{A}$ replaces the element auth by auth^* in order to have the verification procedure computing mt_i^* such that $\mathsf{H}(\mathsf{mt}_i, R_i) = \mathsf{H}(\mathsf{mt}_i^*, R_i^*)$. If this equality holds, $\mathcal{A}$ wins the game. But, this decreases the probability of $\mathcal{A}$ to win thanks to collision the resistance property of the cryptographic hash function H (see Definition 2.2.1), therefore $|\Pr[V_3] - \Pr[V_2]| < \mathsf{negl}(\lambda)$.
This shows that $\mathbf{Adv}^{Trace}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$. □

Theorem 17 (User-Controlled Linkability)**.** *Let* xGS *be the construction provided in Algorithm 7, 8, 9 and 10 constructed with a cryptographic hash function* H*, a PRF* PRF*, a PRP* P *and a* WOTS$^+$ *scheme. Then,* xGS *achieves User-Controlled Linkability based on Definition 5.1.6.*

User-Controlled Linkability. We use a game-based approach to show that xGS provides user-controlled linkability (see Definition 5.1.6). This aims to prove that it is computationally infeasible to generate an unlinkable signature if the signature is meant to be linkable. Let V_i be the event that $\mathcal{A}$ wins the GAME_i and the elements denoted with $*$ are forgeries generated by $\mathcal{A}$.
GAME_1: The original user-controlled linkability experiment $\mathbf{Exp}^{Link}_{\mathcal{A},\mathsf{GS}}(\lambda)$ (Figure 5.1) run with an adversary $\mathcal{A}$.
GAME_2: Same as the previous game but the adversary $\mathcal{A}$ replaces the element R_{x} by a random element R^*_{x}. This decreases the probability of $\mathcal{A}$ to win thanks to the EU-CMA security of XMSS^{MT} (see definition 5.3.2 and 2.4.2). Indeed, The only way to win this game is to forge the XMSS^{MT} signature of $\mathcal{M}$ which was proven to be secure in [HBG+18]. Therefore, $|\Pr[V_2] - \Pr[V_1]| \leq \mathbf{Adv}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) < \mathsf{negl}(\lambda)$.
GAME_3: Same as the previous game but the adversary $\mathcal{A}$ replaces the element auth by auth* in order to have the verification procedure computing mt^*_i such that $\mathsf{H}(\mathsf{mt}_i, R_i) = \mathsf{H}(\mathsf{mt}^*_i, R^*_i)$. If this equality holds, $\mathcal{A}$ wins the game. But, this decreases the probability of $\mathcal{A}$ to win thanks to the collision resistance property of the cryptographic hash function H (see Definition 2.2.1), therefore $|\Pr[V_3] - \Pr[V_2]| < \mathsf{negl}(\lambda)$.

This concludes the proof and demonstrates that $\mathbf{Adv}^{Link}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

□

5.4.6 Further discussions

Reason for multiple trees

The reason why members do not simply use WOTS$^+$ to sign (where each member registers their WOTS$^+$ keys to $\mathcal{M}$) as an alternative to having x modified XMSS trees lies in the user-controlled linkability. A WOTS$^+$ key pair is secure only if used once, and linking two signatures would imply reusing the same key twice, which would break the scheme.

Benefit of XMSS

It is important to notice that the modified XMSS tree could be replaced by a single stateless signature (e.g. SPHINCS+ [BHK+19] or Picnic [CDG+17]). An alternative (and naive) solution is to have x multi-use signature key pairs (e.g. XMSS, SPHINCS+ or Picnic) instead of having x modified XMSS trees. In terms of functionality and security, it is the same as our xGS. However, our construction is much more efficient (both in space and computation) than this naive construction. XMSS is the fastest among the three candidates (in signing and verification) though the key generation process is costly and the memory requirement is large (due to the large number (e.g. 2^h) WOTS$^+$ keys required for each XMSS instance). If we want to have x unlinkable signatures, we need to generate (and store) $\mathsf{x} \cdot 2^h$ WOTS$^+$. In fact, we observe that we can cleverly manage using only 2^h WOTS$^+$ but for x (not one!) (modified) XMSS instances. This comes to the idea of our construction.

TABLE 5.2: Influence of d (i.e. number of layer in the hyper-tree) on the performances with a modified XMSS tree of height $h = 20$ which means that members can generate a maximum of 2^{20} signature. $\mathcal{M}$'s hyper-tree height is $h_{MT} = 80$ which means that $N = 2^{60}$ if $\mathcal{M}$ has an unbounded memory or $N = 2^{20}$ in practice because of the security of the PRP which remains secures for 2^{40} execution [HR10].

		$\lvert\sigma\rvert$	$\lvert\mathcal{M}.\sigma\rvert$	xGS.Sign (ms)			xGS.Verify	xGS.Setup
	d	(KB)	(KB)	Total	Offline	Online	(ms)	(s)
xGS	4	13.96	11.78	1496	1495	1	10.1	3561.61
	5	16.14	13.32				11	223.16
	8	22.54	19.74				17.5	3.38
	10	26.86	24.04				21.4	0.99

Influence of x

An interesting feature of xGS is that it can be used as a linkable-only group signature if $\mathsf{x} = 1$. It could then be used in applications related to e-voting or it can be implemented as a traditional group signature (no linkability) if $\mathsf{x} = 2^h$.

5.5 Evaluation

This section presents a full evaluation of xGS's theoretical and practical performances and compares it with the current state-of-the-art. We implemented xGS in Java and used SHA256 as hash function. We implemented PRF with HMAC-SHA256 and PRP using the Feistel construction [HR10] with a block size of $n = h_{MT}/2$ using AES. According to [HR10], a Feistel-based permutation is secure against Chosen Ciphertext Attack (CCA) for $2^{n(1-1/r)}$, where r is the number of rounds in the PRP. Our implementation fixed $r = 100$ to enable at least $2^{h_{MT}/2}$ evaluations. The XMSS^{MT} part for the manager's signature was implemented with the Bouncy Castle crypto library [otBCI20]. We evaluated the performance with various parameters. Table 5.2 and 5.3 show the average results of 5 experiments running on a machine with an Xeon-Gold-6150 @ 2.70GHz with 16 GB memory.

5.5.1 Signature size

The signature size can be represented as follows: $|\sigma| = h \cdot |\mathsf{Hash}| + |\mathsf{WOTS}^+.\sigma| + |R_i| + |\mathsf{idx}| + |\mathcal{M}.\sigma_i|$, where $|\mathsf{Hash}|$ is the output's size of the cryptographic hash function H, h is the height of the modified XMSS tree (See Figure 5.5), $|\mathsf{WOTS}^+.\sigma|$ is the size of the WOTS^+ signature, $|R_i|$ is the size of the nonce R_i, $|\mathsf{idx}|$ is the size of the index, which is a h-bits string and $|\mathcal{M}.\sigma_i|$ is the size of $\mathcal{M}$'s signature. Table 5.4 compares xGS's signature size with the current state-of-the-art.

Table 5.2 shows our scheme's performance in four instances with different values of d in $\mathcal{M}$'s hyper-tree (see Figure 5.3). The height of the XMSS tree is fixed to 20, giving members the ability to generate up to 2^{20} signatures (linkable or not) and $\mathcal{M}$' hyper-tree is fixed to $h_{MT} = 2^{80}$. Given the security of the PRP P, the xGS instance can support $N = 2^{20}$ group members while having a signature size between 13.96 and 26.86KB (depending on d), clearly outperforming the constructions [EBM18, BEF19] by at least 740KB for a group with 2^{20} members. While their signature size continues to increase logarithmically with the number of members, our signature size can in

TABLE 5.3: Influence of x on the performance of xGS with $h = 20$, $h_{MT} = 80$ and $d = 5$ for 128 bits of post-quantum security. Running time complexity for xGS.UKeygen and xGS.Join is asymptotically $\mathcal{O}(x)$.

xGS	x	Maximum |usk| (KB)	xGS.UKeyGen(s)	xGS.Join(s)		
				Total	Offline	Online
	10	133.53	3583	123.57	123.54	0.03
	100	1332.03	5733	1235.7	1235.4	0.3
	1000	13320.03	5568	12357	12354	3
	10000	133200.03	23613	123570	123540	30

theory remain constant until 2^{60} members if $\mathcal{M}$ has unlimited memory capacity. Another asset over [BEF19] and [KKW18] remains that a new member joining the group does not influence the rest of the members. In both [BEF19] and [KKW18], every new member modifies the group public key, and therefore the rest of the group must undergo an interactive update process anytime a new members join the group while xGS does not need any update process. Although [BLS+19] and [EBM18] outperform our signature size, we provide better applicability as explained in Section 5.5.4. When compared with both lattice constructions [DPLS18] and [EZS+19], xGS' generates a signature smaller of at least 530KB than [DPLS18] and 10KB than [EZS+19]. Additionally, xGS has the advantage to have a signature size independent from the number of group members, which is not the case for [EZS+19]. Table 5.4 presents the signature size for $d = 5$.

5.5.2 usk's size

All elements composing the secret key are summarized in Table 5.1. usk's size depends mainly on the storage of x different $\mathcal{M}$'s signatures ($\mathcal{M}.\sigma$), making the choice of $\mathcal{M}$'s signature scheme to generate them crucial. Note that $\mathcal{M}$'s signature can be generated by stateless digital signature schemes like SPHINCS+ [BHK+19] or Picnic [CDG+17], but this will add at least 20KB on each $\mathcal{M}$'s signature size and increase the user secret key size as $|\mathsf{usk}| = \mathcal{O}(x \cdot |\mathcal{M}.\sigma|)$. The usk's size may be a limitation of our scheme, but members only need to store x $\mathcal{M}.\sigma$s, both states $\mathsf{s_u}$ and $\mathsf{s_l}$, and the secret seed *sd* while the rest can be recomputed from a seeded PRF. Table 5.3 shows the influence of x over the size of usk.

Moreover, the use of XMSS^{MT} as $\mathcal{M}$ signature offers a possible save in storage on the member sides. As explained, each group member stores x manager's signatures but it does not mean that they need to save x complete $\mathcal{M}.\sigma$. If we take a close look into $\mathcal{M}.\sigma = (\{\mathsf{WOTS}^+.\sigma_i, \mathsf{auth}\}_{i=1}^{d}, \mathsf{idx}_{MT})$ and in particular the element idx_{MT}, which is a h_{MT}-bits string indicating the path from the bottom the hyper-tree to the group public key gpk. If we have two $\mathcal{M}$' signatures sharing an identical prefix of length y for idx_{MT}, this implies that y elements of path auth are identical. Therefore, y elements of the path auth do not need to be stored twice. For a post-quantum security level of 128-bits, it saves $y \cdot 256$ bits for each $\mathcal{M}.\sigma$ sharing y elements in auth. Moreover, if $y = (h/d) \cdot z$ with $z \in \mathbb{N}$, z WOTS^+, signatures are not required to be stored.

5.5.3 Computation efficiency

Table 5.2 and 5.3 demonstrate the performance of our algorithms. For the signing procedure, we assume that members have 34MB available to store elements of the modified XMSS tree. 34MB allow storing 2^h WOTS$^+$.pk, which are composed of 256 bits (for 128-bits post-quantum security level). xGS members can sign messages in 1496ms, but it is important to notice that 1495ms are offline computations and can be performed in advance. This means that the online computation requires only 1ms as the sole online operation is the WOTS$^+$ signature of the message m and the rest can be pre-computed. The states $\mathsf{s_u}$ (unlinkable) and $\mathsf{s_l}$ (linkable) indicate which nonce R_i and $\mathcal{M}$'s signature will be used in the next signature and which is the next WOTS$^+$ key pair from which the position idx in the tree and the authentication path auth are defined. All elements of a signature σ except WOTS$^+$.σ can thus be computed in advance. The extra memory required for offline computations is $|\mathsf{auth}| + |\mathsf{idx}| = h * \lambda + \log(h)$ bits while the rest is already stored in usk. Our verification procedure is also efficient (less than 22ms for all settings). The setup performance improves as the number of layer d in the XMSS^{MT} tree of $\mathcal{M}$ increases given that the size of the internal XMSS trees of $\mathcal{M}$'s hyper-tree are smaller and thus faster to compute.

The joining procedure efficiency depends on the memory capacity for the offline operation (See Table 5.3). For the same reason, $\mathcal{M}$ can prepare a set of next signatures before receiving any joining requests (from its state Mst) with additional memory capacity requirement for offline computations as $z*$XMSS^{MT}.σ, where z is the number of pre-prepared signatures. In practice, the xGS manager will require a memory of $z * 13.32$KB. x (maximum number of possible unlinkable signatures) only influences xGS.UKeyGen and xGS.Join procedures by increasing their running-time linearly. xGS's advantage over most other symmetric-based works is that the signing and verification algorithms do not depend on the number of group members, which is the case for [KKW18, BEF19, EBM18]. Although the efficiency of DGM [BLS+19] algorithms also does not depend on the group size, its verification procedure requires an interaction with the manager and their joining procedure is less efficient than ours.

5.5.4 Comparative summary with the current state-of-the-art

In the previous sections, we compared the strengths of our approach with the current state-of-the-art in terms of signature size and algorithm efficiency. Table 5.4 situates our work in the current state-of-the-art post-quantum group signatures. It compares xGS with the two current state-of-the-art lattice-based group signatures [DPLS18, EZS+19] and all symmetric-based constructions [KKW18, BEF19, EBM18, BLS+19]. The main advantage of xGS is its competitive signature size and this advantage is so significant that justifies its stateful feature.

As presented in Section 5.5.1, when it comes to signature size, both G-Merkle [EBM18] and DGM [BLS+19] appear to be particularly competitive. In this section, we highlight that xGS has greater applicability than both G-Merkle and DGM. xGS outperforms the static G-Merkle because of its dynamicity feature. Moreover, G-Merkle offers a limited number of possible signatures and each group member is very limited in the number of maximum messages that he can sign. In addition, G-Merkle can only deal with small groups. The maximum group size proposed is 64 members and each member can generate at most 4096 signatures. In comparison, xGS supports up to 2^{40} members who can sign up to 2^{20} messages. xGS also outperforms the dynamic DGM when it comes to applicability. Even if DGM [BLS+19] claims an efficient setup process, their joining process is outperformed by its xGS counterpart. Indeed, DGM's joining process needs a new complete XMSS tree (including WOTS$^+$ keys which count

for 99% of the computation for an XMSS tree) computed for each WOTS^+ key pairs request while in xGS, the WOTS^+ keys only need to be computed once during the member key generation algorithms. Another applicability issue is that the current DGM verification process is interactive. The verifier needs to check with the central authority $\mathcal{M}$ for the validity of an element named Fallback key for every signature. While the technique used in xGS may be used to prove the validity of the so-called Fallback keys non-interactively, a complete security analysis should be conducted to confirm this assumption. But, even if we apply this technique to DGM, their signature size will be increased by 13.3KB, which gives a signature size similar to xGS.

The most important advantage of xGS over DGM/G-Merkle is the level of trust in the manager. In DGM/G-Merkle, the WOTS^+ keys are generated by the central authority $\mathcal{M}$, which means that $\mathcal{M}$ can frame any honest group member. xGS avoids this issue because each member generates his secret keys. Because of this difference in the design, the xGS manager cannot frame the signature of an honest member even if he colludes with malicious group members. The final advantage of xGS over DGM/G-Merkle is the possible optimization of members' secret key size usk. Indeed, each member can reduce its size by using a seed and a PRF to generate the WOTS^+ secret keys. This optimization would not be possible in DGM/G-Merkle as members' secret keys are generated by the central authority and members need to store this information. Therefore, DGM/G-Merkle performances rely more on memory than xGS.

The results discussed previously and Table 5.2 show that xGS, with the parameters $h_{MT} = 80$ and $d = 5$, offers the best compromise between algorithms' efficiency and signature size. Indeed, if we set $d = 4$, then the setup process is at least 10 times slower and the joining process would also suffer a loss of efficiency. If $d = 8$ or $d = 10$, the xGS.Setup and xGS.Join algorithms would be more efficient, but at least 6KB of additional data would be added to the signature and the size of members' secret keys would increase by a factor $\mathsf{x} \cdot 6\text{KB}$.

To summarize, xGS increases the applicability of post-quantum group signature schemes constructed on symmetric primitives. xGS can deal with large groups without requiring interaction in the verification procedure. xGS counters its stateful feature with a competitive signature size which clearly outperforms existing lattice and symmetric constructions. Finally, xGS is the first post-quantum secure construction providing user-controlled linkability, which offers a choice of a trade-off between the privacy level and members' secret key size.

5.6 Chapter conclusion

In this chapter, we defined formally the concept of group signature with user-controlled linkability in Section 5.1. We then explained how we modified the construction of XMSS Tree in order to create a group signature. A random nonce was introduced in the tree construction, which results in a different roots build from the set of WOTS^+ keys. In xGS, each group member is constructing x modified XMSS trees, which are registered to the group manager and gives each group member the ability to generate x unlikable signatures. xGS solves the two main issues of our first group signature DGM. First, its verification procedure is non-interactive while DGM requires an interaction with the central authority. Second, the level of trust in $\mathcal{M}$ is reduced in xGS. Indeed, the xGS manager cannot frame an honest member, in contrast to DGM $\mathcal{M}$ who may be able to frame a honest member and generate a signature on his behalf.

Due of the research outputs of the previous chapter, we decided to focus on having a non-interactive verification procedure even if it is meant to not provide the stateless feature. Despite xGS not being stateless, we are convinced that xGS counters its stateful feature with a competitive signature size which clearly outperforms the current state-of-the-art of lattice-based and symmetric-based group signatures. Finally, xGS is the first post-quantum secure construction providing user-controlled linkability, which offers a new choice of a trade-off between the privacy level and members' secret key size. xGS offers the possibility to be a linkable group signature if $\mathsf{x} = 1$ and so it finds application in blockchain-based schemes including e-voting or cryptocurrency. Or if $\mathsf{x} = 2^h$, it is a "traditional" group signature. Table 5.4 demonstrates the final contributions related to Research Objective O1 and situates them in the current state-of-art of post-quantum group signatures. To summarize, xGS increases the applicability of post-quantum group signature schemes constructed on symmetric primitives as it can be used as "traditional" group signature when $\mathsf{x} = 2^h$ or a fully linkable group signature if $\mathsf{x} = 1$. xGS can also handle large groups with a competitive signature size.

Table 5.4 summarizes the contributions related to Research Objective O1 (see Chapter 1.3.2 on page 13).

TABLE 5.4: Comparison of different post-quantum group signatures for 128-bits of post-quantum security. N denotes the number of members in the group, N.A.=Non Applicable, B=Maximum possible signatures per member, low= The group manager $\mathcal{M}$ cannot frame a signature, high=$\mathcal{M}$ can frame a signature, x= number of unlinkable signatures that can be generated by each user

Schemes		[DPLS18]	[EZS+19]	[BEF19]	[KKW18]	[EBM18]	DGM	SGS	T-SGS	xGS
$\|\sigma\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
$\|\sigma\|$ (KB)	$N = 2^7$	581	39	1370	315	2.72	2.82	33	359	16.14
	$N = 2^{10}$	581	54	1850	418	N.A	2.82	33	501	16.14
	$N = 2^{20}$	581	140	3450	770	N.A	2.82	33	970	16.14
$\|\mathsf{usk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log B)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\mathsf{x})$
$\|\mathsf{usk}\|$ (KB)		146	0.059	$0.032 \log N$	$0.032 \log N$	$2.2 \log B$	$2.8 \log B$	275	$275 + 0.032 \log N$	$\mathsf{x} \cdot 13.13$
$\|\mathsf{gpk}\|$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
$\|\mathsf{gpk}\|$ (KB)		123	$N \cdot 11.1$	0.032	0.032	0.032	0.032	0.032	0.032	0.032
Sign (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
Sign (ms)[a]		450	No data	No data	3000	N.A	1	1244	7383	1496
Verify (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
Verify (ms)[a]		169	No data	No data	3000	N.A	Interactive	0.1	3886	11
PQ canditate		Lattice	Lattice	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric
Stateless		Yes	Yes	Yes	Yes	No	No	Yes	Yes	No
Dynamic		Static	Partially	Static	Fully	Static	Fully	Fully	Fully	Fully
Traceable		Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes
Non-interactive		Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Level of Trust in $\mathcal{M}$		Low	Low	Low	Low	High	High	High	High	Low

[a]The times provided are for $N = 2^{10}$ and taken from the original papers and were implemented in different programming languages and executed on different types of machines

Chapter 6

ID-based Ring Signature from Symmetric Primitives

Ring signatures [RST01] are currently considered one of the most valuable cryptographic primitives to ensure privacy. They allow a member of a group (i.e. ring) to anonymously sign a message on behalf of a group in a spontaneous manner. This spontaneity allows signers to form a group of their own choice and to generate an anonymous signature. According to their great promise in providing authenticity and anonymity, ring signatures have attracted a lot of interest from the research community.

We decided to follow this interest and investigate the possible design of ring signature based on symmetric primitives. However, we noticed that designing a "traditional" ring signature in public key infrastructures (PKI), that it would be difficult to outperform or improve the work of Katz et al. [KKW18]. Therefore, we decided to follow another approach by designing an ID-based ring signature. ID-based cryptographic primitives suppress the need for certificates in PKI and therefore, it improves the spontaneity of ring signature. As discussed in Chapter 1.1.4, spontaneity is a major advantage that ring signatures have over group signatures. In this chapter, we propose a generic construction for post-quantum ID-based ring signatures (IDRS) based on symmetric-key primitives from which we derive the first two constructions of IDRS.

This chapter is dedicated to the presentation of an ID-based ring signature, and therefore presents the contribution related to Research Objective O2 (see Chapter 1.3.2). This chapter contains material from our paper "Post-Quantum ID-based Ring Signatures from symmetric-key primitives" published in the conference *Applied Cryptography and Network Security (ACNS) 2022* [ACN].

6.0.1 ID-based ring signature (IDRS)

ID-based cryptography [Sha84] was introduced in 1984 to erase the need for certificates in public key infrastructures (PKI). ID-based cryptography utilizes a public key which is the identity of the user, for example, an identity can be an email address or a name. In this framework, a trusted third party named the private key generator (PKG) is required. PKG uses the identity of a user and his private key to generate the secret signing key of the corresponding identity. The interest in combining both ID-based cryptography and ring signature is undeniable as proven by the works [ALYW06],[CLHY05],[GGCT06],[HS04],[CYH05] and [ZK02]. As explained by Chow et al. [CYH05], the main advantage of ID-based Ring signature (IDRS) over "traditional" ring signatures in PKI is that IDRS provides better spontaneity. PKI's ring signatures can only form a ring with users that requested certificates for their public keys while in IDRS signers can form a ring using users' identities even if they did

not request their secret signing keys to PKG. Additionally, IDRS may significantly reduce the communication overhead in sending the list of ring public keys to the verifier along with the signature for PKI's ring signatures. The ring IDs may be much shorter and may even be implicitly known by the verifier (e.g. all employees of a certain organization/department).

6.0.2 Post-quantum IDRS

In 2016, the post-quantum standardization process was launched by NIST and generated considerable attention from researchers. This also motivates the design of post-quantum IDRS as this would provide primitives which ensure anonymity without requiring any certificates. There exist different post-quantum candidates to design quantum-safe cryptographic primitives. Lattice-based cryptography is currently the most investigated candidate due to its promise of flexibility. There are currently multiple lattice-based IDRS; the first one by Wang [Wan08] and more recent works by Zhao et al. [ZT18], Wei et al. [WDZ+15] and Cao et al. [CYH21].

To the best of our knowledge, those are the only quantum-safe IDRS. In this chapter, similar to the rest of the thesis, we focus on another promising candidate: symmetric primitives, for example hash functions or block ciphers. As presented in Chapter 1.2, these are old primitives providing the advantage of a well-studied and well-understood security. Another advantage is that the security of a symmetric-based protocol depends only on the integrated primitives and not on any assumed hard problem. This means that, if a symmetric primitive has been broken, it can simply be replaced and the design would still be secure. On the contrary, if the hardness assumption of lattice-based constructions has been broken, then all schemes relying on this assumption are not secure anymore. The recent design of zero-knowledge proof systems obtained from symmetric primitives opens up new directions. Zero-knowledge proof systems as ZKBoo [GMO16], ZKB++ [CDG+17], KKW [KKW18] and Ligero++ [BFH+20] allow a user to prove the knowledge of a secret witness w such that $C(w) = 1$, where C is a public circuit similar to hash functions. For all these aforementioned reasons, our work studies post-quantum IDRS constructed based on symmetric primitives only. A circuit C is composed of multiplications (or "AND") gates represented by $\cdot$ and additions ("OR") gates represented by $+$.

TABLE 6.1: PicRS and XRS comparisons. PQ=post-quantum, N= ring size, ✓=proven, (✓)= assumed, h=XMSS tree height

IDRS	PQ candidate	$\|\sigma\|$ (Asympt.)	Max. N	IDRS.Setup (Asympt.) time	|SID| (KB)	NIZK	PQ	H	(est.)$\|\sigma\|$(MB)		
									$N = 2^6$	$N = 2^{12}$	$N = 2^{20}$
PicRS	Symmetric	$\mathcal{O}(\log N)$	unli.	$\mathcal{O}(1)$	167	KKW	✓	LowMC	169.964	170.153	170.406
								SHA3	3619	3622	3626
						Ligero++	(✓)	SHA3	2.046	2.046	2.047
								MiMC	1.902	1.902	1.903
								Poseidon	1.898	1.899	1.900
XRS	Symmetric	$\mathcal{O}(\log N)$	2^{20}	$\mathcal{O}(2^h)$	4.899	KKW	✓	LowMC	12.487	12.680	12.930
								SHA3	332.300	335.266	339.211
						Ligero++	(✓)	SHA3	1.490	1.491	1.493
								MiMC	0.973	0.976	0.979
								Poseidon	0.885	0.889	0.893
[ZT18]	Lattice	$\mathcal{O}(N)$	unli.	-	615000	-	✓	-	5	335	32243

6.0.3 Detailed contributions

The contributions of this chapter can be presented in the following three parts:

- **Generic Post-quantum ID-based ring signatures (Section 6.2)**: We designed a circuit C (see Section 6.2.1) to allow signers to execute a zero-knowledge proof that they own a witness w such that $C(w) = 1$. C is divided into two sub-circuits: C_1 and C_2. C_1 proves the membership of the signer to the ring through an accumulator (see Definition 2.7.1) and C_2 proves the knowledge of a signing secret key generated by the private key generator PKG. In our generic IDRS construction, the signing private key for the identity ID is a digital signature of ID generated by PKG, therefore C_2 proves the knowledge of a valid digital signature. Both C_1 and C_2 are linked through an "AND" logical gate ($C = C_1 \cdot C_2$). The generic circuit is illustrated in Figure 6.4.

- **Applicable post-quantum ID-based ring signatures named PicRS and XRS from the generic construction (Section 6.3)**: We implemented sub-circuit C_1 with a Merkle Accumulator (Section 6.3.1) to prove the membership to the ring. sub-circuit C_2 can be initiated in two different ways:

  1. PKG uses Picnic digital signature [CDG+17], which means that a signer needs to prove the knowledge of a valid Picnic signature. We designed circuit $\mathsf{Picnic}.C$ presented in Section 6.3.2, Algorithm 11 and Figure 6.6 to prove this statement.
  2. PKG uses the stateful digital signature XMSS [HBGM15], which means that a signer needs to prove the knowledge of a valid XMSS signature. We designed circuit $\mathsf{XMSS}.C$ presented in Section 6.3.2, Algorithm 12 and Figure 6.7 to prove this statement.

  $\mathsf{Picnic}.C$ allows to design the first IDRS PicRS, where a signer uses circuit $C = \mathsf{PicRS}.C = \mathsf{Merkle}.C(= C_1) \cdot \mathsf{Picnic}.C(= C_2)$ to generate a signature. The second implementation named XRS ensues from circuit $\mathsf{XMSS}.C$. In XRS, a signer uses circuit $C = \mathsf{XRS}.C = \mathsf{Merkle}.C(= C_1) \cdot {}_{\mathsf{XMSS}}.C(= C_2)$ to generate a signature. While PicRS is stateless for the signer and PKG, XRS is still stateless for the signer but requires PKG to keep an updated state to the stateful nature of XMSS. The PicRS's PKG can generate an unlimited number of signing secret keys and therefore handle an unlimited number of users, while there is a cap for the maximum number of users in XRS (e.g. 2^{20} users).

- **Applicable constructions analysis and optimization (Section 6.4)**: We evaluate our constructions with two different zero-knowledge proof systems: KKW [KKW18] and Ligero++[BFH+20]. Each of them is tested with the standard hash function SHA3 and, additionally with the following non-standard hash functions to optimize the signature size: LowMC [ARS+15], MiMC [AGR+16] and Poseidon [GKR+21]. We optimize the complexity of circuit $\mathsf{Picnic}.C$ and $\mathsf{XMSS}.C$ by testing different parameters for Picnic and XMSS to achieve the best possible signature size. In theory, both schemes have a signature size that grows logarithmically ($\mathcal{O}(\log N)$) with the size of the ring represented by N. This is an improvement when compared with lattice-based IDRSs [Wan08], [ZT18], [WDZ+15] or [CYH21], whose signature size grows linearly ($\mathcal{O}(N)$) with the ring size, making them unsuitable for large rings. In practice, PicRS and XRS signature sizes are nearly constant because proving the knowledge of a valid Picnic (PicRS) or XMSS (XRS) signature is the bottleneck of both signature

sizes. PicRS achieves a size of 1.900MB while XRS requires only 889KB for a ring of 4096 members. Table 6.1 demonstrates that these sizes are competitive when compared to the current state-of-the-art of lattice-based IDRS introduced by Zhao et al. [ZT18], which is the only work proposing concrete parameters and allowing us to estimates their signature sizes.

6.0.4 Overview of techniques

At the heart of our generic IDRS, we utilize a non-interactive zero-knowledge proof system (NIZK) based on symmetric primitives, which allows us to prove the knowledge of a witness w such that, for a public circuit C, we have $C(w) = 1$. Traditional digital signatures, for example Picnic [CDG+17], run the NIZK on a circuit related to the underlying one-way function as in the original zero-knowledge proof based signature schemes similar to Picnic [CDG+17] New challenges arise in IDRS as distinct from a traditional digital signature, namely, the generated proof needs to include a part of the IDRS signature involves the "verification" circuit for the signing secret key generation (i.e. verification algorithm of a standard signature by the key generation authority), rather than.

In our generic IDRS construction, a signer with an identity ID owns a witness that is the signing secret key SID. SID is a digital signature (see Definition 2.4.1), generated by PKG, using its ID as a message. A signer will use the NIZK proving procedure on circuit C to generate a signature of a message m. C is designed to prove that he knows a valid SID, in other words, a valid digital signature generated by PKG for an ID belonging to the ring L. Circuit C takes as inputs (i.e. the witness) the signer's identity ID, the corresponding signing secret key SID and the list of identities L (i.e. the ring). More formally, the signer will prove the knowledge of $(\mathsf{ID}, \mathsf{SID}, \mathsf{L})$ such that $C(\mathsf{ID}, \mathsf{SID}, \mathsf{L}) = 1$. C, summarized in Figure 6.4, is composed of two main sub-circuits named C_1 and C_2, which are used to prove the membership in the ring and to prove the knowledge of a valid digital signature.

We construct applicable constructions by defining and optimizing circuit C and both its sub-circuits C_1 and C_2. C_1 is implemented as $\mathsf{Merkle}.C$ which is constructed on top of a Merkle accumulator [DHS15, BEF19, KKW18] and a multiplexer (see Section 6.3.1) to hide the position of the identity into the accumulator. C_2 can be implemented with the verification procedure of the Picnic digital signature or the verification algorithms of the stateful XMSS digital signature. XRS follows the same idea but, instead of having a valid picnic signature as SID, each user owns an XMSS signature. The proof of knowledge of a valid signature will be done through circuit $\mathsf{XMSS}.C(= C_2)$. As the stateful nature of XMSS, $\mathsf{XMSS}.C$ requires the use of multiplexers (see Equation 6.7) to hide the state and, so, provide anonymity.

6.0.5 Outline of chapter 6

This chapter starts with Section 6.1 which presents the Picnic signature [CDG+17, KZ20] and the XMSS signature [HBG+18] which are used in this chapter. This section also includes the formal definition of IDRS. The generic construction is presented in Section 6.2 and the presentation of both possible instances PicRS and XRS are introduced in Section 6.3. This paper concludes by a full evaluation of both applicable constructions, PicRS and XRS, in Section 6.4. This chapter concludes with a final discussion about Research Objective O2.

6.1 Additional definitions

The content presented in this chapter implements cryptographic hash fucntions (see Definition 2.2.1 on page 17), accumulators (see Definition 2.7.1 on page 20) and non-interactive zero-knowledge proof system (NIZK) (see Definition 2.8.1 on page 21). This section provides the definitions the Picnic digital signature scheme [CDG+17, KZ20] and the XMSS stateful digital signature [HBGM15, HBG+18] which are both at the heart of PicRS and XRS.

6.1.1 Picnic signature

The digital signature Picnic, which follows Definition 2.4.1, is linked the zero-knowledge system KKW [KKW18]. KKW is a Σ-protocol [FS86], which can be made non-interactive thanks to the Fiat-Shamir transform [FS86]. The Picnic signature executes the KKW zero-knowledge proof system including the message to be signed into the Fiat-Shamir transform[1]. Therefore, we will present the KKW algorithms through the Picnic algorithms.

KKW: the zero-knowledge proof system

Definition 6.1.1 (KKW)**.** *The zero-knowledge proof system* KKW *can be defined by the following algorithms:*

$\pi \leftarrow \mathsf{KKW.Prove}(x, w)$*: This takes as input the public x and the witness w and outputs a proof π that $(x, w) \in R$. It proves the knowledge of w such that $C(w) = 1$ where C is a binary circuit. (see Remark 1)*

$0/1 \leftarrow \mathsf{KKW.Verify}(x, \pi)$*: This takes as input the public statement y and the proof π. It outputs 1 if π proves the knowledge of w such that $C(w) = 1$.*

KKW applies the MPC-in-the-head paradigm described in [IKOS09], and [CDG+17]. In MPC-in-the-head, the prover simulates a MPC computation between n parties of a binary circuit C composed of "AND" and "XOR" logical gates. The n parties, shares the "XORED" bit b, which is denoted by $[b] = \bigoplus_{i=1}^{n} b_i$. The proof will reveal the views of $n-1$ parties, named opened parties, and the messages broadcasted by the last party, unopened party. The verifier will execute the MPC-in-the-head protocol based on those views and the messages of the unopened party. It is important to know that it requires $(n-1)-$privacy which means knowing the view of $n-1$ parties does not reveal anything about the last party. In KKW, each wire a has a value z_a and a party P_i owns a mask λ_a for the wire a which allow to compute the masked value $\hat{z}_a = z_a \oplus \lambda_a$.

A party P_i owns a seed $\mathsf{seed}^{(i)} \in \{0,1\}^{2\lambda}$, which is used to compute his mask $\lambda_{a,i}$ for each single wire in C. The party P_n owns an additional value aux_n, which is a correction value of AND bits, where AND is the number of "AND" gates in C. In order to achieve acceptable soundness, the proof contains multiple parallel executions of the MPC-in-the-head simulation. KKW is defined following parameters:

- n: the number of MPC parties,
- M: the number of parallel executions of the MPC-in-the-head protocol,
- AND: the number of "AND" gates in C,
- τ: number of online executions revealed in the proof ($\tau \leq M$), and

- $M - \tau$: number of preprocessing computations revealed.

In the prepropressing, each party generates AND triples $\{([\lambda_a], [\lambda_b], [\lambda_c])\}$ such that

$$[\lambda_c] = [\lambda_a] \cdot [\lambda_b]. \tag{6.1}$$

All λ_a, λ_b and λ_c can be generated thanks to party-owned seed $\mathsf{seed}^{(i)}$. Moreover, the correction value aux_n is part of the $state_n$ of the P_n in order to insure the validity of (6.1). This part is done M times. Then the online part starts. The parties then evaluate the circuit C in a gate-by-gate fashion in chronological. The only type of gates that requires communication is the "AND" gates and the following procedure is done:

- Each party locally computes and broadcasts $[s] = \hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma]$, where $\lambda_\gamma = \lambda_a \oplus \lambda_b$. Then computes $\hat{z}_\gamma = \hat{z}_a \cdot \hat{z}_b \oplus s$.

At the end of the circuit C, each party broadcast its share and then the outputs is reconstructed. The online part is done only over τ randomly selected prepossessed executions.

Picnic signature

The rest of this subsection presents the KKW algorithms through the post-quantum digital signature Picnic [CDG+17, KZ20, Pic]. The Picnic signature size can be computed thanks to the following formula [KKW18, dSGDMOS19]:

$$4\lambda + \tau \log(M/\tau) 6\lambda + \tau(2\lambda \log(n) + 2\mathsf{AND} + |w| + 4\lambda). \tag{6.2}$$

Definition 6.1.2 (Picnic)**.** *The Picnic signature follows Definition 2.4.1 and therefore is defined by the following algorithms:*

$(\mathsf{Picnic.pk}, \mathsf{Picnic.sk}) \leftarrow \mathsf{Picnic.KeyGen}(1^\lambda)$*: The key generation algorithm initiates the binary circuit C. It samples w, computes $y \leftarrow C(w)$, and lets* $\mathsf{Picnic.sk} = w$ *and* $\mathsf{Picnic.pk} = y$.

$\mathsf{Picnic}.\sigma \leftarrow \mathsf{Picnic.Sign}(m, \mathsf{Picnic.sk})$*: This is simply an execution of* KKW.Prove *algorithm and uses the message m to generate the challenge with the help of the Fiat-Shamir transform. The signing algorithm has the following steps:*

1. *for $j \in [M]$*
   (a) *Sets $\mathsf{seed}_j^* \xleftarrow{\$} \{0,1\}^{2\lambda}$, to generate $\mathsf{seed}_j^{(1)}, \ldots, \mathsf{seed}_j^{(n)}$ (one for each party) and $r_{j,1}, \ldots, r_{j,n}$, set $\mathsf{seed}_j^{(i)} = state_{j,i}$ for $1 \leq i \leq n-1$ and $state_{j,n} = \mathsf{seed}_j^{(n)} || aux_j$ where aux_j is the correction value for the j-th iteration and is computed as follow.*
   - *For $y \in [\mathsf{AND}]$*
     i. *The party P_i uses $\mathsf{seed}_j^{(i)}$ to sample $\lambda_{a_{j,y}}^{(i)}, \lambda_{b_{j,y}}^{(i)}$ and $\lambda_{c_{j,y}}^{(i)}$*
     ii. *Compute $[\lambda_{a_{j,y}}]$ and $[\lambda_{b_{j,y}}]$,*
     iii. *Compute $\delta_{j,y} = [\lambda_{a_{j,y}}] \cdot [\lambda_{b_{j,y}}] \oplus [\lambda_{c_{j,y}}]$. Let $aux_j^{(n)} = (\delta_{j,y})_{y \in [\mathsf{AND}]}$.*
   (b) *For $i \in [n]$: $com_{j,i} = \mathsf{H}(state_{j,i}, r_{j,i})$*
   (c) *$h_j = \mathsf{H}(com_{j,1}, \ldots, com_{j,n})$.*

(d) *This procedure starts by computing the masked* $\hat{z}_{j,a}$ *from the witness* $w = \mathsf{Picnic.sk}$ *and the* λ_a *computed in step (1a) inputs for each wire* a*. Then compute the binary Circuit* C *by proceeding through the gates in order. For each party* P_i *the messages* $msgs_{j,i}$ *are generated for the iteration* j*,* $msgs_{j,p_j}$ *are actually the communication for each "AND" gate.*

(e) $h'_j = \mathsf{H}(\hat{z}_{j,a}, msgs_{j,1}, \ldots, msgs_{j,n})$.

2. $(\mathcal{C}, \mathcal{P}) = G(m, h_1, h'_1, \ldots, h_M, h'_M)$*, with* $\mathcal{C} \subseteq [M]$ *composed* τ *elements and* $\mathcal{P}$ *representing the set of unopened parties, one per parallel execution. This last part is the only difference between KKW and Picnic. In KKW, the challenge is generated thanks the preprocessing step, while in Picnic, the message* m *to be signed is included in the Fiat-Shamir transform.* G *is a hash function with an output size of* $\tau \cdot \log M + \tau \log N$.

3. ***Return*** $\mathsf{Picnic}.\sigma = (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}^*_j, h'_j\}_{j \notin \mathcal{C}}, \{\{state_{j,i}, r_{j,i}\}_{i \neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j \in \mathcal{C}})$.

$0/1 \leftarrow \mathsf{Picnic.Verify}(m, \mathsf{Picnic}.\sigma, \mathsf{Picnic.pk})$*: This algorithm executes the* KKW *verification procedure. It return* 1 *if the result of* KKW.Verify *is* 1 *and* 0 *otherwise, where* $\mathsf{Picnic}.\sigma = (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}^*_j, h'_j\}_{j \notin \mathcal{C}}, \{\{state_{j,i}, r_{j,i}\}_{i \neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j \in \mathcal{C}})$ *this algorithms does:*

1. *For* $j \in \mathcal{C}$ *and* $i \in [n]$ *with* $i \neq p_j$*, set* $com_{j,i} := \mathsf{H}(state_{j,i}, r_{j,i})$ *and then compute the value* $h_j := \mathsf{H}(com_{j,1}, \ldots, com_{j,n})$.
2. *For* $j \notin \mathcal{C}$ *used* seed^*_j *to compute* h_j *as the signer would (step 1 (a) to (c) of* Picnic.Sign *algorithm)*
3. *For each* $j \in \mathcal{C}$ *run an execution of* C *among the parties* $\{P_i\}_{i \neq p_j}$ *using* $\{state_{i,j}\}_{i \neq p_j}$ *,* $\{\hat{z}_{j,a}\}$*, and* $msgs_{j,p_j}$ *; this yields* $msgs_{j,p_j}$ *and an output bit* b*. Check that* $b = 1$*. Then compute* $h'_j = \mathsf{H}(\{\hat{z}_{j,a}\}, msgs_{j,1}, \ldots, msgs_{j,n})$.
4. *Return* $(\mathcal{C}, \mathcal{P} == G(m, h_1, h'_1, \ldots, h_M, h'_M))$.

The EU-CMA (see Definition 2.4.2) security of Picnic has been proven. This means that for an adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}^{\mathsf{Picnic}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{EU\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{6.3}$$

6.1.2 The stateful digital signature: XMSS

The XMSS [HBGM15] is a digital signature relying on Merkle tree [Mer89] and properties of a cryptographic hash function H. XMSS achieves EU-CMA (see Definition 2.4.2) as presented in [HBGM15]. A signer can produce at most 2^h signatures. As XMSS is stateful as the signer needs to update his key after each signature. At the heart of XMSS, there is a one-time signature scheme named WOTS^+, in this chapter we only consider WOTS^+ in XMSS and therefore, WOTS^+ signature never needs to be verified. We already defined the XMSS signature in Chapter 5, the algorithms are executing the same procedure than but we adapted their definitions in order to facilitate the comprehension of the scheme presented in this chapter.

Definition 6.1.3 (WOTS^+)**.** WOTS^+ *is a one-time signature scheme with the parameters* Wint *and* $\mathsf{len} = \mathsf{len}_1 + \mathsf{len}_2$ *(see Table 6.2).* WOTS^+ *defined by the following algorithms:*

$(\mathsf{WOTS}^+.\mathsf{pk}, \mathsf{WOTS}^+.\mathsf{sk}) \leftarrow \mathsf{WOTS}^+.\mathsf{KeyGen}(1^\lambda)$: *This algorithm computes for* $i \in [\mathsf{len}]$: $\mathsf{WOTS}^+.\mathsf{sk}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$; $\mathsf{WOTS}^+.\mathsf{pk}_i \leftarrow H^{\mathsf{Wint}}(\mathsf{WOTS}^+.\mathsf{sk}_i)$, *then* $\mathsf{WOTS}^+.\mathsf{pk} = \mathsf{Merkle}(\{\mathsf{WOTS}^+.\mathsf{pk}_i\}_{i=1}^{\mathsf{len}})$ *(this reduces* $\mathsf{WOTS}^+.\mathsf{pk}$ *in a* 2λ *bits elements, see Table 6.2).*

$\mathsf{WOTS}^+.\sigma \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(m, \mathsf{WOTS}^+.\mathsf{sk})$: *This generates a signature* $\mathsf{WOTS}^+.\sigma = (\mathsf{WOTS}^+.\sigma_1, \ldots, \mathsf{WOTS}^+.\sigma_{\mathsf{len}}, r) = (H^{md_1}(\mathsf{WOTS}^+.\mathsf{sk}_1), \ldots, H^{md_{\mathsf{len}}}(\mathsf{WOTS}^+.\mathsf{sk}_{\mathsf{len}}), r)$. *Where* md *is compute with the following step:* $r \xleftarrow{\$} \{0,1\}^{2\lambda}$, $md = (md_1, \ldots, md_{\mathsf{len}_1}) \leftarrow \mathsf{H}(\mathsf{ID}, r)$, $c = (md_1, \ldots, md_{\mathsf{len}_2}) \leftarrow \sum_{i=1}^{\mathsf{len}_1}(\mathsf{Wint} - 1 - md_i)$ *and* $md = md||c$

$\mathsf{WOTS}^+.\mathsf{pk}' \leftarrow \mathsf{WOTS}^+.\mathsf{Verify}(m, \mathsf{WOTS}^+.\sigma, \mathsf{WOTS}^+.\mathsf{pk})$: *In the context of the XMSS,* WOTS^+ *signature is not verified. This procedure outputs the corresponding key* $\mathsf{WOTS}^+.\mathsf{pk}'$ *based on the signature* $\mathsf{WOTS}^+.\sigma$ *with the following procedure.* $(\mathsf{WOTS}^+.\mathsf{pk}'_1, \ldots, \mathsf{WOTS}^+.\mathsf{pk}'_{len})$ $= (H^{\mathsf{Wint}-md_1}(\mathsf{WOTS}^+.\sigma_1), \ldots, H^{\mathsf{Wint}-md_{\mathsf{len}}}(\mathsf{WOTS}^+.\sigma_{\mathsf{len}}))$. *Where Where md is compute with the following step:* $md = (md_1, \ldots, md_{\mathsf{len}_1}) \leftarrow \mathsf{H}(\mathsf{ID}, r)$, $c = (md_1, \ldots, md_{\mathsf{len}_2}) \leftarrow \sum_{i=1}^{\mathsf{len}_1}(\mathsf{Wint} - 1 - md_i)$ *and* $md = md||c$ *Then it returns* $\mathsf{WOTS}^+.\mathsf{pk}' = \mathsf{Merkle}(\{\mathsf{WOTS}^+.\mathsf{pk}'_i\}_{i=1}^{\mathsf{len}})$

In XMSS, each leaf is a WOTS^+ key pair which will be used by the signer in chronological order. An XMSS signature is composed of a WOTS^+ signature and an authentication path which are the internal nodes of a binary tree. The authentication path allows one to compute the tree root from the WOTS^+ signature. In contrast to Chapter 5 and for simplification reasons, we omit the use of bitmask in the XMSS tree construction in this chapter.

Definition 6.1.4 (XMSS)**.** XMSS *[BDH11] is a stateful digital signature defined by the following three PPT algorithms:*

$(\mathsf{XMSS.pk}, \mathsf{XMSS.sk}) \leftarrow \mathsf{XMSS.KeyGen}(1^\lambda)$: *This starts by generating* 2^h WOTS^+ *key pairs. Then a binary tree is built using the* $\mathsf{WOTS}^+.\mathsf{pk}$ *as leaves and outputs the root* XMSS.pk*. An example is presented in see Figure 6.1.*

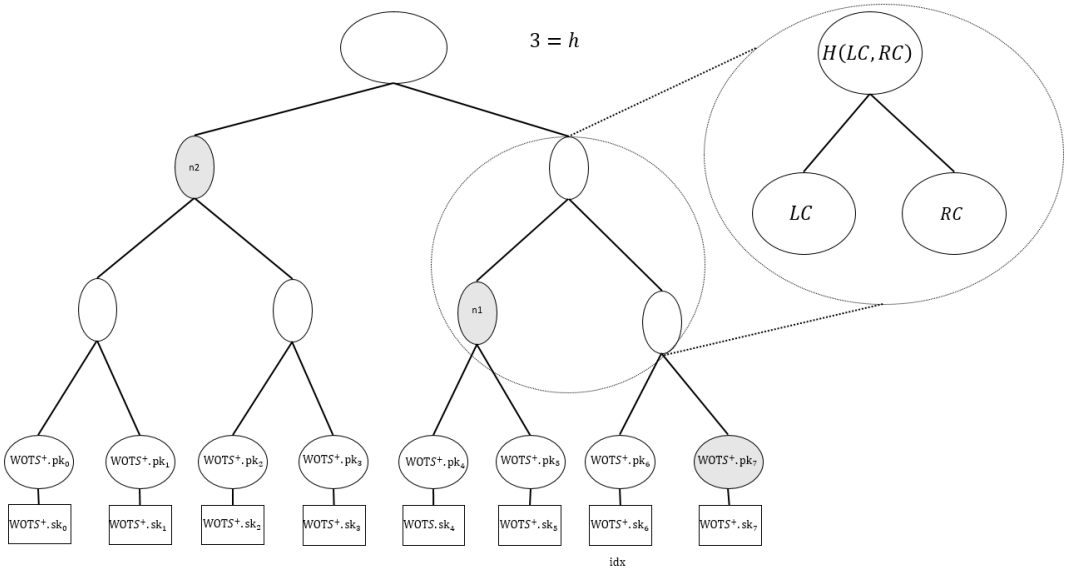
$\mathsf{XMSS}.\sigma \leftarrow \mathsf{XMSS.Sign}(m, \mathsf{XMSS.sk})$: *This takes a message* m *to be signed, the secret key* XMSS.sk *and returns* $\mathsf{XMSS}.\sigma$ *thanks to following procedure:*

1. $\mathsf{WOTS}^+.\sigma \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(m, \mathsf{WOTS}^+.\mathsf{sk}_{\mathsf{idx}})$,
2. $\mathsf{XMSS}.\sigma = (\mathsf{WOTS}^+.\sigma, \mathsf{idx}, \mathsf{auth})$, *where the index* idx *indicates the position of the* WOTS^+ *keys in the tree and* auth *is the authentication path from the* WOTS^+ *to the root (the grey nodes in Figure 6.1), and*
3. $\mathsf{idx} = \mathsf{idx} + 1$. *Update the index* idx *such that the next* WOTS^+ *key pair will be used for the next signature.*

$1/0 \leftarrow \mathsf{XMSS.Verify}(m, \mathsf{XMSS}.\sigma, \mathsf{XMSS.pk})$: *This takes as input a message* m*, a signature* $\mathsf{XMSS}.\sigma$ *and the public key* XMSS.pk*. It outputs* 1 *if* $\mathsf{XMSS}.\sigma$ *is valid and* 0 *otherwise. This procedure executes the following steps:*

1. $\mathsf{WOTS}^+.\mathsf{pk}' = \mathsf{WOTS}^+.\mathsf{Verify}(m, \mathsf{WOTS}^+.\sigma)$ *(see Definition 6.1.3),*
2. *Compute* $\mathsf{XMSS.pk}'$ *from* $\mathsf{WOTS}^+.\mathsf{pk}'$ *with the help of the authentication path* auth *(the grey nodes in Figure 6.1). This corresponds to* h *evaluations of the hash function* H*, and*
3. *Return* $\mathsf{XMSS.pk} == \mathsf{XMSS.pk}'$.

FIGURE 6.1: XMSS tree



The EU-CMA (see Definition 2.4.2) security of XMSS has been proven [BDH11, HBGM15]. This means that for an PPT adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{XMSS}}(\lambda) = \Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{DS}}^{EU\text{-}CMA}(\lambda) = 1] < \mathsf{negl}(\lambda). \tag{6.4}$$

6.1.3 Definition of ID-based ring signature (IDRS)

We now formally define an ID-based ring signature. In IDRS, there is a private key generator (PKG), which is a trusted identity generating the signing secret keys of users. Only users who have received a signing secret key SID from PKG can generate a valid and anonymous signature.

Definition 6.1.5 (ID-based ring signature)**.** *An ID-based ring signature is defined by the tuple of PPT algorithms:* $\mathsf{IDRS} = (\mathsf{IDRS.Setup}, \mathsf{IDRS.KeyGen}, \mathsf{IDRS.Sign}, \mathsf{IDRS.Verify})$

- $(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)$*: This algorithm takes as input the security parameters* λ*, it produces the master public key* mpk*, the master secret key* msk *and the public parameters* param*. This procedure is executed by the private key generator (PKG).*
- $\mathsf{SID} \leftarrow \mathsf{IDRS.KeyGen}(\mathsf{ID}, \mathsf{msk})$*: This algorithm takes as input an identity* $\mathsf{ID} \in \{0,1\}^*$ *and the master secret key* msk*, it outputs the signer's secret signing key* SID*. This procedure is executed by the private key generator PKG and the result is transmitted to the user with the identity* ID*.*
- $\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$*: This algorithm takes as input the message* m*, a list* L *of* N *identities, the identity of the signer* ID*, the signing secret key* SID *of the member* ID*, where* $\mathsf{ID} \in \mathsf{L}$*, the master public key* mpk *and the public parameters* param*. It outputs a ring signature* σ*.*
- $0/1 \leftarrow \mathsf{IDRS.Verify}(m, \mathsf{L}, \sigma, \mathsf{mpk}, \mathsf{param})$*: This algorithm takes as input a ring signature* σ*, a message* m*, the ring list* L*, the master public key* mpk *and the*

$\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{IDRS}}(\lambda)$

Setup:
This oracle is executed by the challenger $\mathcal{CH}$ who performs the setup algorithm $(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)$. PKG's public key mpk is then sent to the adversary $\mathcal{A}$. The sets $\mathcal{Q}, \mathcal{S}$ are initialized: $\mathcal{Q}, \mathcal{S} \leftarrow \emptyset$.
Query:
$\mathsf{SID} \leftarrow \mathcal{CO}(\mathsf{ID})$: This oracle corrupts a user. The adversary $\mathcal{A}$ sends the user's identity ID to the challenger $\mathcal{CH}$ which computes the corresponding secret key $\mathsf{SID} \leftarrow \mathsf{IDRS.KeyGen}(\mathsf{ID}, \mathsf{msk})$ and sends SID to $\mathcal{A}$. ID is added to the set $\mathcal{Q}$.
$\sigma \leftarrow \mathcal{SO}(m, \mathsf{L}, \mathsf{ID}, \mathsf{mpk}, \mathsf{param})$: This signing oracle starts with $\mathcal{A}$ sending to the challenger $\mathcal{CH}$ a list of user's identities L, message m and the identity of the signer $\mathsf{ID} \in \mathsf{L}$ and $\mathcal{CH}$ returns a valid signature σ for message m generated by the user ID ($\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$). The tuple (m, L) is added to the set $\mathcal{S}$.
Forgery:
$\mathcal{A}$ forges $(m^*, \mathsf{L}^*, \sigma^*)$.
if $1 \leftarrow \mathsf{IDRS.Verify}(m^*, \mathsf{L}^*, \sigma^*, \mathsf{mpk}, \mathsf{param}) \wedge ((m^*, \mathsf{L}^*) \notin \mathcal{S}) \wedge (\mathsf{ID} \notin \mathcal{Q}$ for all $\mathsf{ID} \in \mathsf{L}^*)$
then return 1
return 0

FIGURE 6.2: IDRS Unforgeability Game.

public parameters param. *It outputs* 1 *if* σ *is valid and generated by one* $\mathsf{ID} \in \mathsf{L}$, 0 *otherwise.*

Security properties

A secure ID-based ring signature achieves correctness, unforgeability and anonymity. All security experiments presented in Figure 6.2 and 6.3 are performed between an adversary $\mathcal{A}$ and a challenger $\mathcal{CH}$.

Definition 6.1.6 (Correctness)**.** *An* IDRS *achieves correctness if and only if* $\Pr[\mathsf{IDRS.Verify}(m, \mathsf{L}, \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param}), \mathsf{mpk}, \mathsf{param})] = 1$ *for any* $\mathsf{SID} \leftarrow \mathsf{IDRS.KeyGen}(\mathsf{ID}, \mathsf{msk})$ *and* $(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)$.

Definition 6.1.7 (Unforgeability)**.** *An* IDRS *achieves unforgeability if it is infeasible for an adversary* $\mathcal{A}$ *to generate a valid signature* σ *from the identities of the ring. An* IDRS *achieves unforgeability if and only if for a security parameter* λ *the advantage* $\mathsf{Adv}^{Forge}_{\mathcal{A}}$ *of an adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}^{Forge}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{IDRS}}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{6.5}$$

where the unforgeability experiment $\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{IDRS}}(\lambda)$ *is presented in Figure 6.2.*

Definition 6.1.8 (Anonymity)**.** *An* IDRS *for a ring* L*, a message* m *and a signature* $\sigma = \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$*, achieves anonymity if and only if for a security parameter* λ *the advantage* $\mathsf{Adv}^{Anon}_{\mathcal{A}}$ *of an adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}^{Anon}_{\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{Anon}_{\mathcal{A},\mathsf{IDRS}}(\lambda) = 1] - 1/N| < \mathsf{negl}(\lambda), \tag{6.6}$$

where the anonymity experiment $\mathsf{Exp}^{Anon}_{\mathcal{A},\mathsf{IDRS}}(\lambda)$ *is defined in Figure 6.3 and* N *is number of ring members.*

$\mathsf{Exp}^{Anon}_{\mathcal{A},\mathsf{IDRS}}(\lambda)$

Setup:
This game is executed by the challenger $\mathcal{CH}$ who performs the setup algorithm $(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)$. PKG's key pair $(\mathsf{mpk}, \mathsf{msk})$ is sent to the adversary $\mathcal{A}$.
Challenge:
$\sigma \leftarrow \mathcal{CHO}(m, \mathsf{L})$: This interactive oracle is initiated by the adversary $\mathcal{A}$, who sends a message m to be signed and a list of identities L to the challenger $\mathcal{CH}$. $\mathcal{CH}$ randomly picks $\mathsf{ID} \in \mathsf{L}$ and generates a valid signature σ for the selected ID ($\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$). $\mathcal{CH}$ sends σ to $\mathcal{A}$.
Output:
$\mathcal{A}$ chooses ID^* based on the received σ.
if $\mathsf{ID}^* = \mathsf{ID}$
then return 1
return 0

FIGURE 6.3: IDRS Anonymity Game.

6.2 Generic construction for ID-based ring signature from symmetric primitives

This section introduces our proposed generic construction of IDRS based on symmetric primitives. A key part of our proposal is that symmetric-based zero-knowledge proof systems (NIZK) give us the ability to prove the knowledge of an input (i.e. witness) w of a circuit C such that $C(w) = 1$. In an IDRS, the signer needs to demonstrate that (1) he owns a secret key generated by the central authority PKG and that (2) his identity belongs to the rings. This requires a circuit C that proves both statements.

The basic idea behind our generic IDRS is that the PKG possesses a digital signature (see Definition 2.4.1) key pair as master public mpk and secret key msk (i.e. $\mathsf{mpk} = \mathsf{DS.pk}, \mathsf{msk} = \mathsf{DS.sk}$). Then, each user with an identity ID requests a signing key SID to PKG who generates a digital signature $\mathsf{DS}.\sigma$ taking the identity ID as the message or, in other words, PKG computes $\mathsf{SID} \leftarrow \mathsf{DS.Sign}(\mathsf{ID}, \mathsf{msk})$. To sign a message m, a user in possession of a signing secret key generated by PKG proves the knowledge through a NIZK (see Definition 2.8.1) of SID associated with an identify ID belonging to the ring of identities L (i.e $\mathsf{ID} \in \mathsf{L}$). We designed a generic circuit C (see Figure 6.4) which proves the validity of both statements. C is divided into two sub-circuits" C_1 to prove $\mathsf{ID} \in \mathsf{L}$ and C_2 to prove $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$. C_1 and C_2 are associated with "AND" gate to form the overall circuit C.

6.2.1 Generic IDRS algorithms

We now formally define the algorithms for our generic construction of IDRS, which follows Definition 6.1.5.

$(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)$: This algorithm is executed by PKG and performs the following steps:

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$ (see Definition 2.4.1)
- $\mathsf{param} \leftarrow \mathsf{A.Gen}(1^\lambda)$ (See Definition 2.7.1)
- **Return** $(\mathsf{mpk}, \mathsf{msk})$

$\mathsf{SID} \leftarrow \mathsf{IDRS.KeyGen}(\mathsf{ID}, \mathsf{msk})$: This algorithm is executed by PKG on the request of the user with the identity ID. It computes a digital signature $\mathsf{DS}.\sigma$ using the ID as the message. The digital signature $\mathsf{DS}.\sigma$ is transmitted to the user ID with and becomes his signing secret key SID. This procedure executes the following steps:

- $\mathsf{SID} \leftarrow \mathsf{DS.Sign}(\mathsf{ID}, \mathsf{msk})$ (see Definition 2.4.1)
- **Return** SID

$\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$: This procedure takes as inputs the message m, the set of N identities L, in other words the ring, the signing secret key $\mathsf{SID} = \mathsf{DS}.\sigma$ and the master public key and the public parameters param which the initial public key of an empty accumulator. This executes the following steps:

- $(\mathsf{A_L}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{param}, \mathsf{L})$: This "accumulates" the set of identities belonging to the ring L. It returns an accumulator A and its updated public key A.pk.
- $\mathsf{w_{ID}} \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A_L}, \mathsf{L}, \mathsf{ID})$: This returns the witness $\mathsf{w_{ID}}$ for the identity of the signer which will be used to prove that his ID is included in the accumulator A.
- $\pi \leftarrow \mathsf{NIZK.Prove}((m, \mathsf{A.pk}, \mathsf{A_L}, \mathsf{mpk}), (\mathsf{SID}, \mathsf{ID}, \mathsf{w_{ID}}))$ (see Definition 2.8.1) The secret witness is $w = (\mathsf{SID}, \mathsf{ID}, \mathsf{w_{ID}})$, the public statement x is composed of the message m, the accumulator public key A.pk, the accumulator $\mathsf{A_L}$ build on the ring and PKG public key mpk ($x = (m, \mathsf{A.pk}, \mathsf{A_L}, \mathsf{mpk})$). The tuple $(x, w) \in R$ if and only if the following statements stand:
  1. $1 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A_L}, \mathsf{w_{ID}}, \mathsf{ID})$: This is equivalent to prove $\mathsf{ID} \in \mathsf{L}$, so it will be proven through sub-circuit C_1 (See Figure 6.4).
  2. $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$: This will be proven through sub-circuit C_2 (See Figure 6.4).

  Both statements will be separately proven through the sub-circuits C_1 and C_2 which are linked together with a "AND" gate to form the whole circuit C ((See Figure 6.4)) and assure that they are both valid. The message to be signed, m, is embedded by integrating it to the Fiat-Shamir transform [1] [FS86] to generate the challenge.
- $\sigma \leftarrow \pi$
- **Return** σ

$0/1 \leftarrow \mathsf{IDRS.Verify}(m, \mathsf{L}, \sigma, \mathsf{mpk}, \mathsf{param})$: This algorithm takes as inputs the message m, the list of identities L and a ring signature σ. It verifies the validity of σ by executing the following steps:

- $\pi \leftarrow \sigma$

[1] Fiat-Shamir transform converts an interactive protocol into a non-interactive protocol. It generates the challenge c as an outputs of H ($c = \mathsf{H}(r, m)$), where m is the message to be signed in IDRS, instead of receiving it from the verifier.

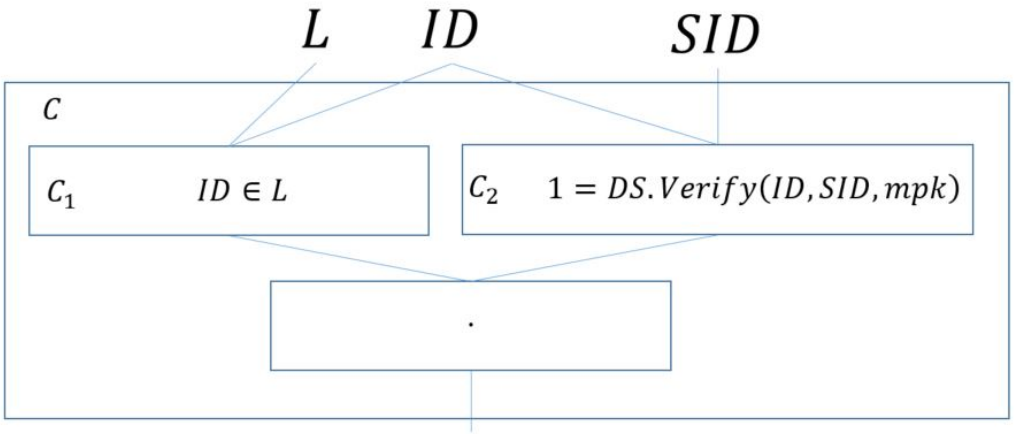


FIGURE 6.4: Generic circuit C

- $(\mathsf{A_L}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{param}, \mathsf{L})$: The verifier construct the accumulator for the set of identities belonging to the ring L. It returns an accumulator $\mathsf{A_L}$ and its updated public key $\mathsf{A.pk}$.
- **Return** $\mathsf{NIZK.Verify}((m, \mathsf{A.pk}, \mathsf{A_L}, \mathsf{mpk}), \pi)$: This returns 1 if the proof π is valid for circuit C.

6.2.2 Security analysis

The security of our generic IDRS depends on the symmetric primitives used, namely a EU-CMA secure digital signature scheme DS (see Definition 2.4.1), a secure accumulator A presented in Definition 2.7.1, a cryptographic hash function H (see Definition 2.2.1) and a secure NIZK NIZK (see Definition 2.8.1).

Theorem 18 (Unforgeability)**.** *Let* IDRS *be the construction provided in Section 6.2.1 with a cryptographic hash function* H*, a EUC-CMA secure digital signature scheme* DS*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then,* IDRS *achieves unforgeability based on Definition 6.1.7.*

Proof. Let V_i be the event that $\mathcal{A}$ wins the unforgeability experiment i and (m^*, L^*, σ^*) is a forgery generated by $\mathcal{A}$. In order to prove the unforgeability of IDRS, we distinguish fours cases in the attempt of signing the message m for the user with the identity ID who belongs to the ring L and please remind that $\sigma^* = \pi^*$:

Event V_1 : $\underline{\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{IDRS}}(\lambda) = 1}$.

Event V_1' :$\underline{V_1}$ happens and $\underline{((\mathsf{ID}^*, \mathsf{SID}^*, \mathsf{w}^*_{\mathsf{ID}}) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}, t, (m^*, \mathsf{A.pk}^*, \mathsf{A}^*_{\mathsf{L}}, \mathsf{mpk}), \pi^*)}$ such that $\underline{((m^*, \mathsf{A.pk}^*, \mathsf{A}^*_{\mathsf{L}}, \mathsf{mpk}), (\mathsf{ID}^*, \mathsf{SID}^*, \mathsf{w}^*_{\mathsf{ID}})) \in R}$: By the simulation extractability property of NIZK (see Definition 2.8.5), we have $\Pr[V_1'] = \Pr[V_1] - \mathsf{negl}(\lambda)$. We divide this event into two disjoint sub-events $v'_{2.1}$ and $v'_{1.2}$:

$v'_{1.1}$: $\underline{\mathsf{ID}^* \in \mathsf{L}^*}$: This event has a negligible probability to happen as this requires breaking the unforgeability of the digital signature scheme DS as $\mathsf{ID}^* \notin \mathcal{Q}$ (see Figure 6.2). $\mathcal{A}$ executes the extractor (see Definition 2.8.5) on a previous signature σ, which gives him a witness SID^*. This event will happen if and only if $\mathcal{A}$ can construct his own SID without interacting with PKG, which

means that he needs to break the digital signature DS used by PKG with the help of the extracted witness. However, as DS achieves EU-CMA security (see Definition 2.4.2), we can conclude that $\Pr[v'_{1.1}] \leq \mathsf{Adv}_{\mathcal{A}}^{EU-CMA} < \mathsf{negl}(\lambda)$.

$v'_{1.2}$: $\underline{\mathsf{ID}^* \notin \mathsf{L}^*}$: This event has a negligible probability to happen due to the collision-freeness property of the accumulator A. In this case, the extractor run on the $\mathcal{A}$'s forgery produces a valid witness $\mathsf{w}^*_{\mathsf{ID}}$ for an extracted identity ID^* not included in the ring L^*, which has been accumulated into $\mathsf{A}_{\mathsf{L}^*}$, i.e. $(\mathsf{A}_{\mathsf{L}^*}, \mathsf{A.pk}^*) = \mathsf{A.Eval}(\mathsf{param}, \mathsf{L}^*)$ but $\mathsf{A.Verify}(\mathsf{A.pk}^*, \mathsf{A}_{\mathsf{L}^*}, \mathsf{w}^*_{\mathsf{ID}}, \mathsf{ID}^*) = 1$. Therefore, if this event happens, we break the collision-freeness property of A (see Definition 2.7.1) Therefore, we can conclude that $\Pr[v'_{1.2}] < \mathsf{negl}(\lambda)$.

Hence $\Pr[V'_1] = \Pr[v'_{1.1}] + \Pr[v'_{1.2}] < \mathsf{negl}(\lambda)$, so we have $\Pr[V_1] < \mathsf{negl}(\lambda)$.

Overall, we can conclude $\Pr[\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{IDRS}}(\lambda) = 1] = \Pr[V_1] < \mathsf{negl}(\lambda)$.

□

Theorem 19 (Anonymity)**.** *Let* IDRS *be the construction provided in Section 6.2.1 with a cryptographic hash function* H*, a EUC-CMA secure digital signature scheme* DS*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then,* IDRS *achieves anonymity based on Definition 6.1.8.*

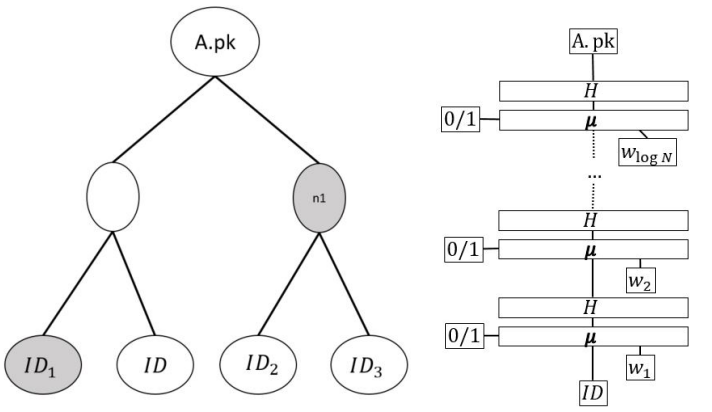
Anonymity. Anonymity is directly stemmed from the zero-knowledge property of NIZK. We use a game-based approach to show that IDRS provides anonymity. Let V_i be the event that the adversary $\mathcal{A}$ wins the GAME_i.
GAME_1: The original anonymity experiment $\mathsf{Exp}^{Anon}_{\mathcal{A},\mathsf{IDRS}}$ (see Figure 6.1.8) is run by $\mathcal{A}$. GAME_2: Same as the previous game but the proof π generated with NIZK on circuit C is replaced with the outputs of its simulator NIZK.Sim (see Definition 2.8.4). This is computationally indistinguishable from the previous game due to zero-knowledge properties of NIZK. Therefore, we can conclude that $|\Pr[V_2] - \Pr[V_1]| = \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) < \mathsf{negl}(\lambda)$. This concludes the Anonymity proof. □

6.3 IDRS: applicable constructions

This section introduces possible applicable constructions of the generic IDRS presented in Section 6.2. This section is divided into two parts which corresponds to the two sub-circuits that composed the circuit presented in Figure 6.4. Section 6.3.1 starts with a presentation of the practical construction of sub-circuit C_1 and discusses its security while Section 6.3.2 is dedicated to presenting possible constructions of sub-circuit C_2 and also discusses their security. The section ends with the summary of two possible implementations of the generic IDRS. We assume that hash functions H output a string of 2λ bits where λ is the post-quantum security level.

6.3.1 Sub-circuit C_1

Sub-circuit C_1 (see Figure 6.4) aims to prove the first statement $1 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_{\mathsf{L}}, \mathsf{w}_{\mathsf{ID}}, \mathsf{ID})$ which is equivalent to prove $\mathsf{ID} \in \mathsf{L}$ or, in other words, that the identity ID of the signer belongs to the ring L. As previously stated, we use an accumulator to prove the membership to the ring. Applying NIZK based on symmetric primitives requires that the verification of a valid witness, A.Verify algorithm, can be expressed as a one-way circuit. This leads us to circuit $\mathsf{Merkle}.C$, derived from the Merkle accumulator.



(A) Example: Merkle accumulator with $N = 4$, $n1 = \mathsf{H}(\mathsf{ID}_2, \mathsf{ID}_3)$ (B) Merkle.C representation

FIGURE 6.5: Merkle Accumulator and circuit Merkle.C

Merkle accumulator and circuit Merkle.C

The Merkle accumulator respects Definition 2.7.1 and "accumulates" the set of identities L as a Merkle tree [Mer89]. Each identity of the ring L is a leaf in a Merkle tree. A Merkle tree is a binary tree in which each internal node is the hash of both its children. The accumulator's public key A.pk is the tree root as illustrated in Figure 6.5a.

The membership proof consists of demonstrating the knowledge of the path from the leaf associated with the signer identity to the root of the tree. This can be represented as a circuit $\mathsf{Merkle}.C(w_{\mathsf{ID}}, \mathsf{ID}) = \mathsf{A.pk}$, where A.pk is the Merkle root and w_{ID} is the authentication path for the identity ID composed of internal nodes of the Merkle accumulator/tree and $\log N$ bits, which indicate the direction of the path. Merkle.C is formally presented in Algorithm 12 on page 126 and is composed of $\log N$ calls of hash function H, where N is the ring size. Additionally, at each level of the Merkle accumulator, Merkle.C goes through a multiplexer μ which is defined as follows:

$$\mu(x, y, b) = \begin{cases} (x, y) & \text{if } b = 0, \\ (y, x) & \text{if } b = 1. \end{cases} \tag{6.7}$$

μ orders the inputs of H depending on the path coming from left or right in the tree. μ hides the path's direction from ID to the root A.pk, hence ensuring anonymity to our IDRS. μ can be written as a circuit $\mu(x, y, b) = (\bar{b}{\cdot}x + b{\cdot}y, \bar{b}{\cdot}y + b{\cdot}x)$. Figure 6.5a depicts an example of a Merkle accumulator and Merkle.C: the corresponding witness for ID is $\mathsf{w}_{\mathsf{ID}} = ((\mathsf{w}_1, 1), (\mathsf{w}_2, 0))$, where the $\mathsf{Merkle}.C(\mathsf{w}_{\mathsf{ID}}, \mathsf{ID}) = \mathsf{H}(\mu(\mathsf{H}(\mu(\mathsf{ID}, \mathsf{w}_1, 1)), \mathsf{w}_2, 0))$. Without the multiplexer, the position of the ID would be identifiable from the path meaning that the anonymity could not be provided. Merkle.C is presented Algorithm 12 on page 126. In conclusion, Merkle.C plays the role of sub-circuit C_1 in our general circuit (see Figure 6.4) to prove that the membership to the ring can be replaced in application by circuit Merkle.C presented in this section. The complexity of Merkle.C grows logarithmically with the number of identities in the ring.

Security and one-wayness Merkle.C

As the security presented in Section 6.2.2 shows, we require a secure accumulator A, in our case, a secure Merkle accumulator which comes from the properties of the hash function H (see Definition 2.2.1). The collision-freeness of the accumulator

comes directly from the collision resistance property of H. This means that it is computationally infeasible to construct the same accumulator from two different sets of ring members. $\mathsf{Merkle}.C$'s one-wayness ensues from the one-wayness of the hash function H (see Definition 2.2.1). This circuit is a Merkle tree with a public root. It should be computationally infeasible to recover any of the leaves from the root and this is achieved because the root is an output of the hash function H, which is a one-way function as presented in Definition 2.2.1.

6.3.2 Sub-circuit C_2

C_2 (see Figure 6.4) aims to prove the validity of signing secret key SID, namely to prove that $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$. Therefore, the verification procedure of the digital signature scheme is to be expressed as a one-way circuit. The current state-of-the-art of digital signatures based on symmetric primitives gives us two possible digital signatures: The stateless Picnic signature (see Definition 6.1.2) and the stateful XMSS signature (see Section 6.1.2). We first present circuit $\mathsf{Picnic}.C$, which aims to prove the knowledge of a valid Picnic signature, and then circuit $\mathsf{XMSS}.C$, which aims to prove the knowledge of a valid XMSS signature.

Circuit $\mathsf{Picnic}.C$

The goal of this circuit is to prove that the signer possesses a valid Picnic signature generated by PKG. Picnic and a detailed description of its parameters are presented in Section 6.1.2. More formally, the signer proves the knowledge of SID such that $1 = \mathsf{Picnic.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$ with the $\mathsf{NIZK.Prove}$ procedure and circuit $\mathsf{Picnic}.C$. Circuit $\mathsf{Picnic}.C$ takes as inputs $\mathsf{SID} = (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}_j^*, h'_j\}_{j\notin\mathcal{C}}, \{\{state_{j,i}, r_{i,j}\}_{i\neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j\in\mathcal{C}})$ and the signer identity ID. To facilitate the explanation of the circuit, we present a high-level picture of $\mathsf{Picnic}.C$ in Figure 6.6 and we divide it into four sub-circuits $\mathsf{Picnic}.c_1, \mathsf{Picnic}.c_2, \mathsf{Picnic}.c_3$ and $\mathsf{Picnic}.c_4$. Each of these sub-circuit execute a specific step of the Picnic verification procedure (see Definition 6.1.2) as presented below:

- Sub-circuit $\mathsf{Picnic}.c_1$ needs to be executed as the first part of the $\mathsf{Picnic.Verify}$ algorithm, which computes the commitment of $n-1$ parties for the τ executions of reveal in the Picnic signature. It starts for every $j \in \mathcal{C}$ and $i \neq p_j$ computes $com_{j,i} := \mathsf{H}(state_{j,i}, r_{j,i})$: This is $(n-1)\tau$ executions of the hash function H. Then it computes the value $\mathsf{H}_j := \mathsf{H}(com_{j,1}, \ldots, com_{j,n})$. At end of the sub-circuit, we have τ elements (recall that $|\mathcal{C}| = \tau$) of 2λ-bits which will be used as inputs in sub-circuit $\mathsf{Picnic}.c_4$. The detailed construction of $\mathsf{Picnic}.c_1$ is presented in Algorithm 12.

- Sub-circuit $\mathsf{Picnic}.c_2$ is executed on the second part of the Picnic verification procedure, namely for $j \notin \mathcal{C}$, using seed_j^* to compute H_j as the signer in the first step of $\mathsf{Picnic.Sign}$ algorithm. This creates commitments of the MPC-in-the-head execution that are not executed in the online phase (see Definition 6.1.2). According to [KZ20], this is used to generate a binary tree similar to GGM [Lyn20] construction with n leaves $\mathsf{seed}_{j,i}$. This sub-circuit can be represented by:

  - For all $j \notin \mathcal{C}$ (goes for $M - \tau$ iterations)
    1. Compute n $\mathsf{seed}_j^{(i)}$ (one per party in MPC) from seed_j^* given in the SID by expending seed_j^* into binary tree of n leaves due to the circuit $\mathsf{Tree}(\mathsf{seed}_j^*, \mathsf{n}) \rightarrow \{\mathsf{seed}_j^{(i)}\}_{i=1}^n$ (see Table 6.2).

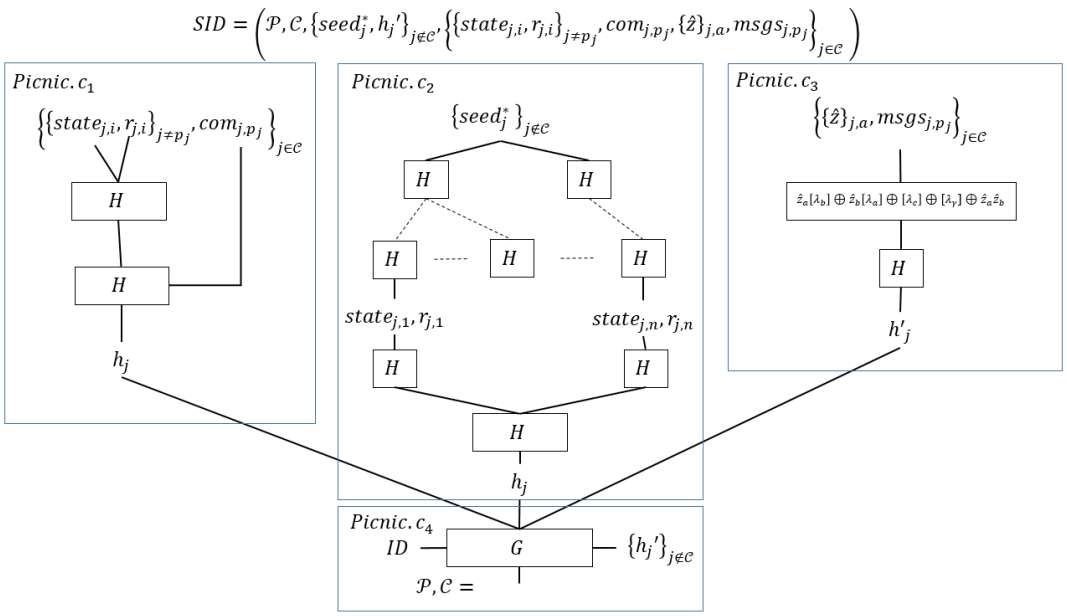


FIGURE 6.6: Circuit Picnic.C

2. Generate $state_{j,i}$ and $r_{j,i}$ from the circuit $\mathsf{ComputeState}(\mathsf{seed}_j^{(i)})$ by computing n (one tuple per party) calls of H (see Table 6.2).
3. For $i \in [n]$: $com_{j,i} := \mathsf{H}(state_{j,i}, r_{j,i})$:
4. Then compute the result of the commitment based $h_j := \mathsf{H}(com_{j,1}, \ldots, com_{j,n})$.

Circuit $\mathsf{Picnic}.c_2$ generates $M - \tau$ outputs of 2λ bits (see detailed construction in Algorithm 11).

- Sub-circuit $\mathsf{Picnic}.c_3$ aims to prove the knowledge of τ valid MPC-in-the-head computation of circuit C_{PKG} (this circuit is used to perform the Picnic signature). For each $j \in \mathcal{C}$ run an execution of circuit C_{PKG} among the parties $\{P_i\}_{i \neq p_j}$ using $\{state_{i,j}\}_{i \neq p_j}$, $\{\hat{z}_{j,a}\}$, and $msgs_{j,p_j}$; this yields $msgs_{j,p_j}$ and an output bit b. Check if $b = 1$. Then compute $\mathsf{H}'_j = \mathsf{H}(\{\hat{z}_{j,a}\}, msgs_{j,1}, \ldots, msgs_{j,n})$. The aim of the sub-circuit is to prove the knowledge of a valid MPC execution. As previously explained, for each "AND" gates of circuit C_{PKG} used by PKG, and for each party P_i, $\mathsf{Picnic}.c_3$ needs to perform the following MPC-in-the-head over the circuit $(\hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma] \oplus \hat{z}_a \cdot \hat{z}_b)$ for each $n-1$ parties for all $|\mathcal{C}| = \tau$ simulations (see Algorithm 11). After the MPC simulation, it checks that the output y is equal to mpk and computes $h'_j = \mathsf{H}(\hat{z}_{j,a}, msgs_{j,1}, \ldots, msgs_{j,n})$.
- Sub-circuit $\mathsf{Picnic}.c_4$ takes as input all elements output by the other circuits $\mathsf{Picnic}.c_1$, $\mathsf{Picnic}.c_2$ and $\mathsf{Picnic}.c_3$ and outputs 1 if the final equality $(\mathcal{C}, \mathcal{P}) \stackrel{?}{=} G(\mathsf{ID}, h_1, h'_1, \ldots, h_M, h'_M)$ holds.

The final circuit $\mathsf{Picnic}.C$ can be represented as $\mathsf{Picnic}.C = \mathsf{Picnic}.c_4(\mathsf{Picnic}.c_3(\mathsf{SID}), \mathsf{Picnic}.c_2(\mathsf{SID}), \mathsf{Picnic}.c_1(\mathsf{SID}), \mathsf{ID})$. The detailed circuit is presented in Algorithm 11 and illustrated in Figure 6.6.

One-wayness and security of Picnic.C

The unforgeability of Picnic have been proven in [CDG+17, KKW18] and therefore provides the desired security according to Theorem 18 and 19. The one-wayness of

Algorithm 11 Picnic.C sub-circuits, for parameters see Table 6.2

SID $= (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}_j^*, h'_j\}_{j\notin\mathcal{C}}, \{\{state_{j,i}\}_{i\neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j\in\mathcal{C}})$

Picnic.c_1

Input: SID
Output: $\{h_j\}_{j\in\mathcal{C}}$

for $j \in \mathcal{C}$ **do**
  $h_j = \mathsf{H}(\mathsf{H}(state_{j,1}, r_{j,1}), \ldots, com_{j,p_j}, \cdots, \mathsf{H}(state_{j,n}, r_{j,n}))$
end for
return $\{h_j\}_{j\in\mathcal{C}}$

Picnic.c_2

Input: SID
Output: $\{h_j\}_{j\notin\mathcal{C}}$

for $j \notin \mathcal{C}$ **do**
  $\{\mathsf{seed}_j^{(i)}\}_{i=1}^n \leftarrow \mathsf{Tree}(\mathsf{seed}_j^*, n)$ (See Table 6.2)
  $(state_{i,j}, r_{i,j}) \leftarrow \mathsf{ComputeState}(\mathsf{seed})$ (See Table 6.2)
  $h_j = \mathsf{H}(\mathsf{H}(state_{j,1}, r_{j,1}), \ldots, \mathsf{H}(state_{j,n}, r_{j,n}))$
end for
return $\{h_j\}_{j\notin\mathcal{C}}$

Picnic.c_3

Input: SID
Output: b

for $j \in \mathcal{C}$ **do**
  Execute C, where C is the circuit used for the Picnic signature.
  for all $\{P_i\}_{i\neq p_j}$ **do**
    for $x \in [\mathsf{AND}]$ **do**
      $(\hat{z}_a{\cdot}[\lambda_b] \oplus \hat{z}_b{\cdot}[\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma] \oplus \hat{z}_a{\cdot}\hat{z}_b)^{(j,i,x)}$ (see Figure 6.6)
    end for
  end for
  $h'_j = \mathsf{H}(\{\hat{z}_j\}, msgs_{j,1}, \ldots, msgs_{j,n})$
end for
return $\{h'_j\}_{j\in\mathcal{C}}$

Picnic.c_4

Input: $\{h_j\}_{j\in\mathcal{C}}, \{h'_j\}_{j\in C}, \{h_j, h'_j\}_{j\notin\mathcal{C}}, \mathsf{ID}$
Output: b

$b = (\mathcal{C}, \mathcal{P}) \stackrel{?}{=} G(\mathsf{ID}, h_1, h'_1, \ldots, h_M, h'_M)$
return b

Picnic.C

Input: SID, ID
Output: 0/1

$b = \mathsf{Picnic}.c_4(\mathsf{Picnic}.c_1(\mathsf{SID}), \mathsf{Picnic}.c_2(\mathsf{SID}), \mathsf{Picnic}.c_3(\mathsf{SID}), \mathsf{ID})$
return b

Picnic.C ensues from the one-wayness of the four sub-circuit presented in Algorithm 11. The one-wayness of Picnic.c_1 depends on the one-wayness of the cryptographic hash function H. Indeed, each step involves only the calls of the hash function H therefore it is computationally infeasible to invert Picnic.c_1. Picnic.c_2 provides also one-wayness for the same reason. Picnic.c_3 one-wayness depends on the one-wayness of the following equation $(\hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma] \oplus \hat{z}_a \cdot \hat{z}_b)$ on which the NIZK is executed. This six inputs equation has a 50% chance to output 0, which makes it one-way. Picnic.c_4's one-wayness follows directly from the one-wayness of hash function G. This important to notice that in Picnic, G computes the challenge (Fiat-Shamir transform) and is modelled as random oracle. Hence, producing NIZKs for calls to a random oracle which is not possible, but in practice G is initialized as an hash function which makes PicRS possible.

XMSS.C circuit

We define circuit XMSS.C that will be used to prove the knowledge of a valid SID such that $1 = \mathsf{XMSS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$ (see Definition 6.1.4). We divided circuit XMSS.C into two sub-circuits XMSS.c_1 and XMSS.c_2 presented in Algorithm 12 on page 126. In order to facilitate the explanation, we also provide a high-level representation of XMSS.C in Figure 6.1.

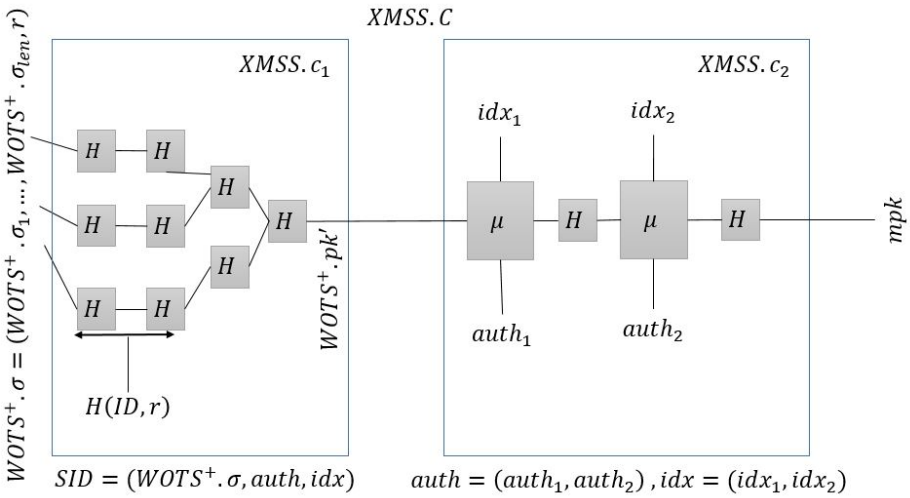


FIGURE 6.7: XMSS.C

- Sub-circuit XMSS.c_1 executes a WOTS$^+$ signatures verification procedure $\mathsf{WOTS^+.pk'} = \mathsf{WOTS^+.Verify}(\mathsf{ID}, \mathsf{WOTS^+}.\sigma)$ (see Definition 6.1.3). It outputs WOTS$^+$.pk$'$.

- Sub-circuit XMSS.c_2 computes XMSS.pk$'$ from WOTS$^+$.pk$'$ (output of circuit XMSS.c_1) with the help of the authentication path auth (the grey nodes in Figure 6.1). This corresponds to h evaluation of the hash function H. This takes the outputs of circuit XMSS.c_1, WOTS$^+$.pk$'$, and using the element auth as chronological outputs to the hash function H and idx (which indicates the order of the inputs of every hash function H). The idx indicates the position of the WOTS$^+$.pk in the XMSS tree. In order to provide anonymity, at each level of the path verification go through a multiplexer gate is defined in 6.7 (the same as the one used for the Merkle accumulator presented in Section 6.3.1). The multiplexer hides the position of the WOTS$^+$.pk in PKG's XMSS tree and

TABLE 6.2: Parameters and additional functions

Wint	Winternitz parameter of WOTS^+
len	WOTS^+ parameters $\mathsf{len} = \mathsf{len}_1 + \mathsf{len}_2$ where $\mathsf{len}_1 = \left\lceil \frac{\lvert m \rvert}{\log(\mathsf{Wint})} \right\rceil$ and $\mathsf{len}_2 = \left\lfloor \frac{\log(\mathsf{len}_1(\mathsf{Wint}-1))}{\log(\mathsf{Wint})} \right\rfloor + 1$
λ	Post-quantum security parameter
$\cdot$	Multiplication/"AND" gate
$+$	Addition/"OR" gate
$\oplus$	xor gate
n	Number of MPC parties for KKW protocol
M	Number of MPC-in-the-head instanciation
idx	Index indicating a leave position in a XMSS/Merkle tree
auth	authentication path in a XMSS tree
$(\mathsf{mpk}, \mathsf{msk})$	master public key & master secret key
ID	user's identity
SID	signing secret key of user ID
h	height of the XMSS tree
τ	number of element in the set $\mathcal{C}$
$r \leftarrow \mathsf{MT}(\mathcal{S})$	This circuit constructs a Merkle root r of binary tree from a set of leaves $\mathcal{S}$. This circuit executes H for $\lvert\mathcal{S}\rvert - 1$ times.
$\{leaf_i\}_{i=1}^{2^{leaves}} \leftarrow \mathsf{Tree}(seed, depth)$	This circuit takes as input a seed *seed* and an integer *depth* indicating the number of the binary tree leaves. This procedure calls $2 \cdot (leaves - 1)$ executions of H.
$(state, r) \leftarrow \mathsf{ComputeState}(seed)$	This circuit takes as input a *seed* and outputs the tuple $(state, r)$. This circuit executes one H.
$\mathsf{Merkle}.C$	Circuit used to prove the membership to the ring L
$\mathsf{Picnic}.C$	Circuit used to prove the knowledge of a valid Picnic signature
$\mathsf{XMSS}.C$	Circuit used to prove the knowledge of a valid XMSS signature
$\mathsf{PicRS}.C$	Circuit of PicRS construction ($=$ Merkle.C·Picnic.C)
$\mathsf{XRS}.C$	Circuit of XRS construction ($=$ Merkle.C·XMSS.C)

so provides unlinkability between two signatures generated by the same signer. For example, in Figure 6.1, circuit would be $\mathsf{XMSS}.c_2(\mathsf{WOTS}^+.\mathsf{pk}'_6, \mathsf{auth}, \mathsf{idx}) = \mathsf{H}(\mu(\mathsf{H}(\mu(\mathsf{H}(\mu(\mathsf{WOTS}^+.\mathsf{pk}_6, \mathsf{auth}_1, 0)), n1, 1)), n2, 1))$

Algorithm 12 $\mathsf{XMSS}.C$ circuit, for parameters see Table 6.2

$\mathsf{SID} = (\mathsf{WOTS}^+.\sigma, \mathsf{idx}, \mathsf{auth})$
$\mathsf{WOTS}^+.\sigma = (\{\mathsf{WOTS}^+.\sigma_i\}_1^{\mathsf{len}}, r)$

$\underline{\mathsf{XMSS}.c_1}$

Input: $\mathsf{ID}, \mathsf{WOTS}^+.\sigma$
Output: $\mathsf{WOTS}^+.\mathsf{pk}'$

1: $md = (md_1, \ldots, md_{\mathsf{len}_1}) \leftarrow \mathsf{H}(\mathsf{ID}, r)$
2: $c = (md_1, \ldots, md_{\mathsf{len}_2}) \leftarrow \sum_{i=1}^{\mathsf{len}_1}(\mathsf{Wint} - 1 - md_i)$
3: $md = (md||c)$
4: **for** $1 \leq i \leq \mathsf{len}$ **do**
5: $\quad \mathsf{WOTS}^+.\mathsf{pk}'_i = \mathsf{H}^{\mathsf{Wint}-md_i}(\mathsf{WOTS}^+.\sigma_i)$
6: **end for**
7: $\mathsf{WOTS}^+.\mathsf{pk}' = \mathsf{MT}(\{\mathsf{WOTS}^+.\mathsf{pk}'_i\}_{i=1}^{\mathsf{len}})$ (see Table 6.2)
8: **return** $\mathsf{WOTS}^+.\mathsf{pk}'$

$\underline{\mathsf{XMSS}.c_2}$

Input: $\mathsf{WOTS}^+.\mathsf{pk}', \mathsf{idx}, \mathsf{auth}$
Output: b

1: $h_1 \leftarrow \mathsf{H}(\mu(\mathsf{WOTS}^+.\mathsf{pk}, \mathsf{auth}_1, \mathsf{idx}_1))$ ▷Where μ is a multiplexer as described in Section 6.3.1 and presented in 6.7
2: **for** $1 \leq i \leq h$ **do**
3: $\quad hash_i = \mathsf{H}(\mu(h_{i-1}, \mathsf{auth}_i, \mathsf{idx}_i))$
4: **end for**
5: **return** $hash_h == \mathsf{mpk}$

$\underline{\mathsf{XMSS}.C}$

Input: SID
Output: $0/1$

1: $b = \mathsf{XMSS}.c_2(\mathsf{XMSS}.c_1(\mathsf{SID}))$
2: **return** b

$\underline{\mathsf{Merkle}.C}$

Input: $\mathsf{w_{ID}} = (\mathsf{w}_i, b_i)_{i=1}^{\log N}, \mathsf{A.pk}$
Output: $0/1$

1: $a_1 = \mathsf{H}(\mu(\mathsf{ID}, \mathsf{w}_1, b_1))$
2: **for** $i = 2$ to $i = \log N$ **do**
3: $\quad a_i = \mathsf{H}(\mu(a_{i-1}, \mathsf{w}_i, b_i))$
4: **end for**
5: **return** $\mathsf{A.pk} = a_{\log N}$

One-wayness and security of $\mathsf{XMSS}.C$

$\mathsf{XMSS}.C$ is composed of two sub-circuits presented in Algorithm 12. The one-wayness of $\mathsf{XMSS}.C$ ensues directly from the unforgeability of XMSS used by PKG and of the one-wayness of the hash function. Indeed according to [HBGM15], the security of XMSS depends on the one-wayness of the hash function, it is computationally infeasible to compute the WOTS^+ public key from the XMSS tree root which is mpk. As it is computationally infeasible to invert $\mathsf{XMSS}.c_2$ from mpk and also infeasible $\mathsf{XMSS}.c_2$.

The information leaked by the stateful nature of XMSS is hidden by integrating the multiplexer μ to circuit $\mathsf{XMSS}.c_2$. Because of the multiplexer, the position of the WOTS^+ signature in PKG's XMSS tree is hidden and avoids the possibility to link two signatures generated by the same signer.

6.3.3 Applicable post-quantum IDRSs from symmetric primitives

This section introduced three possible circuits: $\mathsf{Merkle}.C$, which can take the role of C_1, $\mathsf{Picnic}.C$ and $\mathsf{XMSS}.C$, which can both be implemented as C_2. From these circuits, we can implemented two different IDRSs.

Post-quantum IDRS from Picnic named PicRS

PicRS follows the generic construction presented in Section 6.2.1 and uses Picnic as digital signature DS. This means that the master public and secret keys mpk and msk are set to $\mathsf{msk} = \mathsf{Picnic.sk}$ and $\mathsf{mpk} = \mathsf{Picnic.pk}$ and each user with an identity ID owns a signing secret key $\mathsf{SID} = \mathsf{Picnic}.\sigma$ such that $1 = \mathsf{Picnic.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$. In PicRS, the general circuit C (see Figure 6.4) named $\mathsf{PicRS}.C$ is where the signer executes the NIZK proof. We have circuit $C = \mathsf{PicRS}.C$ where $C = \mathsf{PicRS}.C = \mathsf{Merkle}.C \cdot \mathsf{Picnic}.C$. PicRS security comes from the EU-CMA security of Picnic proven in [Pic], from the properties of the Merkle accumulator and from the one-wayness of circuits $\mathsf{Merkle}.C$ and $\mathsf{Picnic}.C$ discussed in Section 6.3.3.

Post-quantum IDRS from Picnic XMSS named XRS

The idea of the XMSS-based IDRS construction is similar to the PicRS construction but the Picnic digital signature is replaced with the XMSS signature. This means that in XRS, each signer executes NIZK on circuit $C = \mathsf{XRS}.C = \mathsf{Merkle}.C \cdot \mathsf{XMSS}.C$, where $\mathsf{XMSS}.C$ will prove the knowledge of a valid XMSS signature (i.e. $1 = \mathsf{XMSS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$). XRS security comes from the EU-CMA security of XMSS proven in [HBGM15], from the properties of the Merkle accumulator and from the one-wayness of circuits $\mathsf{Merkle}.C$ and $\mathsf{XMSS}.C$ discussed in Section 6.3.3.

6.4 Evaluation

This section analyzes the signature sizes of both applicable constructions PicRS and XRS for a post-quantum security level of $\lambda = 128$ bits. We evaluate both constructions using two different non-interactive zero-knowledge proof systems (NIZK): KKW [KKW18] and Ligero++ [BFH+20]. We chose to work with these NIZK because, on the one hand, KKW is considered the current state-of-the-art of symmetric based NIZK. It has been analyzed in the QROM model and it is considered by NIST as an alternate candidate for the standardization process [AASA+20]. On the other hand, Ligero++ has been less studied and its post-quantum security is only assumed, but it achieves a competitive proof size for large circuits. KKW is optimized to work on binary circuits while Ligero++ is optimized for arithmetic circuits (see Table 6.3). We implemented our schemes using different hash functions which are either considered as binary or as arithmetic circuits (See Table 6.4). We use the standard hash function SHA3 but we also tested our constructions with other hash functions which have an optimized complexity that decreases the overall size of the circuit and of the signature. The complexity of all circuits are expressed in terms of "number of H executions" and we consider an execution of the compression function $\mathsf{H}(x, y) = z$ with $x, y, z \in \{0,1\}^{2\lambda}$. If H has n inputs, the number of counted execution is $n - 1$.

TABLE 6.3: NIZKs and their associated type of circuit. :$|C|$= circuit size/complexity, $|\mathsf{AND}|$= number of multiplication gates in circuit C

NIZK	Type of circuit	Proof Size (Asympt.)
KKW [KKW18]	Binary	$\mathcal{O}(\|\mathsf{AND}\|)$
Ligero++ [BFH+20]	Arithmetic	$\mathcal{O}(\log^2(\|C\|))$

TABLE 6.4: Hash functions' complexity, A=arithmetic, B=binary

Hash	Type	Complexity	
		Mult.	Add.
SHA3	A/B	38400	422400
LowMC [ARS+15]	B	1374	30134110
Poseidon [GKR+21]	A	600	
MiMC [AGR+16]	A	1293	646
μ (Equ. 6.7)	A/B	1024	512

In the rest of this section, we discuss the complexity of circuit $\mathsf{Merkle}.C$. We then start the discussion on optimizing circuit $\mathsf{Picnic}.C$ and $\mathsf{XMSS}.C$ in order to have the shortest signature possible for PicRS and XRS. We express the complexity in terms of number of hash executions. The total complexity of our circuits can be computed with the help of Table 6.4 depending on the used H. We conclude this chapter with a comparison between both schemes and a comparison with the current state-of-the-art of post-quantum IDRS constructed with lattices and some final recommendations.

6.4.1 $\mathsf{Merkle}.C$ complexity

The complexity of $\mathsf{Merkle}.C$, which is used to "accumulate" all identities in the ring L increases logarithmically ($\mathcal{O}(\log N)$) with the size of the ring. This ensues from the Merkle tree structure of the accumulator (see Section 6.3.1). The complexity of circuit $\mathsf{Merkle}.C$ can be expressed as $\underline{|\mathsf{Merkle}.C| = \log N \cdot (|H| + |\mu|)}$, where $|H|$ is the complexity of the hash function H, $|\mu|$ is the complexity of the multiplexer (see Equation 6.7), and N the ring size. $\mathsf{Merkle}.C$ is implemented in both constructions, therefore this discussion is valid for both PicRS and XRS.

6.4.2 PicRS signature's size

The signature size of PicRS depends on the complexity of the circuit $\mathsf{PicRS}.C = \mathsf{Merkle}.C \cdot \mathsf{Picnic}.C$ (see Section 6.3.3). To analyze the PicRS signature size, we need to investigate circuit $\mathsf{Picnic}.C$ described in Section 6.3.2 and in Algorithm 11. We optimize the complexity circuit $\mathsf{Picnic}.C$ by testing different parameters, proposed in their last paper [KZ20], for the Picnic digital signature used by PKG. We express the complexity of $\mathsf{Picnic}.C$ in function of executions of hash functions H and G, of number of multiplication and addition gates. We consider that an execution of H has two inputs and therefore the number of calls of H grows linearly with the number of inputs. Table 6.5 demonstrates that $\mathsf{Picnic}.C$ achieves its optimal circuit size for the Picnic scheme with the parameters $n = 3$, $\tau = 438$, and $M = 438$. This comes principally from the fact that such an instance does not need to execute sub-circuit $\mathsf{Picnic}.c_2$ as $M = \tau$. Therefore, for the rest of the analysis of PicRS, we use Picnic parameters ($n = 3$, $\tau = 438 = M$) as the digital signature for PKG. Table 6.1 shows

TABLE 6.5: Picnic.C's complexity for different Picnic parameters n, M, and τ. It shows the number of executions for the hash function H and G, the multiplication (Mult.) and addition (Add.) gates

Picnic			Circuits complexity									
n	M	τ	|Picnic.c_1|	|Picnic.c_2|	|Picnic.c_3|			|Picnic.c_4|	|Picnic.C|			
			H	H	H	Mult.	Add.	G	H	G	Mult.	Add.
3	438	438	1752	0	5256	2680560	244831488	1	7008	1	2680560	244831488
16	604	68	2040	1673392	4352	3121200	285077760	1	1679784	1	3121200	285077760
64	803	50	6300	2495442	12800	9639000	880387200	1	2514542	1	9639000	880387200

TABLE 6.6: XMSS.C circuit sizes for each sub-circuit for different Wint and len.

XMSS			Circuits complexity		
Wint	len	h	|XMSS.c_1|	|XMSS.c_2|	|XMSS.C|
4	133	20	$665 \cdot \|H\|$	$20 \cdot (\|H\| + \mu)$	$\mathbf{685} \cdot \|\mathbf{H}\| + \mathbf{20} \cdot \|\mu\|$
16	67	20	$1139 \cdot \|H\|$		$1159 \cdot \|H\| + 20 \cdot \|\mu\|$
64	44	20	$2880 \cdot \|H\|$		$2900 \cdot \|H\| + 20 \cdot \|\mu\|$

the signature sizes of PicRS for different ring sizes with different hash functions and NIZKs.

6.4.3 XRS signature's size

XRS signature size depends on the complexity of circuit XRS.C composed of sub-circuits XMSS.C and Merkle.C (see Section 6.3.3) on which the NIZK.Prove algorithm is executed. In this part, we focus on the specific sub-circuit XMSS.C and its internal sub-circuit XMSS.c_1, which is composed of $\mathsf{len} \cdot \mathsf{Wint} + \mathsf{len} - 1$ executions of the hash function H and the second sub-circuit XMSS.c_2 consisting of h calls to H and the multiplexer μ. Table 6.6 presents the complexity of circuit XMSS.C with different parameters for the XMSS scheme used by PKG to generate all secret signing keys. We used parameters proposed by XMSS's original paper [HBGM15]. Our results demonstrates that XMSS should be implemented with the parameters $\mathsf{Wint} = 4$, $\mathsf{len} = 133$, and $h = 20$ to optimize the complexity of XMSS. It is important to note that 2^h is the maximum number of SID that can be generated by PKG in this case. The detailed reason of setting $h = 20$ is presented in Section 6.4.4.

After optimizing the complexity of the XMSS.C, we evaluate the signature size of XRS for different group sizes and with different NIZKs and hash functions. The details of XRS features and results are presented in Table 6.1.

6.4.4 PicRS vs XRS

Choice of NIZK

As presented in Table 6.1, our implementations using Ligero++ as a NIZK are the best options when targeting a signature size optimization. It works better for large circuits as its proof size grows logarithmically with the circuit size while KKW grows linearly with the number of multiplication (or "AND") gates in the circuit. For this reason, our Ligero++-based implementations can use the standard hash function SHA3 and still achieve a decent signature size while our KKW-based implementation requires a specifically designed hash function (LowMC) to be competitive. To the best of our knowledge, KKW is optimized for binary circuits similar to LowMC while Poseidon and MiMC are arithmetic circuits that work over larger finite fields, which makes

them unpractical for KKW. However, KKW has been submitted to the NIST standardization process [AASA+20] and therefore gives stronger security guarantees than the two other NIZK. Ligero++ was only published recently and its post-quantum security has been assumed as it relies on known post-quantum paradigms, but it has not been proven.

Signature size

Table 6.1 summarizes the performance of both schemes in terms of signature size. XRS clearly outperforms PicRS due to the lower complexity of circuit XMSS.C (see Table 6.6) used in XRS compared to Picnic.C (see Table 6.5) implemented in PicRS. Even if in theory both signature sizes should increase logarithmically with the ring size, we observe that both schemes provide a nearly constant signature size because the circuit complexity of PicRS.C and XRS.C depends mainly on Picnic.C and XMSS.C. Picnic.C represent 99% of PicRS.C complexity and XMSS.C 95% of XRS.C's complexity for the largest ring $N = 2^{20}$. Because of Ligero++ proof size that increases only logarithmically with the circuit size while KKW's one increases linearly with the number of multiplication gates, PicRS and XRS implemented with Ligero++ have a signature size "more constant" than the ones implemented with KKW (see Table 6.1). A possible optimization for XRS could be replacing the WOTS$^+$ scheme by a few-time signature scheme named FORS [BHK+19] in the XMSS scheme used by PKG, which should further reduce the signature size for XRS. However, this assumption requires a formal security analysis.

PKG characteristic

In PicRS, PKG enjoys the stateless feature of Picnic and therefore does not need to update his secret key after a signature as it is required for XRS. The main advantage of PicRS over XRS is that PKG can theoretically generate an infinite number of signing secret keys, so can handle an infinite number of users, while XRS is limited to 2^h users. Our implementation showed in Table 6.1 sets $h = 20$ due to the computation complexity of generating a XMSS tree (e. g. IDRS.Setup algorithm) which is grows exponentially with h.

Comparison with lattice-based IDRS

Table 6.1 also highlights the competitiveness of XRS and PicRS when it comes to signature sizes compared with lattice-based IDRS. It is important to highlight that none of the lattice-based works gave a precise signature size. We estimated the signature size of Zhao et.al. [ZT18] work according to their formula. We fixed their parameters to $n = 1000$ (n is their security parameters for the short integer solution problem (SIS)), $q = 2^{40}$, $w = 3$ and $k = 41$. Their estimated size is presented in Table 6.1. Regardless of the difference of the actual signature size, XRS and PicRS enjoy a nearly constant signature while all current state-of-the-art of lattice constructions [ZT18], [CYH21],[Wan08], [WDZ+15] have a signature size increasing linearly with the ring size N. Therefore, all of our implementations shown in Table 6.1 are more suitable for large rings than the lattice-based IDRS.

Final recommendations

Table 6.1 shows that XRS implemented with Ligero++ is our most promising construction when an optimized signature size is desired. Ligero++-based constructions

offer an excellent compromise between signature size and security guarantees. It achieves a competitive signature size with hash functions Poseidon, MiMC and even with the standard hash SHA3. As illustrated in Table 6.1, XRS outperforms PicRS with a smaller signature size and a smaller SID that comes from the difference in size between XMSS and Picnic. It is also important to highlight that our possible constructions have been evaluated theoretically and it would be interesting to investigate the applicability with an implementation. According to KKW [KKW18] and Ligero++[BFH+20] original papers the circuit's complexity influences the memory of the signer while generating the proof. This increases the advantage of XRS over PicRS. Due to the extremely high complexity of our circuits, a signer in both Ligero++-based and KKW-based IDRS could run out-of memory when computing the proof. The new designed Mac'n'Cheese [BMRS21] zero-knowledge proof could be a solution to this problem but it would increase the signature dramatically and, to the best of our knowledge, it requires lattices to achieve post-quantum security. Our final recommendation would be to use XRS implemented either with Poseidon and Ligero++ to achieve the shortest signature size possible or with KKW combined with LowMC to ensure proven post-quantum security.

6.5 Chapter's conclusion

This chapter presents the contributions related to Research Objective O2 and our final results are presented in Table 6.1 on page 107. We designed the first generic construction for an ID-based ring signature (IDRS) based on symmetric primitives. We introduced a one-way generic circuit C which allows to prove the knowledge of a signing secret key SID belonging to a user with identity ID whose identity belongs to the ring L. A signing secret key SID is a digital signature generated by the private key generator (PKG). PKG signs the identity ID to generate the signing secret key SID.

We then derived two possible IDRSs instances of our generic construction named PicRS and XRS. We constructed a Merkle accumulator from which we derived the one-way circuit Merkle.C to allow the signer to prove that his identity belongs to the ring L. Then, we construct the circuit Picnic.C to prove the knowledge of a valid Picnic signature generated by PKG. Circuit PicRS.C = Merkle.C & Picnic.C is at the heart of the first IDRS PicRS. Our second construction XRS follows the same approach but it utilizes XMSS instead of Picnic to generate the signing secret keys. Therefore, at the heart of XRS, there is circuit XRS.C = Merkle.C & XMSS.C, where XMSS.C aims to prove the knowledge of a valid XMSS signature.

The applicability of both PicRS and XRS were evaluated and it appears that XRS outperforms PicRS when it comes to the signature sizes. This comes from the lower complexity of circuit XMSS.C in comparison to Picnic.C. Both contruction also enjoys a nearly constant signature size, in other words a signature size that is not dependent on the number of members in the ring. This an important advantage over the lattice-based IDRSs who have a signature size that increases linearly with the ring and therefore are not applicable to a large ring. Both PicRS and XRS were evaluated with two different NIZKs and four different hash function. Our evaluation lead us to the conclusion that XRS implemented with Poseidon as a hash function and Ligero++ as NIZK is the best way to achieve the shortest signature size possible. Hovewer, when we desire **proven** post-quantum security, we suggest replacing Ligero++ with KKW and Poseidon with LowMC, as KKW has been prove quantum-safe in its original papers [KKW18, Pic].

Chapter 7

Conclusion

The growth of the digital world's importance increases the need for privacy and security and the need for protocols protecting users has never been more critical. Moreover, the highly probable appearance of quantum computers will create a more powerful threat to this digital world. Researchers have invested considerable effort to design quantum-safe protocols because of the imminence of this quantum threat. This thesis investigates the need for privacy protocols resisting quantum computers. We identified that among the possible post-quantum candidates, symmetric primitives appear to be the most interesting option for a number of reasons. They are efficient on classical computers, which is essential because we need to implement quantum safe protocols on these classical computers. Indeed, we need to shift to quantum security before the actual appearance of quantum machines, otherwise, all non-quantum safe protocols would be not secure when the democratization of quantum computers happen. They are relatively old primitives and therefore, their security is well understood by the research community, which offers security guarantees. Another element that convinced us to work with symmetric-based protocols is that their security depends only on the symmetric primitives used. This means that, if the used primitives have been broken, the implemented protocol can still be secure by using another symmetric primitive. The final motivation of this research was to determine whether symmetric-based schemes provided an applicable alternative to the well-researched area of lattice-based schemes.

To ensure privacy, we decided to concentrate our effort on designing group and ring signatures. Group signatures allow the member of a group to sign a message anonymously. A group is supervised by a central authority that allows new members to join. Ring signatures work similarly to group signatures but there is no central authority and, therefore, a member of the group, a ring in this case, can sign anonymously in a spontaneous way without any joining procedure. For these reasons, this research project is organized around the two following questions:

(1) *How to design applicable post-quantum group signatures from symmetric primitives?*

We wanted to design a fully dynamic group signature to allow members to join and leave the group at their will. The main aim was to reduce the signature size and its dependency with the group size. We also wanted to optimize the efficiency of the algorithms as much as possible.

(2) *How to design applicable post-quantum ring signatures from symmetric primitives?*

We wanted to design a ring signature. The main aim was to reduce the signature size and its dependency with the ring size. We also wanted to optimize the efficiency of the algorithms and the range of applications as much as possible.

7.1 A road to an answer

This thesis contributes to the research community by designing four different constructions that provide anonymity and resist attacks coming from quantum computers. These constructions are group and ring signature schemes constructed exclusively using symmetric primitives similar to cryptographic hash functions or block ciphers. This research project fills a research gap as most post-quantum group and ring signatures were designed with lattice-based cryptography and existing symmetric-based schemes are limited in terms of applications. Moreover, this project was also motivated by interesting features coming from symmetric primitives including their efficiency on “classical" computers, which is crucial as we need to shift to quantum-safe protocols before the democratization of quantum computers. In this thesis, we designed three group signatures and proposed one generic ID-based ring signature.

7.1.1 Research Objective O1: Group Signature

Research Objective O1 firstly focused on achieving the smallest signature possible before trying to optimize the algorithm performance to achieve the best possible applicability. All of our contributions are presented in blue in Table 7.1.

First contribution: Dynamic & Revocable Group Merkle signature (DGM)

DGM, presented in Chapter 3, is the first fully dynamic post-quantum group signature designed in this project. It is an extension of the static group signature G-Merkle [EBM18], which is very limited in terms of application but offers a competitive signature size. Therefore, we extended it to improve its applicability. The main advantage of DGM is its constant signature size (with respect to the number of group members N) compared to the current state-of-the-art. Table 1.1.6 demonstrates this trend. When compared to the benchmark of group signatures with symmetric primitives [KKW18], DGM achieves a signature size 115 times smaller than [KKW18], as their smallest signature size provided is 315KB for a group of 2^7 members.

Another advantage of DGM is that a valid signature remains valid during the whole life of the group, which is not the case in either [KKW18] or [BEF19] because the element of the public key used to prove the membership is updated after each member joins the group, meaning that signatures generated previously are no longer valid. In DGM, the Merkle root of the main tree allows a member to prove its membership and this never changes, regardless of new members joining. Additionally, full dynamicity is achieved thanks to an innovative procedure using the primitive named Symmetric Puncturable Encryption (SPE) [SYL+18], which avoids the use of lists of revoked members that can be impractical for large groups. However, the DGM verification has a major cost due to the addition of the fully dynamic feature. The dynamicity brings an interaction between $\mathcal{M}$ and the verifier, which could be an important limitation for possible applications.

To summarize, the main contributions ensuing from DGM are:

- A new fully dynamic group signature,
- a competitive signature size which also does not depend on the group size,
- an innovative revocation procedure through Symmetric Puncturable Encryption.

Second contribution: (Traceable-) SPHINCS Group Signature ((T)-SGS)

(T-)SGS was designed to solve the issue of DGM, namely the use of a state and the interactive protocol. The innovation behind this scheme is the transformation of the competitive digital signature SPHINCS+ into a GS. (T-)SGS is presented in Chapter 4.

SGS is a transformation of SPHINCS+ into a group signature without the traceability property. Its results, presented in Table 7.1, show that SGS has two significant advantages compared to the other group signatures that do not provide traceability [BEF19]. First, the group size does not affect the size of the signature as it only depends on the setup parameters. Second, the practical size achieved by SGS is significantly shorter than the minimum one that can be achieved by [BEF19].

Table 7.1 demonstrates the size achieved by the traceable group signature, T-SGS, for different group sizes. The cost of adding traceability is an increasing signature size (see Table 7.1), which has for consequences that T-SGS is outperformed by the work of Katz et. al. [KKW18]. Even if we provide a full dynamicity (see Table 7.1) compared to them, our deficit in terms of signature size makes T-SGS less applicable. Despite its initial success in removing the interaction and the state compared to DGM, the design of T-SGS has an important issue. The only way to ensure traceability, in other words, to go from SGS to T-SGS, is to integrate an accumulator to the SPHINCS+ design. This makes our innovative use of SPHINCS+ redundant. This contribution demonstrates that the only possibility to design a stateless group signature from symmetric primitives is to build an accumulator with a non-interactive zero-knowledge proof. For this reason, we showed the limitations of using SPHINCS+ to design a group signature. To summarize, the contributions ensuing from (T-)SGS are:

- A new approach to design a non-traceable group signature with SPHINCS+,
- a new stateless fully dynamic group signature named T-SGS.

Third contribution: Efficient Post-Quantum Group Signature from Symmetric Primitives with User-Controlled Linkability (xGS)

xGS, introduced in Chapter 5, was designed to counter the failure of T-SGS while still solving the issues of DGM. Similar to DGM, xGS is constructed from the idea of combining multiple Merkle trees. xGS is also stateful but the signature size appears to be so competitive compared to the current state-of-the-art (see Table 7.1) that it makes the stateful feature “acceptable". xGS enjoys a constant signature size and outperforms both lattice-based and symmetric-based constructions, except for DGM. More importantly, xGS has a non-interactive verification system, which is an improvement compared to DGM, and the level of trust in the manager has been reduced. While the manager can frame honest group members in DGM and (T-)SGS, this is not the case in xGS. Finally, xGS provides user-controlled linkability, a property that gives group members the opportunity of choosing if they want their signature to be linkable or not. This feature allows group members to choose between a high level of privacy or better applicability. User-controlled linkability is the desired property in applications like Direct Anonymous Attestation (DAA) [BFG+13, CDL16]. We also believe that online survey systems are favorable to have this feature as the additional linkability may provide extra information to the organization (e.g. the same customer goes shopping in different stores or different dates). The recent work presented by Diaz et al. [DL21] highlights the interest of user-controlled linkability with a possible application for smart vehicles in Intelligent Transportation Systems where data (i.e. messages)

are sent and used to detect any anomaly. A driver could choose to make their data linkable to help detect any problem with their vehicle while remaining anonymous. Another driver could also choose to keep their data unlinkable to have a higher level of privacy. To the best of our knowledge, xGS is the only group signature providing user-controlled linkability that is resistant against quantum attacks. To summarize, xGS brings the following contributions:

- a non-interactive fully dynamic group signature from symmetric primitives,
- the first post-quantum group signature providing user-controlled linkability,
- a short signature size, which does not depend on the group size,
- a reduced level of trust in the central authority.

Final words regarding Research Objectives O1

The work done for Research Objective 02 generated three group signature schemes and this part aims to address the final recommendation for our group signatures. The group signature xGS presented in Chapter 5 is the one we recommend using. It achieves the best trade-off between algorithm efficiency and signature size. xGS achieves a signature size of 14KB larger than DGM presented in Chapter 3. Nevertheless, xGS has a non-interactive verification procedure and the level of trust in the manager is lower as it is infeasible for her to forge a signature. In DGM, the manager needs to be consulted for the verification of a signature and could forge a signature when corrupted, which means that DGM is designed on a weaker notion of unforgeability than xGS. We believe that a high level of trust in the manager is still acceptable as it can be seen as a trusted third party. We are convinced that xGS's stateful feature is acceptable because of its signature size which is at least 300KB shorter than the best stateless option [KKW18] and does not depend on the number of group members. If the stateless feature is required, then we suggest using the work proposed by Katz et al. [KKW18] which outperforms our stateless T-SGS. Normally, traceability is considered a required property for group signature. But in certain specific cases such as for Anonymous Attestation [BEF19], this feature is not desired. In such cases our non-traceable stateless group signature SGS (see Chapter 4) can be handy even if SGS was originally designed as an intermediate step for the transformation of the digital signature SPHINCS+ [BHH+15, BHK+19] towards our group signature T-SGS.

In this thesis, we discuss only two techniques of revocation. The first one has been designed in this project and is based on Symmetric Puncturable Encryption (SPE) (see Chapter 3.3). The second one is the "traditional" revocation technique of establishing a public list of revoked members and having the verifier go through the list when verifying the validity of a group signature. As discussed in Chapter 3.4.4, our SPE-based technique, despite being promising, implies extensive computation and memory complexity. This lack of efficiency motivated us to use the traditional approach for the other group signature.

It is important to highlight that there exist other techniques to provide revocation such as Boneh et al [BS04], but not post-quantum secure and lattice-based Langlois et al. [LLNW14], which are both based on homomorphism. Therefore it is not possible to provide revocability using their approach when using symmetric primitives. To the best of our knowledge, there are no other post-quantum revocation techniques based on symmetric primitive, other than the one discussed in this thesis.

7.1.2 Research Objective O2: Ring Signature

In this project, we designed a generic construction of post-quantum ID-based ring signature (IDRS)(see Chapter 6) while targeting the resolution of our second research objective. The ID-based feature has been added to extend the range of applicability of ring signatures (O2(d)) as it improves the spontaneity of the ring signature compared to the traditional approach.

Fourth Contribution IDRS : Post-Quantum ID-based Ring Signatures from Symmetric-key primitives

The idea behind the generic scheme is to use the zero-knowledge proof system (NIZK) based on symmetric primitives [KKW18] because it allows us to prove the knowledge of the input w such that $C(w) = 1$ for a one-way circuit C. To build our generic IDRS, we designed a circuit C that proves membership to a ring and the knowledge of a valid signing secret key SID and executes the NIZK proving procedure. SID is generated by the private key generator (PKG), which computes $\mathsf{SID} \leftarrow \mathsf{DS.Sign}(\mathsf{ID}, \mathsf{msk})$, where ID is the user's identity, msk the master secret key and DS a digital signature scheme. Therefore, the circuit C needs to prove that $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$ (mpk= master public key) and that ID belongs to the ring (a set of IDs). Based on the generic construction, we designed two IDRS constructions. The first construction, named PicRS (see Chapter 6.3), uses Picnic [CDG+17, Pic] as a digital signature scheme combined with a Merkle accumulator to prove the membership proof on which the zero-knowledge proof system is performed. The second construction, named XRS (see Chapter 6.3), implements the stateful signature scheme XMSS [HBGM15] instead of Picnic. This change reduces the size of the signature significantly, as illustrated in Table 7.2.

We also implemented both PicRS and XRS with different NIZKs, Ligero++ [BFH+20] and KKW [KKW18], and with different hash functions, Poseidon [GKR+21], LowMC [ARS+15], MiMC [AGR+16] and the standard function SHA3-256. The results, summarized in Table 7.2, demonstrate that XRS achieves the shortest signature size when implemented with Ligero++ and Poseidon. However, if a proven post-quantum security is desired, then KKW and LowMC should be used.

Both schemes enjoy a nearly constant signature size because the bottleneck of both schemes is to prove $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$, which remains constant regardless of the size of the ring. When compared with the current state-of-the-art of post-quantum IDRS (all lattice-based), both our constructions outperform lattice-based constructions, particularly for large signature size, and they have the immense advantage of a nearly constant signature size in contrast to the lattice-based IDRS signature size grows linearly. It is important to note that none of the other post-quantum IDRS provide an actual signature size, therefore, we approximated the size of [ZT18] thanks to the parameters given in their paper.

It is important to notice that from the instance proposed in Table 7.2, only constructions based on KKW have a proven post-quantum security. For other constructions, the post-quantum security is only assumed but has not been proven at this stage. For example, to the best of our knowledge, the hash function Poseidon has not been proven post-quantum secure but satisfies all properties of a hash function and has been published in a top conference. We included post-quantum schemes without proofs because they have an immense potential for improving our constructions' performances and are likely to be quantum-safe.

Our contributions and their comparison with the current state-of-the-art are presented in Table 7.2. To summarize, the contributions ensuing from the pursuite of Research Objective O2 are:

- The first post-quantum ID-based ring signature constructed from symmetric primitives,
- the first two applicable IDRSs: PicRS and XRS,
- a nearly constant signature size for both PicRS and XRS,
- a shorter signature size compared to lattice-based constructions (see Table 7.2) for both PicRS and XRS.

It is important to mention that sub-objective O2 (d) has not been achieved as presented in Chapter 1.3.2. We wanted to extend the designed ring signature by adding linkability, but we constructed an ID-based ring signature instead of a classical ring signature as initially targeted. The tools available with the symmetric primitives did not allow us to improve the state-of-the-art of ring signatures designed by Katz et al. [KKW18]. Therefore, we opted to add the ID-based feature which offers a better spontaneity to ring signature as it avoids the use of certificates for the users keys.

TABLE 7.1: Comparison of different post-quantum group signatures for 128-bits of post-quantum security. N denotes the number of members in the group, N.A.=Non Applicable, B=Maximum possible signatures per member, low= The group manager $\mathcal{M}$ cannot frame a signature, high=$\mathcal{M}$ can frame a signature , x= number of unlinkable signatures that can be generated by each user

Schemes		[DPLS18]	[EZS+19]	[BEF19]	[KKW18]	[EBM18]	DGM	SGS	T-SGS	xGS
$\lvert\sigma\rvert$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
$\lvert\sigma\rvert$ (KB)	$N = 2^7$	581	39	1370	315	2.72	2.82	33	359	16.14
	$N = 2^{10}$	581	54	1850	418	N.A.	2.82	33	501	16.14
	$N = 2^{20}$	581	140	3450	770	N.A.	2.82	33	970	16.14
$\lvert\mathsf{usk}\rvert$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log B)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\mathsf{x})$
$\lvert\mathsf{usk}\rvert$ (KB)		146	0.059	$0.032 \log N$	$0.032 \log N$	$2.2 \log B$	$2.8 \log B$	275	$275 + 0.032 \log N$	$\mathsf{x} \cdot 13.13$
$\lvert\mathsf{gpk}\rvert$ (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
$\lvert\mathsf{gpk}\rvert$ (KB)		123	$N \cdot 11.1$	0.032	0.032	0.032	0.032	0.032	0.032	0.032
Sign (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
Sign (ms)[a]		450	No data	No data	3000	N.A.	1	1244	7383	1496
Verify (asympt.)		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(\log BN)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
Verify (ms)[a]		169	No data	No data	3000	N.A.	Interactive	0.1	3886	11
PQ canditate		Lattice	Lattice	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric	Symmetric
Stateless		Yes	Yes	Yes	Yes	No	No	Yes	Yes	No
Dynamic		Static	Partially	Static	Fully	Static	Fully	Fully	Fully	Fully
Traceable		Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes
Non-interactive		Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Level of Trust in $\mathcal{M}$		Low	Low	Low	Low	High	High	High	High	Low

[a]The times provided are for $N = 2^{10}$ and taken from the original papers and were implemented in different programming languages and executed on different types of machines

TABLE 7.2: PicRS and XRS comparisons. PQ=post-quantum, N= ring size, ✓=proven, (✓)= assumed, h=XMSS tree height

IDRS	PQ candidate	$\|\sigma\|$ (Asympt.)	Max. N	IDRS.Setup (Asympt.) time	|SID| (KB)	NIZK	PQ	H	(est.)$\|\sigma\|$(MB)		
									$N = 2^6$	$N = 2^{12}$	$N = 2^{20}$
PicRS	Symmetric	$\mathcal{O}(\log N)$	unli.	$\mathcal{O}(1)$	167	KKW	✓	LowMC	169.964	170.153	170.406
								SHA3	3619	3622	3626
						Ligero++	(✓)	SHA3	2.046	2.046	2.047
								MiMC	1.902	1.902	1.903
								Poseidon	1.898	1.899	1.900
XRS	Symmetric	$\mathcal{O}(\log N)$	2^{20}	$\mathcal{O}(2^h)$	4.899	KKW	✓	LowMC	12.487	12.680	12.930
								SHA3	332.300	335.266	339.211
						Ligero++	(✓)	SHA3	1.490	1.491	1.493
								MiMC	0.973	0.976	0.979
								Poseidon	0.885	0.889	0.893
[ZT18]	Lattice	$\mathcal{O}(N)$	unli.	-	615000	-	✓	-	5	335	32243

7.2 Future works & final words

Our work and contributions generate new possible future works and future research directions. Our three main contributions related to group signature demonstrate that stateful group signatures, similar to DGM (see Chapter 3)or XGS (see Chapter 5), are the best way to achieve short signature size and, therefore, they have a better chance to be implemented in a real-world application. A possible research direction would be to investigate applications where the state is either necessary or not a threat to the security. An example that we have in mind would be blockchain, where we could try to link the state of the group signature and the state of the blockchain as it has been done in [BDE+21] with stateful verifiable random functions (VRF). Linking the state of the protocol, in which our group signature is implemented, with the state of each group member would avoid mistakes by a member that may lead to a breach in the security.

Another area that this project highlights is user-controlled linkability, a feature of our last group signature xGS (see Chapter 5). We proposed two possible applications with smart cars or the customer experience (see Chapter 5.5). Future works could investigate the utility of user-controlled applicability in different applications. We believe that applications, where a user could chose between different levels of security and a better applicability like the costumer experience that we discussed in Chapter 5, could have an interest to develop such property.

As stated before, group signatures based on symmetric primitives tend to be stateful or to have a large signature size, which hinders their applicability compared to lattice-based constructions. Chapter 4 demonstrated that the only way to construct stateless group signatures is to combine an accumulator with a non-interactive zero-knowledge proof system. This work creates a new direction for the research community, which could investigate ways to improve the complexity of non-interactive zero-knowledge proof systems or accumulators to compete with lattice-based constructions, that currently achieve a better performance.

When it comes to future works related to our second Research Objective (O2), the promising technical and theoretical contributions of our generic ID-based ring signature (IDRS) from symmetric-key primitives could motivate the research community to develop an implementation of our proposed instances PicRS and XRS (see Chapter 6.2). An implementation could indeed confirm the great promise of both schemes as well as help to determine which primitives (non-interactive zero-knowledge proof system and digital signature scheme) offer the better implementation option.

Finally, our nearly constant signature size could motivate the research community to extend our scheme by adding the linkability property, which is a required property when implementing a ring signature in a blockchain-based technology. Due to the rise of blockchain-based technologies, this should be very a promising research direction.

We strongly believe that the way to construct group and ring signature by using only hash function without requiring any NIZK - which is the case for DGM (see Chapter 3), xGS (see Chapter 5) or G-Merkle [EBM18] - has reached its limit in terms of performances and possible constructions. The non-flexibility of using hash function only made it difficult to reduce the signature or improve the performance of algorithms.

Additionally, we believe that Katz et al. presents the most efficient way to design group and ring signatures with symmetric-based NIZK and the room of improvement to try another way to construct them would be unproductive. Therefore we suggest that researchers should invest their time on trying to design new symmetric primitives to improve the performance of the ring/group signature.

We suggest that researchers should focus on constructing new hash functions with the goal of optimizing their circuit complexity, or in other words, the goal of reducing as much as possible the number of logical gates in the circuit. An example is the work done by Grassi et al. [GKR+21] with their hash function Poseidon. We suggest that this kind of work is extremely promising as hash function with optimized complexity will improve the performances of NIZK. Another approach could be to improve the NIZK and reduce their dependency to the complexity of the hash function and improve the performance of group/ring signature. This research direction has been taken in the works presented by Beullens [BdSG20] and Baum et al. [BSGK+21]. Designing new hash functions or new symmetric-based NIZK also have the advantage of improving other applications beyond just group and ring signatures.

All in all, this thesis improves the current state-of-the-art of post-quantum group and ring signatures and offers an alternative to the larger number of lattice-based constructions. It also demonstrates that post-quantum schemes constructed with symmetric primitives have real potential for applications and deserve additional commitment from the research community to improve the applicability and preserve the privacy of users as we head towards the democratization of the quantum computers.

Bibliography

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the second round of the nist post-quantum cryptography standardization process. *NIST, Tech. Rep., July*, 2020.

[ABCG17] Quentin Alamélou, Olivier Blazy, Stéphane Cauchie, and Philippe Gaborit. A code-based group signature scheme. *Designs, Codes and Cryptography*, 82(1-2):469–493, 2017.

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Annual international cryptology conference*, pages 255–270. Springer, 2000.

[ACMa] ACM ASIACCS 2022. https://asiaccs2022.conferenceservice.jp/.

[ACMb] ACM CSUR 2022. https://dl.acm.org/journal/csur.

[ACN] ACNS 2022. https://sites.google.com/di.uniroma1.it/acns2022/.

[ACST06] Man Ho Au, Sherman SM Chow, Willy Susilo, and Patrick P Tsang. Short linkable ring signatures revisited. In *European Public Key Infrastructure Workshop*, pages 101–115. Springer, 2006.

[AGR+16] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*, pages 191–219. Springer, 2016.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS*, pages 2087–2104. ACM, 2017.

[ALYW06] Man Ho Au, Joseph K Liu, Tsz Hon Yuen, and Duncan S Wong. Id-based ring signature scheme secure in the standard model. In *International Workshop on Security*, pages 1–16. Springer, 2006.

[ARS+15] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In *EUROCRYPT*, pages 430–454. Springer, 2015.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55. Springer, 2004.

[BCC+15] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on ddh. In *European Symposium on Research in Computer Security*, pages 243–265. Springer, 2015.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. In *ACNS*, pages 117–136, 2016.

[BDE+21] Maxime Buser, Rafael Dowsley, Muhammed F Esgin, Shabnam Kasra Kermanshahi, Veronika Kuchta, Joseph K Liu, Raphael Phan, and Zhenfei Zhang. Post-quantum verifiable random function from symmetric primitives in pos blockchain. *IACR Cryptol. ePrint Arch.*, 2021:302, 2021.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *International conference on the theory and application of cryptology and information security*, pages 41–69. Springer, 2011.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[BdSG20] Ward Beullens and Cyprien Delpech de Saint Guilhem. Legroast: Efficient post-quantum signatures from the legendre prf. In *International Conference on Post-Quantum Cryptography*, pages 130–150. Springer, 2020.

[BEF18] Dan Boneh, Saba Eskandarian, and Ben Fisch. Post-quantum epid group signatures from symmetric primitives. Technical report, Cryptology ePrint Archive, Report 2018/261, 2018. https://eprint. iacr. org/2018/261, 2018.

[BEF19] Dan Boneh, Saba Eskandarian, and Ben Fisch. Post-quantum epid signatures from symmetric primitives. In *Cryptographers' Track at the RSA Conference*, pages 251–271. Springer, 2019.

[Ber11] Daniel J Bernstein. Post-quantum cryptography. *Encyclopedia of Cryptography and Security*, pages 949–950, 2011.

[BFG+13] David Bernhard, Georg Fuchsbauer, Essam Ghadafi, Nigel P Smart, and Bogdan Warinschi. Anonymous attestation with user-controlled linkability. *International Journal of Information Security*, 12(3):219–249, 2013.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: a new optimized sublinear iop. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2025–2038, 2020.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO*, pages 390–420. Springer, 1992.

[BHH+15] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. Sphincs: practical stateless hash-based signatures. In *EUROCRYPT*, pages 368–397. Springer, 2015.

[BHK+] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. Sphincs+: Stateless hash-based signature. https://sphincs.org/. Accessed: 2020-04-2020.

[BHK+19] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The sphincs+ signature framework. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2129–2146, 2019.

[BHSB19] Michael Backes, Lucjan Hanzlik, and Jonas Schneider-Bensch. Membership privacy for fully dynamic group signatures. In *ACM CCS 2019*, pages 2181–2198, 2019.

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography Conference*, pages 60–79. Springer, 2006.

[BKM09] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. *Journal of Cryptology*, 22(1):114–138, 2009.

[BL09] Ernie Brickell and Jiangtao Li. Enhanced privacy id from bilinear pairing. *IACR Cryptol. ePrint Arch.*, 2009:95, 2009.

[BLS+19] Maxime Buser, Joseph K Liu, Ron Steinfeld, Amin Sakzad, and Shi-Feng Sun. Dgm: A dynamic and revocable group merkle signature. In *European Symposium on Research in Computer Security*, pages 194–214. Springer, 2019.

[BMRS21] Carsten Baum, Alex J Malozemoff, Marc B Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *Annual International Cryptology Conference*, pages 92–122. Springer, 2021.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, pages 614–629. Springer, 2003.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BS04] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 168–177, 2004.

[BSGK+21] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from aes. In *IACR International Conference on Public-Key Cryptography*, pages 266–297. Springer, 2021.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *Cryptographers' Track at the RSA Conference*, pages 136–153. Springer, 2005.

[CBH+18] Konstantinos Chalkias, James Brown, Mike Hearn, Tommy Lillehagen, Igor Nitto, and Thomas Schroeter. Blockchained post-quantum signatures. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1196–1203. IEEE, 2018.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS*, pages 1825–1842. ACM, 2017.

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In *Public-Key Cryptography–PKC 2016*, pages 234–264. Springer, 2016.

[CFS01] Nicolas T Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a mceliece-based digital signature scheme. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 157–174. Springer, 2001.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[CLHY05] Sherman SM Chow, Richard WC Lui, Lucas CK Hui, and Siu-Ming Yiu. Identity based ring signature: Why, how and what next. In *European Public Key Infrastructure Workshop*, pages 144–161. Springer, 2005.

[CM98] Jan Camenisch and Markus Michels. A group signature scheme with improved efficiency. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 160–174. Springer, 1998.

[CP94] Lidong Chen and Torben P Pedersen. New group signature schemes. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 171–181. Springer, 1994.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Annual International Cryptology Conference*, pages 410–424. Springer, 1997.

[CVH91] David Chaum and Eugène Van Heyst. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 257–265. Springer, 1991.

[CYH05] Sherman SM Chow, Siu-Ming Yiu, and Lucas CK Hui. Efficient identity based ring signature. In *International Conference on Applied Cryptography and Network Security*, pages 499–512. Springer, 2005.

[CYH21] Chengtang Cao, Lin You, and Gengran Hu. Fuzzy identity-based ring signature from lattices. *Security and Communication Networks*, 2021, 2021.

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

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *Cryptographers' track at the rsa conference*, pages 127–144. Springer, 2015.

[DL21] Jesus Diaz and Anja Lehmann. Group signatures with user-controlled and sequential linkability. In *Public-Key Cryptography–PKC 2021*, pages 360–388. Springer, 2021.

[DPLS18] Rafaël Del Pino, Vadim Lyubashevsky, and Gregor Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 574–591, 2018.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[DRS18] David Derler, Sebastian Ramacher, and Daniel Slamanig. Post-quantum zero-knowledge proofs for accumulators with applications to ring signatures from symmetric-key primitives. In *International Conference on Post-Quantum Cryptography*, pages 419–440. Springer, 2018.

[dSGDMOS19] Cyprien Delpech de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P Smart. Bbq: using aes in picnic signatures. In *International Conference on Selected Areas in Cryptography*, pages 669–692. Springer, 2019.

[EBM18] Rachid El Bansarkhani and Rafael Misoczki. G-merkle: A hash-based group signature scheme from standard assumptions. In *PQCrypto*, pages 441–463. Springer, 2018.

[ESOa] ESORICS 2019. https://esorics2019.uni.lu/.

[ESOb] ESORICS 2022. https://esorics2022.compute.dtu.dk/.

[EZS+19] Muhammed F Esgin, Raymond K Zhao, Ron Steinfeld, Joseph K Liu, and Dongxi Liu. Matrict: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 567–584, 2019.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1986.

[GGCT06] Chandana Gamage, Ben Gras, Bruno Crispo, and Andrew S Tanenbaum. An identity-based ring signature scheme with enhanced privacy. In *2006 Securecomm and Workshops*, pages 1–5. IEEE, 2006.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[GKV10] S Dov Gordon, Jonathan Katz, and Vinod Vaikuntanathan. A group signature scheme from lattice assumptions. In *ASIACRYPT*, pages 395–412. Springer, 2010.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security*, pages 1069–1083, 2016.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996.

[HBG+18] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: extended merkle signature scheme. In *RFC 8391*. IRTF, 2018.

[HBGM15] Andreas Hülsing, Denis Butin, Stefan Gazdag, and Aziz Mohaisen. Xmss: Extended hash-based signatures. In *Crypto Forum Research Group Internet-Draft.(2015). draft-irtf-cfrg-xmss-hash-based-signatures-01*, 2015.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *ASIACRYPT*, pages 79–102. Springer, 2015.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[HR10] Viet Tung Hoang and Phillip Rogaway. On generalized feistel networks. In *Annual Cryptology Conference*, pages 613–630. Springer, 2010.

[HS04] Javier Herranz and Germán Sáez. New identity-based ring signature schemes. In *International Conference on Information and Communications Security*, pages 27–39. Springer, 2004.

[Hül13] Andreas Hülsing. W-ots+–shorter signatures for hash-based signature schemes. In *International Conference on Cryptology in Africa*, pages 173–188. Springer, 2013.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009.

[Int20] Intel enhanced privacy id (epid) security technology. https://software.intel.com/content/www/us/en/develop/articles/intel-enhanced-privacy-id-epid-security-technology.html, 2020.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 525–537, 2018.

[KZ20] Daniel Kales and Greg Zaverucha. Improving the performance of the picnic signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 154–188, 2020.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

[LLLS13] Fabien Laguillaumie, Adeline Langlois, Benoît Libert, and Damien Stehlé. Lattice-based group signatures with logarithmic signature size. In *ASIACRYPT*, pages 41–61. Springer, 2013.

[LLM+16] Benoît Libert, San Ling, Fabrice Mouhartem, Khoa Nguyen, and Huaxiong Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In *ASIACRYPT*, pages 373–403. Springer, 2016.

[LLNW14] Adeline Langlois, San Ling, Khoa Nguyen, and Huaxiong Wang. Lattice-based group signature scheme with verifier-local revocation. In *PKC*, pages 345–361. Springer, 2014.

[LLNW16] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: logarithmic-size ring signatures and group signatures without trapdoors. In *EUROCRYPT*, pages 1–31. Springer, 2016.

[LNW15] San Ling, Khoa Nguyen, and Huaxiong Wang. Group signatures from lattices: simpler, tighter, shorter, ring-based. In *PKC*, pages 427–449. Springer, 2015.

[LNWX17] San Ling, Khoa Nguyen, Huaxiong Wang, and Yanhong Xu. Lattice-based group signatures: Achieving full dynamicity with ease. In *ACNS*, pages 293–312. Springer, 2017.

[LNWX18] San Ling, Khoa Nguyen, Huaxiong Wang, and Yanhong Xu. Constant-size group signatures from lattices. In *PKC*, pages 58–88. Springer, 2018.

[LW05] Joseph K Liu and Duncan S Wong. Linkable ring signatures: Security models and new schemes. In *International Conference on Computational Science and Its Applications*, pages 614–623. Springer, 2005.

[Lyn20] Ben Lynn. Ggm constructions. https://crypto.stanford.edu/pbc/notes/crypto/ggm.html, 2020.

[McI93] Peter McIlroy. Optimistic sorting and information theoretic complexity. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 467–474. Society for Industrial and Applied Mathematics, 1993.

[Mer89] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[MS17] Giulio Malavolta and Dominique Schröder. Efficient ring signatures in the standard model. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 128–157. Springer, 2017.

[otBCI20] Legion of the Bouncy Castle Inc. The legion of the bouncy castle, 2020.

[Pic] Picnic: Post quantum signatures. https://github.com/microsoft/Picnic. Accessed: 2020-04-20.

[PK01] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity—a proposal for terminology. In *Designing privacy enhancing technologies*, pages 1–9. Springer, 2001.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RST01] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 552–565. Springer, 2001.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*, pages 47–53. Springer, 1984.

[SYL+18] Shi-Feng Sun, Xingliang Yuan, Joseph K Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *ACM CCS 2018*, pages 763–780. ACM, 2018.

[TSS+18] Wilson Abel Alberto Torres, Ron Steinfeld, Amin Sakzad, Joseph K Liu, Veronika Kuchta, Nandita Bhattacharjee, Man Ho Au, and Jacob Cheng. Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice ringct v1. 0). In *Australasian Conference on Information Security and Privacy*, pages 558–576. Springer, 2018.

[TW05] Patrick P Tsang and Victor K Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In *International Conference on Information Security Practice and Experience*, pages 48–60. Springer, 2005.

[Unr15] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *EUROCRYPT*, pages 755–784. Springer, 2015.

[Wan08] Jin Wang. Identity-based ring signature from lattice basis delegation. *Beijing: Tsinghua University*, 2008.

[WDZ+15] Baodian Wei, Yusong Du, Huang Zhang, Fangguo Zhang, Haibo Tian, and Chongzhi Gao. Identity based threshold ring signature from lattices. In *International Conference on Network and System Security*, pages 233–245. Springer, 2015.

[WS11] Jin Wang and Bo Sun. Ring signature schemes from lattice basis delegation. In *International Conference on Information and Communications Security*, pages 15–28. Springer, 2011.

[ZK02] Fangguo Zhang and Kwangjo Kim. Id-based blind signature and ring signature from pairings. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 533–547. Springer, 2002.

[ZT18] Gongming Zhao and Miaomiao Tian. A simpler construction of identity-based ring signatures from lattices. In *International Conference on Provable Security*, pages 277–291. Springer, 2018.