UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Arbeitsgruppe Codes und Kryptographie

Post-Quantum Secure Group Signatures

Master's Thesis

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Laurens Porzenheim

submitted to:

Prof. Dr. Johannes Blömer

and

Jun. Prof. Dr. Sevag Gharibian

Paderborn, February 1, 2019

Declaration

(Translation from German)

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Original Declaration Text in German:

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

City, Date

Signature

Contents

1 Introduction **1**

2 Notation **3**

3 Basic Cryptographic Definitions **5**
3.1 Pseudo-Random Generator 5
3.2 Pseudo-Random Function 6
3.3 Hash Function 6
3.4 Secret Sharing Scheme 7
3.5 Commitment Scheme 7
3.6 Digital Signature Scheme 8
3.7 Group Signature Scheme 9
3.7.1 Fixed Group Size 9
3.7.2 Dynamic Group Size 12
3.8 Σ-Protocol 16
3.9 Non-Interactive Proof System 16
3.10 Unruh's Transform 18
3.11 Multi-Party Computation Protocol 19
3.12 Input Pre-Processing Model 20
3.13 Merkle Tree 21

4 Symmetric-Key Primitive Based Group Signatures **23**
4.1 The Construction of Katz et al. 23
4.1.1 An MPC Protocol 23
4.1.2 A Proof of Knowledge for Any NP-Relation 27
4.1.3 The Group Signature of Katz et al. 32
4.1.4 Efficiency 39
4.2 Conclusion Katz 40
4.3 Extension to a Dynamic Group Signature 40

5 Lattice Based Group Signatures **45**
5.1 Basic Definitions for Lattice-Based Cryptography 45
5.2 The Group Signature of Gordon et al. 48
5.2.1 Building Blocks 48
5.2.2 Construction 50
5.2.3 Techniques 52

Bibliography **55**

1 Introduction

In provable security, *digital signatures* are the equivalent to traditional signatures. They allow a user to digitally sign a document. As in the traditional setting, a verifier then can check the signature for authenticity to confirm the right person signed the document. Furthermore, digital signatures offer integrity, meaning the document cannot be altered after it was signed without the signature becoming invalid.

An extension to digital signatures are so-called *group signatures*. With them, a user that belongs to a group can sign a document on behalf of the group. While group signatures also offer authenticity and integrity, they also hide which specific user created a signature, which is called anonymity. Thus, it can only be verified that someone of the group is the author of the signature. For example, this is useful when a company wants to grant multiple employees the ability to sign a contract, but a third party should not see who exactly is responsible to prevent direct influencing. This example motivates another desired property of group signatures, called opening. While anonymity of group members protects the employees, the board of directors wants to be able to look up authorship of a signature, in case an employee misuses his signing power. Then, we want that the board, i.e. a trusted party, and only the board is able to break anonymity.

There exists a standard technique to construct a group signature scheme, called the sign-encrypt-proof paradigm. That technique uses a digital signature scheme, an encryption scheme and a proof of knowledge. The digital signature scheme is used to provide authenticity. Members of the group are given a certificate of membership, which is a digital signature of an identifier. Then, the encryption scheme is used for traceability A member encrypts his certificate and adds the ciphertext to its group signature. The board of directors, as they are in possession of the decryption key, can then decrypt the ciphertext to reveal who signed a signature. However, in this scenario a user can be dishonest and encrypt something else than his certificate. To prevent this, the proof of knowledge is used. He needs to prove somehow that he encrypted a valid certificate. Therefore, the proof of knowledge is used to identify the signer as registered and to link the encryption scheme and identification, thus also the digital signature, together. The user also needs to incorporate the message in this process somehow, to link the message to the signature.

Although the sign-encrypt-proof paradigm produces a secure group signature scheme, one can consider other techniques. This may have different reasons, for example efficiency. When using the standard technique, group signature sizes or public key sizes may get large. Furthermore, the runtime of the signing algorithm

or the verification algorithm may become impractical. Another consideration is the security model of the group signature scheme. If the application allows a more relaxed model, then other components may be enough to satisfy the requirements.

When deciding the security model, one needs to consider what an adversary is capable of. Due to the recent progress of the capabilities of quantum computers, it is advisable to use security models that allow an adversary capable of quantum operations. For example, Shor [16] showed that a sufficiently large quantum computer is able to solve factorization and the discrete logarithm problem. Therefore, constructions relying on these problems are not secure anymore when faced with a quantum adversary. Thus, it is of great interest to come up with new constructions that are secure even against quantum computers, i.e. *post-quantum secure*.

This thesis presents two construction of post-quantum secure group signature schemes, one from Katz et al. [10] and one from [8]. For both schemes, we present their building blocks and the construction itself. While we only discuss the security of the scheme of Gordon et al., we provide our own security proofs for several building blocks of the construction of Katz et al. as well as for their scheme itself. Furthermore, we discuss what techniques both constructions use and what advantages and disadvantages they entail. We also provide a candidate construction of a group signature scheme of our own. For that, we learn from the techniques of the other constructions. We give only a discussion that explains the intuition.

2 Notation

Before we begin with the content of the paper, we introduce some notation.

By writing $a||b$ we mean the concatenation of two strings a, b. We define $[n] = \{x \in \mathbb{N} : 1 \leq x \leq n\}$ for an $n \in \mathbb{N}$. If S is a finite set, $x \leftarrow S$ denotes drawing a uniformly random x from that set and $|S|$ denotes the size of the set. If A is a probabilistic algorithm with input x, we denote by $y \leftarrow A(x)$ assigning y a value according to the distribution $A(x)$. We say that an algorithm A is ppt, if it is a probabilistic polynomial time algorithm. For a probabilistic algorithm A with input x, $[A(x)]$ denotes the set of all possible outputs of $A(x)$. By $\oplus$ we denote the bit-wise XOR-operator.

We denote vectors as lower-case bold-face letters, e.g $\mathbf{a}$ and matrices as upper-case bold-face letters, e.g. $\mathbf{A}$. By $\mathbf{A}^T$ we denote the transpose of the matrix. $\|\mathbf{a}\|$ denotes a norm of vector $\mathbf{a}$. In this paper, we use the ℓ_2-norm, but other norms can be used as well. The norm $\|\mathbf{A}\|$ of matrix is defined as $\max_i \|\mathbf{a}_i\|$, where the $\mathbf{a}_i$ are the columns of $\mathbf{A}$. By $\tilde{\mathbf{A}}$ we denote the Hermite normal form of matrix $\mathbf{A}$. For an $x \in \mathbb{R}$, $\lfloor x \rceil$ denotes rounding x to the nearest $y \in \mathbb{Z}$.

We denote by $A \leftrightarrow B$ two interactive algorithms A, B that interact with each other.

3 Basic Cryptographic Definitions

We begin with defining formal models for some cryptographic primitives, so that we have a framework for our later constructions. Furthermore, we give security definitions for the primitives. This way, we know when our constructions are considered to be secure.

One of the most basic definitions is that of a *negligible function*. We need such a function to be able to define asymptotic security. In general, we require that the probability that an adversary breaks the security of a primitive is less than a negligible function. Then, we know that increasing the security parameter makes it harder for an adversary to be successful.

Definition 3.1 *A function f is called negligible, if for all $c \in \mathbb{N}$ there exists a $n_0 \in \mathbb{N}$, such that for all $n \geq n_0$ we have that $f(n) < n^{-c}$.*

Another basic definition we need is that of a keyed function. This definition is just a formalism, but we need it to describe a construction.

Definition 3.2 *A keyed function $\mathcal{F}$ is a tuple (Kg, f) of a ppt key generation algorithm $\mathsf{Kg} : 1^\lambda \mapsto k \in \mathcal{K}$ and a polynomial-time computable function $f : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$.*

3.1 Pseudo-Random Generator

With a pseudo-random generator, one is able to expand a short, random string into a longer string. The distribution of the latter string is then computationally indistinguisihable from that of a random string of the same length. This is useful, for example, if one needs to send a long random string, but wants to make the message shorter by only sending the input for the pseudo-random generator.

Definition 3.3 *A pseudo-random generator is a deterministic polynomial time algorithm* RNG *such that*

- *on input $s \in \{0,1\}^*$ outputs some $y \in \{0,1\}^*$ with $|y| = p(|s|)$, for some polynomial $p(\cdot)$,*
- *for every ppt adversary $\mathcal{A}$ there exists a negligible function* negl$(\cdot)$ *such that*

$$\left|\Pr[r \leftarrow \{0,1\}^{p(\lambda)} : \mathcal{A}(r) = 1] - \Pr[s \leftarrow \{0,1\}^\lambda : \mathcal{A}(\mathsf{RNG}(s)) = 1]\right| \leq \mathsf{negl}(\lambda).$$

3.2 Pseudo-Random Function

A primitive related to pseudo-random generators is the pseudo-random function. In contrast to the former, the output of the keyed function is indistinguishable from that of a random function, instead of a uniformly random string.

Definition 3.4 *A pseudo-random function* PRF *for an input space* $\mathcal{X}$ *is a tuple of two algorithms* $\mathsf{PRF} = (\mathsf{Kg}, f)$.

- $\mathsf{Kg}(1^\lambda) \to k$: *the ppt key-generation algorithm takes a security parameter* λ *as input and outputs a key* k.
- $f(k, x) \to y$: *on input key* k *and* $x \in \mathcal{X}$, *the deterministic function outputs some* $y \in \mathcal{Y}$.

Furthermore, we require that for every adversary $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$, *such that the advantage of* $\mathcal{A}$, *defined by*

$$|\Pr[k \leftarrow \mathsf{Kg}(1^\lambda) : \mathcal{A}^{f(k,\cdot)}(1^\lambda) = 1] - \Pr[g \leftarrow \mathcal{F}(\mathcal{X}, \mathcal{Y}) : \mathcal{A}^{g(\cdot)}(1^\lambda) = 1]|,$$

is smaller than $\mathsf{negl}(\lambda)$, *where* $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ *is the set of all functions from* $\mathcal{X}$ *to* $\mathcal{Y}$.

3.3 Hash Function

A hash function is primarily used to map large strings to smaller ones. This property can be useful to improve efficiency of a construction, as messages or outputs can be shorter.

Definition 3.5 *A hash function is a tuple of two algorithms* $\mathcal{H} = (\mathsf{Kg}, H)$.

- $\mathsf{Kg}(1^\lambda) \to k$: *on input* 1^λ *for a security parameter* λ, *the ppt key generation algorithm outputs a key* k.
- $H(k, x) \to y$: *on input key* k *and bit string* x, *the deterministic algorithm outputs a hash value* y.

We may write $H_k(x)$ as shorthand for $H(k, x)$. Since we map a large space into a smaller space, there exist inputs to the hash function that map to the same image. This may influence security of a construction, thus we are interested in so-called collision-resistant hash functions.

Definition 3.6 *Let* λ *be a security parameter. We call a hash function* $\mathcal{H}(\mathsf{Kg}, H)$ *collision-resistant, if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$, *such that*

$$\Pr[k \leftarrow \mathsf{Kg}(1^\lambda), (x_1, x_2) \leftarrow \mathcal{A}(1^\lambda, k) : H_k(x_1) = H_k(x_2)] \leq \mathsf{negl}(\lambda).$$

3.4 Secret Sharing Scheme

As the name suggests, a secret sharing scheme is useful when one wants to share a secret value among a group. Then, each member gets a so-called secret share. If all members come together, the secret shares can be used to reconstruct the original secret. However, without all of the secret shares, it is not possible to learn anything about the original secret.

Definition 3.7 *An n-out-of-n secret sharing scheme* Π *for a finite secret space* K *is a tuple* $(\mathsf{Share}, \mathsf{Recon})$ *of ppt algorithms.*

- $\mathsf{Share}(s) \to (s_i)_{i\in[n]}$*: on input* $s \in K$ *the algorithm outputs a vector* $(s_i)_{i\in[n]}$ *of secret shares.*
- $\mathsf{Recon}((s_i)_{i\in[n]}) \to s$*: when given a vector* $(s_i)_{i\in[n]}$ *of secret shares, the algorithm outputs some secret* s*.*

We call such a secret sharing scheme correct, if for all $s \in K$ *we have that* $\Pr[\mathsf{Recon}(\mathsf{Share}(s)) = s] = 1$*.*

Definition 3.8 *We call an n-out-of-n secret sharing scheme* $\Pi = (\mathsf{Share}, \mathsf{Recon})$ *for secret space* K *perfectly private, if for all* $s, s' \in K$*, all* $T \subseteq [n]$ *with* $1 \leq |T| < n$*, all possible vectors of shares* $(s_i)_{i\in T}$*, we have that*

$$\Pr[(\mathsf{Share}(s))_{i\in T} = (s_i)_{i\in T}] = \Pr[(\mathsf{Share}(s'))_{i\in T} = (s_i)_{i\in T}].$$

A simple secret sharing scheme, that we use later, is the XOR-secret sharing scheme.

Construction 3.9 *The n-out-of-n XOR-secret sharing scheme for* $K = \{0,1\}$ *consists of the following algorithms:*

- $\mathsf{Share}(s)$*: Choose* $(s_i)_{i\in[n-1]} \in \{0,1\}^{n-1}$ *uniformly at random. Compute* $s_n = s \oplus \bigoplus_{i=1}^{n-1} s_i$*. Output* $(s_i)_{i\in[n]}$*.*
- $\mathsf{Recon}((s_i)_{i\in[n]})$*: Output* $\bigoplus_{i=1}^{n} s_i$*.*

The scheme is correct because $s = \bigoplus_{i=1}^{n} s_i$ by definition. Furthermore, by definition for every set $T \subseteq [n]$ with $|T| \leq n-1$, we know that for an adversary seeing $(s_j)_{j\in T}$ all s_j look uniformly at random, due to the properties of XOR.

Lemma 3.10 *The n-out-of-n XOR-secret sharing scheme is correct and perfectly private.*

3.5 Commitment Scheme

A commitment scheme allows a user to commit to a value. The commitment then hides the original value, but it can be opened to the original value by using an additional information, called opening value. An example for this process is an

auction, where bidders submit their bid by committing to it, so it cannot be seen what others do. After that, every bid is opened and the highest one wins.

Definition 3.11 *A commitment scheme consists of three ppt algorithms* $(\mathsf{Kg}, \mathsf{Com}, \mathsf{Open})$ *with the following properties:*

- $\mathsf{Kg}(1^\lambda) \to \mathsf{pk}$*: the key generation takes as input* 1^λ *where* λ *is some security parameter. Then, it outputs a public key* pk*. This implicitly defines a message space* $\mathcal{M}$*.*
- $\mathsf{Com}(\mathsf{pk}, m) \to (\mathsf{com}, d)$*: on input public key* pk *and message* m*, the commitment algorithm outputs a commitment* com *and an opening value* d*.*
- $\mathsf{Open}(\mathsf{pk}, \mathsf{com}, d) \to m$*: when given public key* pk*, commitment* com *and opening value* d*, the opening algorithm outputs either a message* m *or the error symbol* $\perp$*.*

We call a commitment scheme correct, if for all security parameters λ*, all* $\mathsf{pk} \in [\mathsf{Kg}(1^\lambda)]$*, all* $m \in \mathcal{M}$ *we have that* $\Pr[\mathsf{Open}(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, m)) = m] = 1$*.*

Definition 3.12 *We call a commitment scheme* $\Pi = (\mathsf{Kg}, \mathsf{Com}, \mathsf{Open})$ *secure, if it satisfies the following conditions:*

Perfectly Hiding *For every security parameter* λ*, all* $\mathsf{pk} \in [\mathsf{Kg}(1^\lambda)]$*, all* $m, m' \in \mathcal{M}$*, we have that the distributions of* $(\mathsf{com}, \cdot) \leftarrow \mathsf{Com}(\mathsf{pk}, m)$ *and* $(\mathsf{com}, \cdot) \leftarrow \mathsf{Com}(\mathsf{pk}, m')$ *are the same.*

Computationally Binding *For all ppt adversaries we have that the probability that they output a commitment and two different opening values, such that the commitment is opened to two different messages, is negligible. Formally, let* λ *be a security parameter. Then, for every ppt adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\begin{aligned}\Pr[\mathsf{pk} \leftarrow \mathsf{Kg}(1^\lambda), (\mathsf{com}, d_1, d_2) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}) : d_1 \neq d_2 \wedge \\ \mathsf{Open}(\mathsf{pk}, \mathsf{com}, d_1) \neq \mathsf{Open}(\mathsf{pk}, \mathsf{com}, d_2) \wedge \\ \mathsf{Open}(\mathsf{pk}, \mathsf{com}, d_1), \mathsf{Open}(\mathsf{pk}, \mathsf{com}, d_2) \neq \perp] \leq \mathsf{negl}(\lambda).\end{aligned}$$

3.6 Digital Signature Scheme

A digital signature scheme is the equivalent of a traditional signature, where one can sign documents or messages, which can then be checked for validity. However, in the digital signature scheme, a signer gets a secret signing key, with which he can sign something. Then, everyone can check for validity by using a public key.

Definition 3.13 *A signature scheme consists of three ppt algorithms* $(\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$ *with the following properties:*

$\mathsf{Exp}^{\text{euf-cma}}_{\Pi,\mathcal{A}}(\lambda)$

1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\lambda)$
2 : $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$
3 : If $\mathsf{Vrfy}(\mathsf{pk}, m^*, \sigma^*) = 1$ and m^* was never queried for, output 1, else 0.

Figure 3.1: The unforgeability game for a digital signature.

- $\mathsf{Kg}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$*: the key generation outputs a public key* pk *and a secret key* sk *when given* 1^λ*, where* λ *is a security parameter. The public key implicitly defines a message space* $\mathcal{M}$*.*

- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$*: the signing algorithm on input secret key* sk *and message* m *outputs a signature* σ*.*

- $\mathsf{Vrfy}(\mathsf{pk}, m, \sigma) \to b$*: on input public key* pk*, message* m *and signature* σ*, the verifying algorithm outputs a bit* b*, where 1 denotes the signature is accepted.*

A signature scheme is correct, if for all security parameters λ*, all* $(\mathsf{pk}, \mathsf{sk}) \in [\mathsf{Kg}(1^\lambda)]$*, all* $m \in \mathcal{M}$*, we have that* $\Pr[\mathsf{Vrfy}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1$.

Definition 3.14 *Let* λ *be a security parameter. We call a signature scheme* $\Pi = (\mathsf{Kg}, \mathsf{SignVrfy})$ *existentially unforgeable under chosen message attack (EUF-CMA), if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathsf{Exp}^{euf\text{-}cma}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

3.7 Group Signature Scheme

Group signatures are an extension of digital signatures. Instead of a single signer, there exists a group of signers. As before, we do not want that anyone except the approved signers, i.e. the group members, are able to sign messages. Furthermore, we want that group members stay anonymous in their group. This means that one cannot distinguish which group member signed a message. However, there exists a group manager that is able to trace people, i.e. see who signed a signature.

3.7.1 Fixed Group Size

There exist different formal models for group signatures. One distinction is whether a group signature scheme has a fixed size for groups or a dynamic one. In the former case, before the group signature scheme is set up, it is known how many group members exist, denoted by $\ell \in \mathbb{N}$.

$\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)$

1: Initialize empty sets $\mathbb{M},\mathbb{C}$.
2: $(\mathsf{gpk},\mathsf{msk},(\mathsf{usk}_i)_{i\in[\ell]}) \leftarrow \mathsf{Kg}(1^\lambda,1^\ell)$
3: $\mathcal{A}$ is given $\mathsf{gpk},\mathsf{msk}$ and gets access to the following oracles:
- $\mathsf{S}(i,m)$: Return $\mathsf{Sign}(\mathsf{usk}_i,m)$ to the adversary. Add (m,i) to $\mathbb{M}$.
- $\mathsf{Corrupt}(i)$: Return usk_i to the adversary. We say user i is corrupted. Add i to $\mathbb{C}$.

4: $\mathcal{A}$ outputs m^*,σ^*.
5: If $\mathsf{Vrfy}(\mathsf{gpk},m^*,\sigma^*)=1$ and $\mathsf{Open}(\mathsf{msk},m^*,\sigma^*)=:i\notin\mathbb{C}$ and $(m^*,i)\notin\mathbb{M}$, the experiment outputs 1 and we say that $\mathcal{A}$ wins. Else, he loses.

Figure 3.2: The traceability game for a static group signature scheme.

Definition 3.15 *A fixed size group signature* Π *consists of four ppt algorithms:*

- $\mathsf{Kg}(1^\lambda,1^\ell) \to (\mathsf{gpk},\mathsf{msk},(\mathsf{usk}_i)_{i\in[\ell]})$*: the key-generation algorithm takes as input a security parameter* 1^λ *and a length parameter* 1^ℓ *and outputs a group public key* gpk*, master secret key* msk*, and user signing keys* usk_i *for each* $i\in[\ell]$*. The group public key implicitly defines a message space* $\mathcal{M}$*.*
- $\mathsf{Sign}(\mathsf{usk}_i,m)\to\sigma$*: the signing algorithm gets a user secret key* usk_i *and a message* m*, and then outputs a signature* σ*.*
- $\mathsf{Vrfy}(\mathsf{gpk},m,\sigma)\to b$*: with the group public key* gpk*, a signature* σ *and a message* m*, the verifying algorithm outputs a bit* b*, where 1 means the verification accepts.*
- $\mathsf{Open}(\mathsf{msk},m,\sigma)\to i$*: when given the master secret key* msk*, a message* m *and a signature* σ*, the opening algorithm outputs an index* $i\in[\ell]$ *or an error symbol* $\perp$*.*

A fixed size group signature is correct, if there exists a negligible function $\mathsf{negl}(\cdot)$ *such that for all security parameters* λ*, all length parameters* $\ell\in\mathbb{N}$*, all* $(\mathsf{gpk},\mathsf{msk},(\mathsf{usk}_i)_{i\in[\ell]})\in[\mathsf{Kg}(1^\lambda,1^\ell)]$*, all* $m\in\mathcal{M}$ *and all* $i\in[\ell]$ *we have that*

$$\Pr[\mathsf{Vrfy}(\mathsf{gpk},m,\mathsf{Sign}(\mathsf{usk}_i,m))=1]=1 \text{ and}$$

$$\Pr[\mathsf{Open}(\mathsf{msk},m,\mathsf{Sign}(\mathsf{usk}_i,m))=i]\geq 1-\mathsf{negl}(\lambda).$$

To define security for a group signature, we define two games, one for traceability and one for anonymity.

Definition 3.16 *A fixed size group signature* $\Pi=(\mathsf{Kg},\mathsf{Sign},\mathsf{Vrfy},\mathsf{Open})$ *is traceable, if for all ppt adversaries* $\mathcal{A}$ *and all polynomials* $\ell(\cdot)$*, there exists a negligible*

$\mathsf{Exp}^{anon}_{\Pi,\mathcal{A}}(\lambda,\ell)$

1: Initialize empty set $\mathbb{C}$.
2: $(\mathsf{gpk},\mathsf{msk},(\mathsf{usk}_i)_{i\in[\ell]}) \leftarrow \mathsf{Kg}(1^\lambda,1^\ell)$
3: $\mathcal{A}$ is given gpk and gets access to the following oracles:
- $\mathsf{S}(i,m)$: Return $\mathsf{Sign}(\mathsf{usk}_i,m)$ to the adversary.
- $\mathsf{Corrupt}(i)$: Return usk_i to the adversary. We say user i is corrupted. Add i to $\mathbb{C}$.
- $\mathsf{Trace}(m,\sigma)$: Return $\mathsf{Open}(\mathsf{msk},m,\sigma)$.

4: $\mathcal{A}$ outputs a message m^* and some $i_0,i_1 \in [\ell]$.
5: $b \leftarrow \{0,1\}$, $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{usk}_{i_b},m^*)$
6: $\mathcal{A}$ gets σ^* and outputs b'. He may query his oracles, except for $\mathsf{Trace}(m^*,\sigma^*)$.
7: If $b=b'$ and $i_0,i_1 \notin \mathbb{C}$, the experiment outputs 1 and we say that $\mathcal{A}$ wins.

Figure 3.3: The anonymity game for a static group signature scheme.

function $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell(\lambda))=1] \leq \mathsf{negl}(\lambda).$$

Definition 3.17 *A fixed size group signature* Π *offers weak anonymity, if for all ppt adversaries* $\mathcal{A}$ *and all polynomials* $\ell(\cdot)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that*

$$\Pr[\mathsf{Exp}^{anon}_{\Pi,\mathcal{A}}(\lambda,\ell(\lambda))=1] \leq \frac{1}{2}+\mathsf{negl}(\lambda).$$

There exists an alternate anonymity definition, where the adversary gets no tracing oracle, but gets the keys of the users. This implies that users cannot even distinguish signatures signed by them from others. Due to the lack of the tracing oracle, we assume that the group manager is honest.

$\mathsf{Exp}^{anon-cpa}_{\Pi,\mathcal{A}}(\lambda,\ell)$

1 : $(\mathsf{gpk},\mathsf{msk},(\mathsf{usk}_i)_{i\in[\ell]}) \leftarrow \mathsf{Kg}(1^\lambda,1^\ell)$
2 : $(m^*,i_0,i_1 \in [\ell]) \leftarrow \mathcal{A}(\mathsf{gpk},(\mathsf{usk}_i)_{i\in[\ell]})$
3 : $b \leftarrow \{0,1\}$, $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{usk}_{i_b},m^*)$
4 : $\mathcal{A}$ gets σ^* and outputs b'. If $b=b'$ the experiment outputs 1, else 0.

Figure 3.4: An game for an anlternate definition of anonymity of a static group signature scheme.

Definition 3.18 *A fixed size group signature* Π *offers CPA-anonymity, if for all ppt adversaries* $\mathcal{A}$ *and all polynomials* $\ell(\cdot)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that*

$$\Pr[\mathsf{Exp}^{anon-cpa}_{\Pi,\mathcal{A}}(\lambda, \ell(\lambda)) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

3.7.2 Dynamic Group Size

In the dynamic case, the scheme is set up with an empty group and the role of the group manager is split into the role of the issuer and the opener. Then, users can join by registering with the issuer, who updates a registry with registration information. Additionally, there exists a judge algorithm. This way, we can allow the opener to be potentially corrupt. However, he also has to output a proof that he correctly opened a signature. This proof is then verified by the judge algorithm.

For the definitions of the formal model and security, we follow the work of [3].

Definition 3.19 *A dynamic group signature scheme* Π *consists of five ppt algorithms* $\mathsf{Setup}, \mathsf{UKg}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Open}$ *and a protocol* $\mathsf{Join} \leftrightarrow \mathsf{TJoin}$.

- $\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{gpk}, \mathsf{isk}, \mathsf{osk})$*: The setup algorithm takes as input a security parameter and outputs a group public key* gpk*, an issuing key* isk *and an opening key* osk*. It also sets up a registry* reg*.*

- $\mathsf{UKg}(1^\lambda, \mathsf{gpk}) \rightarrow (\mathsf{usk}_i, \mathsf{upk}_i)$*: The user key generation algorithm takes as input a security parameter and outputs a user secret key* usk_i *and a user public key* upk_i*.*

- $\mathsf{Join}(\mathsf{usk}_i) \leftrightarrow \mathsf{Issue}(\mathsf{upk}_i, \mathsf{isk})$*: When* Join *is given a user secret key* usk_i*, while* Issue *gets the user public key* upk_i *and the issuing secret key* isk*, the algorithms interact with each other. At the end,* Join *outputs a certificate* cert_i *and* Issue*, which has write access to* reg*, updates* reg_i*.*

- $\mathsf{Sign}(\mathsf{usk}_i, \mathsf{cert}_i, m) \rightarrow \sigma$*: The signing algorithm gets as input a user secret key* usk_i*, a certificate* cert_i *and a message* m*, and then outputs a signature* σ*.*

- $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma) \rightarrow b$*: With the group public key* gpk *a signature* σ *and a message* m*, the verifying algorithm outputs a bit* b*, where 1 means the verification accepts.*

- $\mathsf{Open}(\mathsf{osk}, m, \sigma) \rightarrow (i, \pi)$*: When given the tracing key* osk*, a message* m *and a signature* σ*, the opening algorithm, which has read access to* reg*, outputs an index* i *together with a non-interactive proof* π *or an error symbol* $\perp$*.*

- $\mathsf{Judge}(\mathsf{gpk}, i, \mathsf{upk}_i, m, \sigma, \pi) \rightarrow b$*: When given the group public key* gpk*, an index* i *with the associated user public key* upk_i*, a message* m*, a signature* σ *and a proof* π*, the judge algorithm outputs a bit* b*.*

$\mathsf{Exp}^{d-corr}_{\Pi,\mathcal{A}}(\lambda)$

1: Create empty set $\mathbb{U}$.
2: $(\mathsf{gpk}, \mathsf{jsk}, \mathsf{tsk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3: Run $\mathcal{A}(1^\lambda)$, while the adversary gets read access to reg and access to the following oracle:
   - $\mathsf{Add}(i)$: If $i \in \mathbb{U}$, the oracle returns $\perp$. Else, add i to $\mathbb{U}$ and run $(\mathsf{usk}_i, \mathsf{upk}_i) \leftarrow \mathsf{UKg}(1^\lambda, \mathsf{gpk})$. Then, run $\mathsf{Join}(1^\lambda, \mathsf{usk}_i) \leftrightarrow \mathsf{Issue}(1^\lambda, \mathsf{upk}_i, \mathsf{isk})$ to get cert_i and an updated reg. The oracle then returns upk_i.
4: $\mathcal{A}$ outputs (j, m).
5: If $j \notin \mathbb{U}$ or $\mathsf{cert}_i = \perp$, output 0.
6: $\sigma \leftarrow \mathsf{Sig}(\mathsf{usk}_j, \mathsf{cert}_j, m)$
7: If $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma) = 0$ output 1.
8: $(k, \pi) \leftarrow \mathsf{Open}(\mathsf{osk}, m, \sigma)$
9: If $j \neq k$ or $\mathsf{Judge}(\mathsf{gpk}, j, \mathsf{upk}_j, m, \sigma, \pi) = 0$ output 1, else 0.

Figure 3.5: The correctness game for a dynamic group signature scheme.

In contrast to the static model, we use a game to define correctness of a dynamic group signature scheme. This is due to the different numbers and orders of users that can join.

While we use the security game from [3], we relax their security requirements. Instead of unbounded adversaries, we only require correctness to hold for ppt adversaries. Furthermore, we allow a negligible error.

Definition 3.20 *We say that a dynamic group signature scheme* Π *is correct, if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that*

$$\Pr[\mathsf{Exp}^{d-corr}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

Since the dynamic model differs from the static model, we have to split the the traceability property of the static model into two new properties. While one of them is also called traceability, this requires only that an adversary is not able to create a signature that opens to nobody. The other property, called non-frameability, provides us with the guarantee that a signature created by group of corrupted users does not open to an honest user. While we allow for the latter game that the issuer may be corrupted, we assume that the issuer is honest for the traceability game.

Definition 3.21 *We say that a dynamic group signature* Π *is traceable, if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathsf{Exp}^{d-trace}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

$\mathsf{Exp}^{d-trace}_{\Pi,\mathcal{A}}(\lambda)$

1: Initialize empty sets $\mathbb{U}, \mathbb{C}$.
2: $(\mathsf{gpk}, \mathsf{isk}, \mathsf{osk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3: $\mathcal{A}$ is given gpk and osk, gets read access to reg and access to the following oracles:
- $\mathsf{AddU}(i)$: if $i \in \mathbb{U} \cup \mathbb{C}$ return $\perp$. Else, compute $(\mathsf{usk}_i, \mathsf{upk}_i) \leftarrow \mathsf{UKg}(1^\lambda, \mathsf{gpk})$. Then, add i to $\mathbb{U}$ and give upk_i to $\mathcal{A}$.
- $\mathsf{Corrupt}(i, \mathsf{upk})$: if $i \in \mathbb{U}$, give $\mathsf{usk}_i, \mathsf{cert}_i$ to $\mathcal{A}$, remove i from $\mathbb{U}$ and add i to $\mathbb{C}$. If $\mathsf{upk} \neq \perp$ and $i \in \mathbb{U} \cup \mathbb{C}$ set $\mathsf{upk}_i = \mathsf{upk}$.
- $\mathsf{J}(i)$: if $i \in \mathbb{U}$, then execute $\mathsf{Join}(\mathsf{usk}_i) \leftrightarrow \mathsf{Issue}(\mathsf{upk}_i, \mathsf{isk})$. If $i \in \mathbb{C}$, then $\mathcal{A}$ interacts with $\mathsf{Issue}(\mathsf{upk}_i, \mathsf{isk})$.

4: $\mathcal{A}$ outputs m^*, σ^*.
5: If $\mathsf{Vrfy}(\mathsf{gpk}, m^*, \sigma^*) = 0$, output 0.
6: $(j, \pi) \leftarrow \mathsf{Open}(\mathsf{osk}, m^*, \sigma^*) = \perp$
7: If $j = 0$ or $\mathsf{Judge}(\mathsf{gpk}, j, \mathsf{upk}_j, m, \sigma, \pi) = 0$ the experiment outputs 1, else 0.

Figure 3.6: The traceability game for a dynamic group signature scheme.

$\mathsf{Exp}^{nf}_{\Pi,\mathcal{A}}(\lambda)$

1: Initialize empty sets $\mathbb{U}, \mathbb{C}, \mathbb{M}$.
2: $(\mathsf{gpk}, \mathsf{isk}, \mathsf{osk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3: $\mathcal{A}$ is given gpk, isk and osk, gets write access to reg and access to the following oracles:
- $\mathsf{AddU}(i)$: if $i \in \mathbb{U} \cup \mathbb{C}$ return $\perp$. Else, compute $(\mathsf{usk}_i, \mathsf{upk}_i) \leftarrow \mathsf{UKg}(1^\lambda, \mathsf{gpk})$. Then, add i to $\mathbb{U}$ and give upk_i to $\mathcal{A}$.
- $\mathsf{Corrupt}(i, \mathsf{upk})$: if $i \in \mathbb{U}$, give $\mathsf{usk}_i, \mathsf{cert}_i$ to $\mathcal{A}$, remove i from $\mathbb{U}$ and add i to $\mathbb{C}$. If $\mathsf{upk} \neq \perp$ and $i \in \mathbb{U} \cup \mathbb{C}$ set $\mathsf{upk}_i = \mathsf{upk}$.
- $\mathsf{J}(i)$: if $i \in \mathbb{U}$, then $\mathsf{Join}(1^\lambda, \mathsf{usk}_i)$ interacts with $\mathcal{A}$.
- $\mathsf{S}(i, m)$: if $i \in \mathbb{U}$ and $\mathsf{cert}_i \neq \perp$, return $\mathsf{Sign}(\mathsf{usk}_i, \mathsf{cert}_i, m)$ and add (i, m) to $\mathbb{M}$. Else, return $\perp$.

4: $\mathcal{A}$ outputs m^*, σ^*, j, π.
5: If $\mathsf{Vrfy}(\mathsf{gpk}, m^*, \sigma^*) = 1$ and $\mathsf{Judge}(\mathsf{gpk}, j, \mathsf{upk}_j, m, \sigma, \pi) = 1$ and $j \in \mathbb{U}$ and $\mathsf{cert}_j \neq \perp$ and $(j, m^*) \notin \mathbb{M}$, the experiment outputs 1, else 0.

Figure 3.7: The non-frameability game for a dynamic group signature scheme.

$\mathsf{Exp}^{d-anon}_{\Pi,\mathcal{A}}(\lambda)$

1: Initialize empty sets $\mathbb{U}, \mathbb{C}$.
2: $(\mathsf{gpk}, \mathsf{isk}, \mathsf{osk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3: $\mathcal{A}$ is given gpk and isk, gets write access to reg and access to the following oracles:
- $\mathsf{AddU}(i)$: if $i \in \mathbb{U} \cup \mathbb{C}$ return $\perp$. Else, compute $(\mathsf{usk}_i, \mathsf{upk}_i) \leftarrow \mathsf{UKg}(1^\lambda, \mathsf{gpk})$. Then, add i to $\mathbb{U}$ and give upk_i to $\mathcal{A}$.
- $\mathsf{Corrupt}(i, \mathsf{upk})$: if $i \in \mathbb{U}$, give $\mathsf{usk}_i, \mathsf{cert}_i$ to $\mathcal{A}$, remove i from $\mathbb{U}$ and add i to $\mathbb{C}$. If $\mathsf{upk} \neq \perp$ and $i \in \mathbb{U} \cup \mathbb{C}$ set $\mathsf{upk}_i = \mathsf{upk}$.
- $\mathsf{J}(i)$: if $i \in \mathbb{U}$, then execute $\mathsf{Join}(\mathsf{usk}_i) \leftrightarrow \mathsf{Issue}(\mathsf{upk_i}, \mathsf{isk})$. If $i \in \mathbb{C}$, then $\mathcal{A}$ interacts with $\mathsf{Issue}(\mathsf{upk_i}, \mathsf{isk})$. Else, return $\perp$.
- $\mathsf{S}(i, m)$: if $i \in \mathbb{U}$, return $\mathsf{Sign}(\mathsf{usk}_i, \mathsf{cert}_i, m)$. Else, return $\perp$.
- $\mathsf{Trace}(m, \sigma)$: Return $\mathsf{Open}(\mathsf{osk}, m, \sigma)$.

4: $\mathcal{A}$ outputs a message m^* and some $i_0, i_1 \in \mathbb{U}$, such that $\mathsf{cert}_{i_0}, \mathsf{cert}_{i_1} \neq \perp$.
5: $b \leftarrow \{0,1\}^*$, $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{usk}_{i_b}, \mathsf{cert}_{i_b}, m^*)$.
6: $\mathcal{A}$ gets σ^* and outputs b'. During this, he may query his oracles, except for $\mathsf{Trace}(m^*, \sigma^*)$.
7: If $b = b'$, the experiment outputs 1, else 0.

Figure 3.8: The anonymity game for a dynamic group signature scheme.

Definition 3.22 *We say that a dynamic group signature* Π *offers non-frameability, if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathsf{Exp}^{nf}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

As in the static model, anonymity is a desired property. We adapt the definition of [3] to resemble anonymity in the static model. However, this means we weaken the security model in two places. First, we do not give the user secret keys to the adversary. This means we do not guarantee selfless anonymity, where even a himself user cannot distinguish his own signatures from that of others. Second, we assume that the issuer is honest. Therefore, in our model the adversary does not get the issuer secret key and Issue is executed honestly.

Definition 3.23 *We say that a dynamic group signature* Π *is anonymous, if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that*

$$\Pr[\mathsf{Exp}^{d-anon}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

3.8 Σ-Protocol

A Σ-protocol allows a prover to convince a verifier that the former knows some secret value relative to some statement. We use NP-relations to formalize the relations between statements and secret values.

Definition 3.24 *A binary relation R is called an NP relation, if there exists a deterministic polynomial time algorithm that decides $R(\cdot,\cdot)$ and if for each $(x,w) \in R$ we have that $|w|$ is polynomial in $|x|$.*

We call w a witness for x, if $(x,w) \in R$.

Definition 3.25 *An interactive argument for an NP relation R consists of two interactive ppt turing machines $\mathcal{P},\mathcal{V}$. We call $\mathcal{P}$ the prover and $\mathcal{V}$ the verifier. Then, for any $(x,w) \in R$ we have that when $\mathcal{P}$ is given (x,w) and interacts with $\mathcal{V}$ on input x, then $\mathcal{V}$ outputs 1 with probability greater than $\frac{2}{3}$ at the end of their interaction. Furthermore, for any $x \notin L_R$ and any interactive algorithm $\mathcal{P}^*$ on input x that interacts with $\mathcal{V}$ on input x, we have that the verifier accepts with probability less than $\frac{1}{3}$.*

A Σ-protocol is then defined in the following way.

Definition 3.26 *A Σ-protocol $\Pi = (\mathcal{P},\mathcal{V})$ for an NP-relation R is a three-way interactive argument with the following properties:*

Correctness: *for every $(x,w) \in R$, we have that $\Pr[\mathcal{P}(x,w) \leftrightarrow \mathcal{V}(x) = 1] = 1$.*

Special Soundness: *there exists an extractor E that when given $x \in L_R$ and two accepting transcripts $(a,c,r),(a',c',r')$ of Π, such that $a = a'$ and $c \neq c'$, outputs some w such that $(x,w) \in R$ with overwhelming probability over some security parameter.*

Honest Verifier Zero-Knowledge: *there exists a simulator $\mathcal{S}$ that on input x outputs transcripts, such that for all $(x,w) \in R$ we have that the distributions of transcripts of $\mathcal{P}(x,w) \leftrightarrow \mathcal{V}(x)$ and $\mathcal{S}(x)$ are the same.*

3.9 Non-Interactive Proof System

While Σ-protocols themselves are difficult use in practice, as they assume that the verifier is honest, they can serve as a building block for non-interactive proof systems. Such a proof system also allows a prover to convince a verifier that he knows a witness for a statement, albeit with better security guarantees, namely a dishonest verifier. Furthermore, a non-interactive proof system does not require interaction between the prover and verifier. Instead, the prover prepares a proof, which a verifier can check. We take our formal definitions from [4].

Definition 3.27 *A non-interactive proof system for an NP-relation R consists of two ppt algorithms $(\mathcal{P},\mathcal{V})$, such that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $(x,w) \in R$ we have that $\Pr[\mathcal{V}(x,\mathcal{P}(x,w)) = 1] \geq 1 - \mathsf{negl}(\lambda)$.*

Furthermore, we assume that there exists a well defined set $\mathcal{L}$ that is decidable in polynomial time and for which $L_R \subseteq \mathcal{L}$ holds.

Definition 3.28 *A non-interactive proof system $(\mathcal{P}, \mathcal{V})$ for an NP-relation R is called* zero-knowledge *if there exists a simulator $\mathcal{S}$ if for all ppt adversaries, that have access to an oracle $\mathcal{O}$ and a proof oracle $\mathcal{O}_P$, the following settings are computationally indistinguishable.*

- *Queries to $\mathcal{O}$ are answered by a random oracle. If the adversary queries $\mathcal{O}_P(x, w)$, the response is $\mathcal{P}(x, w)$ if $(x, w) \in R$, else $\perp$.*
- *If $\mathcal{A}$ queries $\mathcal{O}(x)$, the simulator $\mathcal{S}$ gets x and answers with some value. On a query $\mathcal{O}_P(x, w)$, if $(x, w) \in R$, the simulator gets x and answers with some proof π. If $(x, w) \notin R$, the response is $\perp$.*

$\mathsf{Exp}^{SSE}_{\Pi,\mathcal{S},\mathcal{A},\mathcal{K}}(\lambda)$

1: Choose randomness r for $\mathcal{A}$.
2: Run $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}.\mathcal{O}(\cdot),\mathcal{S}.\mathcal{O}_P(\cdot)}(r)$. Let Q be the list of query/answer pairs that $\mathcal{A}$ makes.
3: If $\mathcal{V}(x, \pi) = 0$ or $(x, \pi) \in Q$, the experiment outputs 1 and we say that $\mathcal{K}$ wins.
4: Run $w \leftarrow \mathcal{K}(Q, x, \pi)$. During this, $\mathcal{K}$ may query invoke. On such a query, the challenger runs $\mathcal{A}(r)$ again, but $\mathcal{K}$ may respond to the oracle queries of $\mathcal{A}$.
5: If $(x, w) \in R$, then the experiment outputs 1 and we say that $\mathcal{K}$ wins. Else, the experiment outputs 0 and $\mathcal{K}$ loses.

Figure 3.9: The simulation-sound extractability game for a non-interactive proof system.

Definition 3.29 *A non-interactive zero-knowledge proof system $\Pi = (\mathcal{P}, \mathcal{V})$ for an NP-relation R is called* simulation-sound extractable*, if there exists a simulator $\mathcal{S}$, that fulfills the zero-knowledge property of Π and when queried $\mathcal{O}_P(x')$ for $x' \in \mathcal{L}$ the simulator returns π such that $\mathcal{V}(x', \pi) = 1$ with overwhelming probability, and there exists an extractor $\mathcal{K}$ such that for every prover $\mathcal{A}$ we have that $\Pr[\mathsf{Exp}^{SSE}_{\Pi,\mathcal{S},\mathcal{A},\mathcal{K}}(\lambda)] = 1]$ is non-negligible.*

We may refer to a non-interactive zero-knowledge simulation-sound extractable proof system as a NIZKPoK.

As mentioned before, Σ-protocols can serve as a building block for a non-interactive proof system. The idea is to compute the announcement of the Σ-protocol and to hash it together with the statement, which results in a challenge. The announcement and challenge are then used to compute the response of the Σ-protocol. This way of constructing a non-interactive proof system from a Σ-protocol is called *strong Fiat-Shamir transform.*

Construction 3.30 *Let $\Sigma = (\mathcal{P}, \mathcal{V})$ be some Σ-protocol. Let $\mathcal{H} = (\mathsf{Kg}, H)$ be a hash function. Construct a non-interactive proof system $(\mathcal{P}', \mathcal{V}')$ in the following way. $\mathcal{P}'$ computes an announcement a by using $\mathcal{P}(x, w)$. It then computes its own challenge $c = H(x, a)$ and uses $\mathcal{P}(x, a, c)$ to compute a response r. $\mathcal{P}'$ then outputs (a, r). The verifier $\mathcal{V}'$ computes $c = H(x, a)$ and then uses $\mathcal{V}(x, a, c, r)$ to verify the proof. Denote $\mathsf{sFS}(\Sigma) = (\mathcal{P}', \mathcal{V}')$.*

Theorem 3.31 ([4]) *Let Σ be a Σ-protocol with a challenge space exponentially large in a security parameter λ. If Σ has correctness, special soundness and special honest-verifier zero-knowledge, and we model H as a random oracle, then $\mathsf{sFS}(\Sigma)$ is a non-interactive proof system that is zero-knowledge and simulation-sound extractable with respect to expected polynomial-time adversaries.*

3.10 Unruh's Transform

Although the Fiat-Shamir transform allows us to get a non-interactive zero-knowledge simulation-sound extractable proof system from a Σ-protocol, there is a caveat. The theorem only guarantees security against classical adversaries, i.e. non-quantum. Furthermore, there is a result [2] showing that relative to certain oracles the Fiat-Shamir transform produces insecure non-interactive proofs, although the underlying Σ-protocol is secure, if the adversary is able to query the hash function in superposition. This indicates that the Fiat-Shamir transform may not be secure in general against a quantum adversary.

To alleviate this problem, Unruh [17] created another transform to get non-interactive proofs from a Σ-protocol, called Unruh's transform. He proved his construction to be secure even against quantum adversaries.

Theorem 3.32 *For every Σ-protocol Σ, there exists a non-interactive proof system $\mathsf{Unruh}(\Sigma)$ with access to a random oracle that is correct, zero-knowledge and simulation-sound online-extractable, if Σ is correct and has special soundness as well as honest-verifier zero-knowledge.*

Note that we did not define simulation-sound online extractability. The idea behind this is that the extractor does not get access to a list of queries the adversary did, but instead gets a description of the state of the oracle. However, for this thesis, it suffices to know that the extractor can successfully extract a witness for the non-interactive proof. This means that whenever we use the Fiat-Shamir transform, we can use Unruh's transform instead to make the proof secure against a quantum adversary. As expected, this comes at a cost. A proof generated with Unruh's transform consists of multiple transcripts of the Σ-protocol, while the Fiat-Shamir transform uses only one.

3.11 Multi-Party Computation Protocol

A multi-party computation protocol is a protocol where multiple parties compute an output of a function together. Every party gets its own, private input and can send messages to other parties. At the end of their interaction, they each output some (possibly different) value. We adopt the formal definitions of [9].

Definition 3.33 *A multi-party computation (MPC) protocol* Π *for a function* $f : \mathcal{X} \times \mathcal{W}_1 \times .. \times \mathcal{W}_n \to \mathcal{Y}^n$ *with parties* $P_1, .., P_n$*, public input* $x \in \mathcal{X}$ *and private input* $w_i \in \mathcal{W}_i$ *is defined by a deterministic polynomial-time algorithm* Next*, called the next message function.*

- $\mathsf{Next}(i, x, w_i, r_i, (m_1, .., m_j))$*: the algorithm is given index* $i \in [n]$*, public input* $x \in \mathcal{X}$*, private input* $w_i \in \mathcal{W}_i$ *and randomness* r_i*. Furthermore, it gets tuples* m_k *of messages that party* i *received in round* k *for* $1 \leq k \leq j$*. Then, the algorithm* Π *outputs either the tuple of messages party* i *sends in round* $j + 1$ *or a bit indicating that party* i *is finished along with the final (local) output of* i*.*

Denote by the view View_i in a protocol run the tuple of the public input x, private input w_i, the randomness r_i and all messages received by party i.

Definition 3.34 *We call an MPC protocol* Π *for a function* $f : \mathcal{X} \times \mathcal{W}_1 \times .. \times \mathcal{W}_n \to \mathcal{Y}^n$ *correct, if for all* $x \in \mathcal{X}$*, and all* $(w_1, .., w_n) \in \mathcal{W}_1 \times .. \times \mathcal{W}_n$ *we have that*

$$\Pr[r_1, .., r_n \leftarrow \mathcal{R} : \Pi(x, w_1, .., w_n; r_1, .., r_n) = f(x, w_1, .., w_n)] = 1,$$

where $\Pi(x, w_1, .., w_n; r_1, .., r_n)$ *denotes the composed output of all parties in* Π *started with* $(x, w_i; r_i)$ *respectively and* $\mathcal{R}$ *denotes a distribution of uniformly random bit strings of appropriate length.*

When using an MPC protocol, the parties want their private inputs to stay secret. For this notion, there exists two adversarial models. Either, we assume that all parties stay honest and adhere to the protocol, while still trying to learn information about other parties' secret. We call these *honest-but-curious* parties. The other adversarial model is where parties need not adhere to the protocol, i.e. they may send messages that better their chance of learning others' secrets. For our usage of MPC protocol, only the honest-but-curious model is relevant.

Definition 3.35 *We call an MPC protocol* Π *for a function* $f : \mathcal{X} \times \mathcal{W}_1 \times .. \times \mathcal{W}_n \to \mathcal{Y}^n$ *t-private , where* $1 \leq t < n$*, if there exists a ppt simulator* $\mathcal{S}$*, such that for all* $x \in \mathcal{X}$*, all* $(w_1, .., w_n) \in \mathcal{W}_1 \times .. \times \mathcal{W}_n$*, every set of corrupted players* $T \subseteq [n]$ *with* $|T| \leq t$*, we have that the joint view of all parties in* T*, denoted by* $\mathsf{View}_T(x, w_1, .., w_n)$ *is distributed exactly as* $\mathcal{S}(T, x, (w_i)_{i \in T}, (f(x, w_1, .., w_n))_{i \in T})$*.*

3.12 Input Pre-Processing Model

While we later want to use an MPC protocol in a construction, the protocol we require does not fit the standard model. Thus, we create our own model, called *input pre-processing model*.

In the normal MPC protocol model is it assumed that every party possesses its own private input, along with some public input. What we require is that there exists a public input along with a single secret input. Then, we assume there exists a trusted party, that gets the public input and secret input and computes some pre-processing secrets, one for each party, and hands the respective secrets to the parties. The parties engage in their protocol to compute the target function based on the public input and their respective pre-processing secret. In short, the parties use the pre-processing secrets instead of the secret input to compute the function. Furthermore, we want that no party learns what the secret input to the pre-processing was, unless a large enough subset of them acts together.

One could argue that the existence of the trusted party defeats the purpose of the MPC, since the trusted party knows the secret input and can thus compute the function value himself. However, this model can be useful in a scenario where the trusted party has only limited computing power, whereas calculating the function value is costly. Another argument, as mentioned before, is that we need an MPC protocol from this model in a later construction. We intend to simulate the MPC protocol in that construction, thus our use case is very specific. Thus, it is admissible if the model does not reflect the real world accurately.

We begin with the formal definition of an MPC protocol in the input pre-processing model. Since the model is based on a standard MPC protocol, the definitions look similar. Here, however, we have the algorithm Pre in addition, which models the generation of the pre-processing secrets by the trusted party.

Definition 3.36 *A multi-party computation protocol* Π *in the input pre-processing model for a function* $f : \mathcal{X} \times \mathcal{W} \to \mathcal{Y}^n$ *with parties* $P_1, .., P_n$*, public input* $x \in \mathcal{X}$ *and private input* $w \in \mathcal{W}$ *is a tuple* $(\mathsf{Pre}, \mathsf{Next})$ *of a ppt algorithm* Pre *and a deterministic polynomial-time algorithm* Next*.*

- $\mathsf{Pre}(x, w)$*: on input* x, w *the pre-processing algorithm outputs pre-processing secrets* $(w_1, .., w_n)$*.*

- $\mathsf{Next}(i, x, w_i, r_i, (m_1, .., m_j))$*: the algorithm is given index* $i \in [n]$*, public input* x*, pre-processing secret* w_i *and randomness* r_i*. Furthermore, it gets tuples* m_k *of messages that party* i *received in round* k *for* $1 \leq k \leq j$*. Then, the algorithm* Π *outputs either the tuple of messages party* i *sends in round* $j + 1$ *or a bit indicating that party* i *is finished along with the final (local) output of* i*.*

If all entries of the output of f are always the same, we simply speak about a function $f : \mathcal{X} \times \mathcal{W} \to \mathcal{Y}$ and all parties output the same value.

We again can define the view View_i of a party i. As before, it consists of the public input x, the respective pre-processing secrets w_i and randomness r_i, as well as all messages the party received during the protocol execution. Furthermore, we define the correctness of an MPC protocol in this model.

Definition 3.37 *We call an MPC protocol Π in the input pre-processing model for a function $f : \mathcal{X} \times \mathcal{W} \to \mathcal{Y}^n$* ***correct****, if for all $x \in \mathcal{X}$, and all $w \in \mathcal{W}$ we have that*

$$\Pr[(w_1, .., w_n) \leftarrow \mathsf{Pre}(x, w), r_1, .., r_n \leftarrow \mathcal{R} : \Pi(x, (w_i, r_i)_{i\in[n]}) = f(x, w)] = 1,$$

where $\Pi(x, (w_i, r_i)_{i\in[n]})$ denotes the composed output of all parties in Π started with x and their respective w_i, r_i.

This is similar to the standard definition, except that we include the generation of the w_i in the probability.

At last, we also want our MPC protocol to be secure. As in the standard model, we do not want an honest-but-curios set of parties up to a certain size to learn anything about the pre-processing secrets of the other parties. Additionally, we want that no such set of parties is able to learn anything about the original secret input. We reflect these requirements in the formal definition below.

Definition 3.38 *We call an MPC protocol Π in the input pre-processing model for a function $f : \mathcal{X} \times \mathcal{W} \to \mathcal{Y}^n$* **t-*private*** *, where $1 \leq t < n$, if there exists a ppt simulator $\mathcal{S}$, such that for all $x \in \mathcal{X}$, all $w \in \mathcal{W}$, every set of honest-but-curious players $T \subseteq [n]$ with $|T| \leq t$, we have that*

$$\begin{aligned}&\Pr[(w_i)_{i\in[n]} \leftarrow \mathsf{Pre}(x, w) : \mathsf{View}_T(x, (w_i)_{i\in[n]}) = v]\\ =&\Pr[\mathcal{S}(T, x, (f(x, w))_{i\in T}) = v],\end{aligned}$$

where $\mathsf{View}_T(x, (w_i)_{i\in[n]})$ denotes the joint view of all $i \in T$, when the protocol is started with input $(x, (w_i)_{i\in[n]})$ and $f(x, w)_{i\in T}$ denotes $f(x, w)$ restricted to entries with index in T.

3.13 Merkle Tree

When dealing with large datasets, a user may want to show to another that a specific data point is in the set. However, if the verifying party is not in possession of the data set, there is no direct way for the party to check the inclusion themselves. To solve this problem, one can use a Merkle tree. It builds a binary tree of hashes of the dataset, so that the whole data set is represented by a single (small) hash value.

Definition 3.39 *Let $\mathcal{H} = (\mathsf{Kg}, H)$ be some hash function. Let $k \in \mathsf{Kg}(1^\lambda)$. Let $d_1, .., d_n$ be some data points. Assume wlog. that $n = 2^\ell$ for some $\ell \in \mathbb{N}$. Then, a Merkle tree of those data points is a complete binary tree, where the i-th leaf node is $H_k(d_i)$. An inner node then is the hash value of the concatenated values of its*

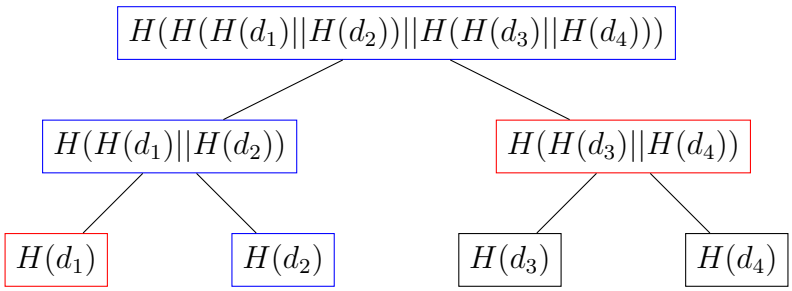

Figure 3.10: A Merkle tree. The blue nodes are the path from d_2 to the root. The red nodes thus belong to the validation set of d_2. The key for the hash function is left out for readability.

children. We call the root of that tree a Merkle root.

Then, to show inclusion of a data point, one needs to present a small number of hashes to convince the verifying party, called validation set.

Definition 3.40 *A validation set of a data point d_i consists of the values of nodes that are siblings to nodes of the path from the data point to the root. We call a set of such values* valid *regarding a root value r, if the successive re-computation of nodes on the shortest path to the root, given only the data point and the validation set, result in r. Furthermore, we say that a validation set matches some Merkle tree, if every node in the validation set can be found in the Merkle tree at the correct position.*

An example of a Merkle tree and a validation set can be seen in Figure 3.10.

Lemma 3.41 *There exists a deterministic polynomial time algorithm that given a data point d and validation set s^* that does not match some Merkle tree, but is valid regarding its root r, computes a collision for the hash function H used to compute the Merkle tree.*

Proof. Since s^* does not match the Merkle tree, there must exist a node $v^* \neq r$ in the partial tree induced by s^* and d that is different from the corresponding node v in the Merkle tree, but their parent nodes must be equal, since s^* is valid. Assume wlog. that v^* is a left child. Thus, if we take the sibling nodes v_s^*, v_s of v^*, v respectively, we know that $(v^*||v_s^*), (v||v_s)$ form a collision for H. □

4 Symmetric-Key Primitive Based Group Signatures

Constructions of group signatures can be classified into different classes. One such class consists of constructions that are based on symmetric-key primitives only. Since the latter are conjectured to be quantum-secure (cf. [5]), such constructions are secure if they are secure against non-quantum adversaries. This property makes the class of symmetric-key based group signatures attractive, as symmetric-key primitives are well understood.

In this chapter, we gradually explain one construction of a group signature that belongs to the aforementioned class, in particular the construction of Katz et al. [10]. Furthermore, we prove that this construction is not secure in the way it was presented by the original authors. However, we show that with a small change the group signature can be made secure. We also discuss the asymptotic efficiency and drawbacks of the construction, as well as the techniques that can be learned from it. Last, we construct a new group signature that draws ideas from the first construction and aims to improve it.

4.1 The Construction of Katz et al.

The idea of the construction of Katz et al. [10] can be explained in three steps. First, they show how to construct a special MPC protocol, which parties can use to compute the output of any circuit together. Then, they construct a proof of knowledge based on this MPC protocol. The proof of knowledge supports every NP-relation by using the verifier of the NP-relation as the circuit for the MPC. After that, they use this proof of knowledge and a special pseudo-random function to create the group signature itself.

4.1.1 An MPC Protocol

Katz et al. construct an MPC protocol in the input pre-processing model, so that for every circuit C consisting of only AND-gates and XOR-gates and any input w to that circuit, the protocol distributes information about w and additional secret input to the parties. Then, they jointly compute $C(w)$, without them getting to know the exact w.

The idea to construct this MPC protocol is to assign a mask λ_α to every wire α of the circuit in the pre-processing step, therefore also to the input. Then, the

$\mathsf{Pre}(x, w)$

1 : Parse $x = C$, where C is a circuit consisting of AND-gates and XOR-gates.
2 : Choose $\mu_\alpha \leftarrow \{0,1\}$ for each circuit input wire α.
3 : Choose $\mu_\beta \leftarrow \{0,1\}$ for each output wire β of an AND-gate.
4 : Iterate through all gates of the circuit: For each output wire γ of an XOR-gate with input wires α and β, compute $\mu_\gamma = \mu_\alpha \oplus \mu_\beta$. For each AND-gate with input wires α and β, compute $\mu_{\alpha,\beta} = \mu_\alpha \cdot \mu_\beta$.
5 : $y = w \oplus \mu_{\alpha_1}||, .., ||\mu_{\alpha_k}$, where $\alpha_1, .., \alpha_k$ are the input wires of C.
6 : For each such $\mu_\alpha, \mu_\beta, \mu_{\alpha,\beta}$ compute the shares $([\mu_\alpha]_i)_{i\in[n]} \leftarrow \mathsf{Share}(\mu_\alpha)$ and $([\mu_\beta]_i)_{i\in[n]}, ([\mu_{\alpha,\beta}]_i)_{i\in[n]}$ respectively.
7 : Set w_i to be the vector of all i-th shares $[\mu_\delta]_i$ with $\delta \in \mathsf{IW} \cup \mathsf{OA} \cup \mathsf{A}$, where IW is the set of input wires, OA is the set of output wires of an AND-gate and A is the set of all identifiers (α, β), where α and β are input wires of an AND-gate.
8 : **return** $(y, (w_i)_{i\in[n]})$

Figure 4.1: A pre-processing algorithm for an MPC protocol.

masks are shared among the parties with a secret sharing scheme, so that every party holds a share $[\lambda_\alpha]$ for every wire α. Furthermore, every party gets to know the masked input $\hat{w}$. With this information, they can jointly compute a masked wire value $\hat{z}_\alpha$ for each wire in the circuit, including the output wire. After that, they each make their wire mask share of *only* the output wire public. Thus, every party can reconstruct the mask share of the output and compute the real wire value z_α for the output wire α, which every party outputs. To realize this, we use the XOR-secret sharing scheme $(\mathsf{Share}, \mathsf{Recon})$, as can be seen in Figures 4.1 and 4.2.

As can be seen in the pre-processing algorithm, we only need to output mask shares of the circuit input wires as well as the output wires of all AND-gates. This is because the mask shares and masked wire values of an XOR-gate output can be computed locally by a party, if the corresponding values of the input wires are known. To be able to compute the masked output value of an AND-gate, however, we also need an additional value for each AND-gate. Thus, we compute and share $\mu_{\alpha,\beta}$ to help with that issue.

Lemma 4.1 *Let C be any circuit consisting of AND-gates and XOR-gates. Let w be an input to that circuit. The pre-processing function from Figure 4.1 and the next-message function from Figure 4.2 define a correct MPC protocol for the function $f(C, w) = C(w)$.*

$\mathsf{Next}(i, y, w_i, r_i, (m_1, .., m_j))$

1: Parse $x = C$, where C is a circuit consisting of AND-gates and XOR-gates.
2: Parse $w_i = (a, b, c)$.
3: Parse a as a vector of $[\mu_\alpha]$, one entry for each input wire α of C.
4: Parse b as a vector of $[\mu_\alpha]$, one entry for each output wire α of an AND-gate in C.
5: Parse c as a vector of $[\mu_{\alpha,\beta}]$, one entry for each AND-gate of C with input wires α, β.
6: Recompute the previous state by invoking $\mathsf{Next}(i, x, w_i, r_i, (m_1, .., m_{j-1}))$.
7: If m_j is a broadcast for the output wire α of the last gate, parse $m_j = ([\mu_\alpha])_{i\in[n]}$, output $\hat{z}_\alpha \bigoplus_{i=1}^{n} [\mu_\alpha]_i$ and stop.
8: Else, parse $m_j = ([s]_i)_{i\in[n]}$.
9: For the j-th AND-gate, compute $\hat{z}_\gamma = \hat{z}_\alpha \hat{z}_\beta \bigoplus_{i=1}^{n} [s]_i$, for input wires α, β, output wire γ.
10: For each XOR-gate "between" the j-th and $j+1$-th AND-gate of C with input wires α, β, compute the value of the output wire by $\hat{z}_\gamma = \hat{z}_\alpha \oplus \hat{z}_\beta$.
11: Furthermore, for that XOR-gate, compute $[\mu_\gamma] = [\mu_\alpha] \oplus [\mu_\beta]$.
12: Then, for output wire γ of the $j+1$-th AND-gate compute $[s] = \hat{z}_\alpha[\mu_\beta] \oplus \hat{z}_\beta[\mu_\alpha] \oplus [\mu_{\alpha,\beta}] \oplus [\mu_\gamma]$.
13: Broadcast the share $[s]$.
14: At any time, after the last gate is processed, broadcast all $[\mu_\alpha]$ for the output wires α of the circuit.

Figure 4.2: A next-message function for an MPC protocol. If α is a wire of C, denote by z_α the value on that wire and by $\hat{z}_\alpha$ the masked value. Assume wlog. that the gates in the description of C are ordered by depth.

Proof. We prove this with an invariant stating that the masked values $\hat{z}_\alpha$ that the parties compute are always the real value z_α XOR-ed with the mask, i.e. $\hat{z}_\alpha = z_\alpha \oplus \mu_\alpha$ for any wire α of C. First, let α, β be the input wires of an XOR-gate and γ be the output wire. Assume that each party holds values $\hat{z}_\alpha, \hat{z}_\beta$ fulfilling the invariant. Then, we know that

$$\begin{aligned}\hat{z}_\alpha \oplus \hat{z}_\beta =& z_\alpha \oplus \mu_\alpha \oplus z_\beta \oplus \mu_\beta \\ =& z_\alpha \oplus z_\beta \oplus \mu_\alpha \oplus \mu_\beta \\ =& z_\gamma \oplus z_\gamma.\end{aligned}$$

Thus, every party can compute the correct $\hat{z}_\gamma$.

An observation that we need for the XOR-gates is that assuming each party knows its shares $[\mu_\alpha], [\mu_\beta]$ for the input wires of an XOR-gate, it can compute its share $[\mu_\gamma] = [\mu_\alpha] \oplus [\mu_\beta]$ for the output wire γ of the XOR-gate. This is correct, since we have that

$$\begin{aligned}\mu_\alpha \oplus \mu_\beta =& \bigoplus_{i=1}^{n} [\mu_\alpha]_i \oplus [\mu_\beta]_i \\ =& \bigoplus_{i=1}^{n} [\mu_\gamma]_i \\ =& \mu_\gamma.\end{aligned}$$

Now, let α, β be the input wires of an AND-gate, while γ is the output wire. Assume that each party holds values $\hat{z}_\alpha, \hat{z}_\beta$ fulfilling the invariant. By the observation we can assume that each party holds or computes its shares $[\mu_\alpha], [\mu_\beta]$. Furthermore, by definition each party has $[\mu_{\alpha,\beta}]$ and $[\mu_\gamma]$. The parties compute $[s]$ locally, broadcast their share and compute $\hat{z}_\gamma = \hat{z}_\alpha \hat{z}_\beta \bigoplus_{i=1}^{n} [s]_i$. This is the correct masked value, due to the following equations. By definition, we know that

$$\begin{aligned}\hat{z}_\gamma =& \hat{z}_\alpha \hat{z}_\beta \oplus \bigoplus_{i=1}^{n} [s]_i \\ =& \hat{z}_\alpha \hat{z}_\beta \oplus \bigoplus_{i=1}^{n} \hat{z}_\alpha [\mu_\beta]_i \oplus \hat{z}_\beta [\mu_\alpha]_i \oplus [\mu_{\alpha,\beta}]_i \oplus [\mu_\gamma]_i.\end{aligned}$$

Using the distributive property and the definition, we get

$$\begin{aligned}\hat{z}_\gamma =& \hat{z}_\alpha \hat{z}_\beta \oplus \hat{z}_\alpha \left(\bigoplus_{i=1}^{n} [\mu_\beta]_i \right) \oplus \hat{z}_\beta \left(\bigoplus_{i=1}^{n} [\mu_\alpha]_i \right) \oplus \left(\bigoplus_{i=1}^{n} [\mu_{\alpha,\beta}]_i \right) \oplus \left(\bigoplus_{i=1}^{n} [\mu_\gamma]_i \right) \\ =& \hat{z}_\alpha \hat{z}_\beta \oplus \hat{z}_\alpha \mu_\beta \oplus \hat{z}_\beta \mu_\alpha \oplus \mu_{\alpha,\beta} \oplus \mu_\gamma.\end{aligned}$$

Then, using the distributive property twice, we get

$$\begin{aligned}\hat{z}_\gamma =& \hat{z}_\alpha(\hat{z}_\beta \oplus \mu_\beta) \oplus \mu_\alpha(\hat{z}_\beta \oplus \mu_\beta) \oplus \mu_\gamma \\ =& \hat{z}_\alpha z_\beta \oplus \mu_\alpha z_\beta \oplus \mu_\gamma \\ =& (\hat{z}_\alpha \oplus \mu_\alpha) z_\beta \oplus \mu_\gamma \\ =& z_\alpha z_\beta \oplus \mu_\gamma \stackrel{!}{=} z_\gamma \oplus \mu_\gamma,\end{aligned}$$

thus the computed value is the correct masked value. □

Lemma 4.2 *Let C be any circuit consisting of AND-gates and XOR-gates. Let w be an input to that circuit. The MPC protocol in the input pre-processing model for the function $f(C, w) = C(w)$ defined by the pre-processing function from Figure 4.1 and the next-message function from Figure 4.2 has $n-1$-privacy.*

Proof. The secret inputs and the messages sent are shares of the XOR-secret sharing scheme. Since the n-out-of-n XOR-secret sharing scheme is perfectly $n-1$-private, for any set of parties T with $|T| \leq n-1$ the shares look uniformly random. Thus, a simulator for the MPC protocol outputs a uniformly random value for each required witness and message. □

4.1.2 A Proof of Knowledge for Any NP-Relation

With the MPC control from above, we can construct a Σ-protocol for any NP-relation, that we later turn into a (non-interactive) proof of knowledge. The idea behind the Σ-protocol is the so-called *MPC-in-the-head paradigm*. It tells us that the prover in a proof of knowledge executes the MPC protocol by himself by simulating all parties. The circuit, for which he does this, simply is a verification algorithm deciding whether a witness belongs to the NP-relation. Then, the prover commits to the initial views and messages sent by the simulated parties and sends the commitments to the verifier. That verifier chooses one of the parties at random and responds to the prover with it as the challenge. The prover opens the views by all parties except for the challenged one together with all messages sent and reveals that to the verifier. To verify the proof, the verifier then has to recompute the MPC protocol by himself with the information given to him by the prover. Since he knows the secret inputs for all parties except for one, the verifier recomputes the view of all of them, but whenever they expect a message from the challenged party, the verifier uses the message provided by the prover. If everything is consistent and the MPC protocol outputs the desired output, the verifier accepts. Honest-verifier zero-knowledge is then guaranteed by the hiding of the commitment and privacy of the MPC protocol, while special soundness follows from the binding of the commitment.

In our case, we have to adapt the MPC-in-the-head paradigm a bit. Since our MPC protocol also has a pre-processing stage, we have to include that in the protocol as well. This works in a way similar to before. Instead of committing to

the initial view of the parties as well as the messages, we now have two stages. In the first, the prover computes several outputs of the pre-processing algorithm. Then, for each such output and each party, the prover commits to the initial view of the party for that pre-processing output. The verifier then chooses all but one of the pre-processing outputs that the prover has to open *fully*. The output that he did not choose remains unopened. After that, in the second stage, we continue with the standard MPC-in-the-head paradigm, where we use the unopened pre-processing output to simulate the MPC protocol. Let $(\mathsf{msgs}_i)_{i\in[n]} \leftarrow \mathsf{MPC}(C, (y, (w_i)_{i\in[n]}))$ be the shorthand for the messages output by the parties when starting the MPC protocol defined by Figure 4.2 with input $(C, (y, (w_i)_{i\in[n]}))$, where msgs_i is the vector of messages sent by party i. We can formally define the protocol described above:

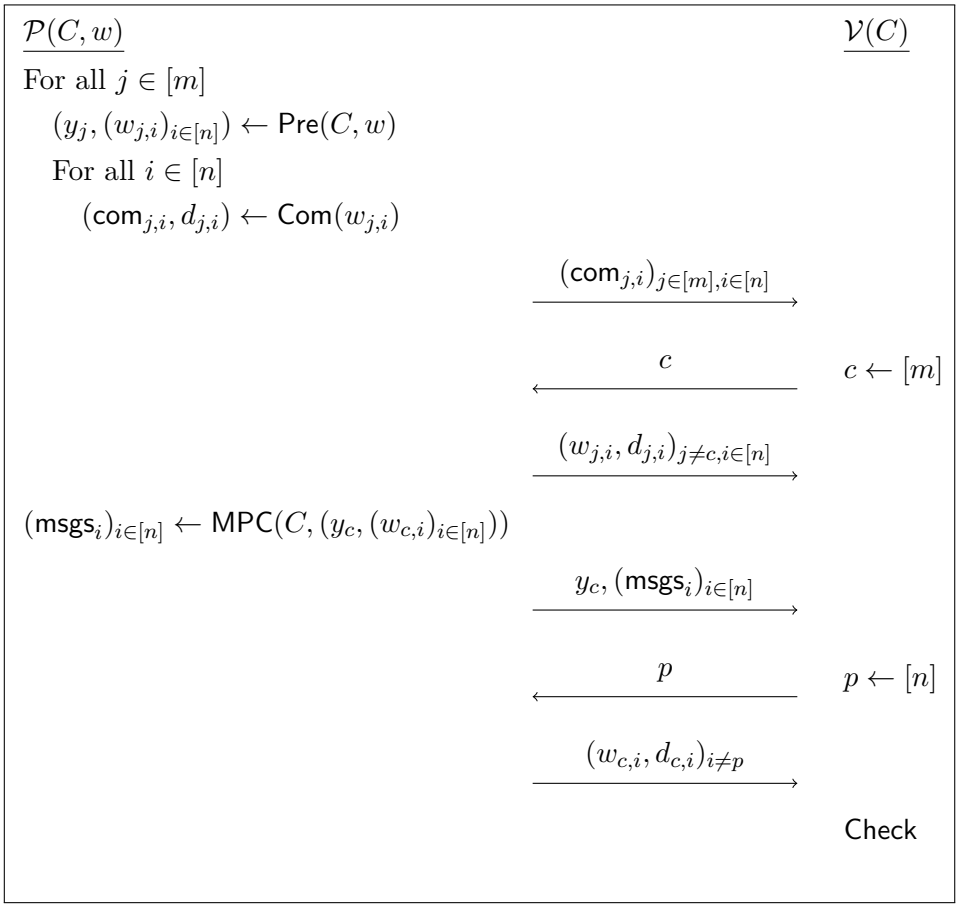

Figure 4.3: A 5-round protocol. Check is an algorithm that takes all information available to the verifier, recomputes the simulation of the MPC protocol based on this and checks for consistency. If everything is consistent, it outputs 1, else 0. For this protocol, we do not formally define Check.

Although this is a 5-round protocol and we want to have a Σ-protocol, i.e. three rounds, it helps us understand the three-round version. To construct a three-round

version from the 5-round protocol, we compress the two stages into one. Thus, the prover prepares several pre-processing outputs and immediately simulates the MPC protocol for each output and commits as usual. The verifier then chooses pre-processing stages and parties that he wants to be opened together and the prover responds with the requested information.

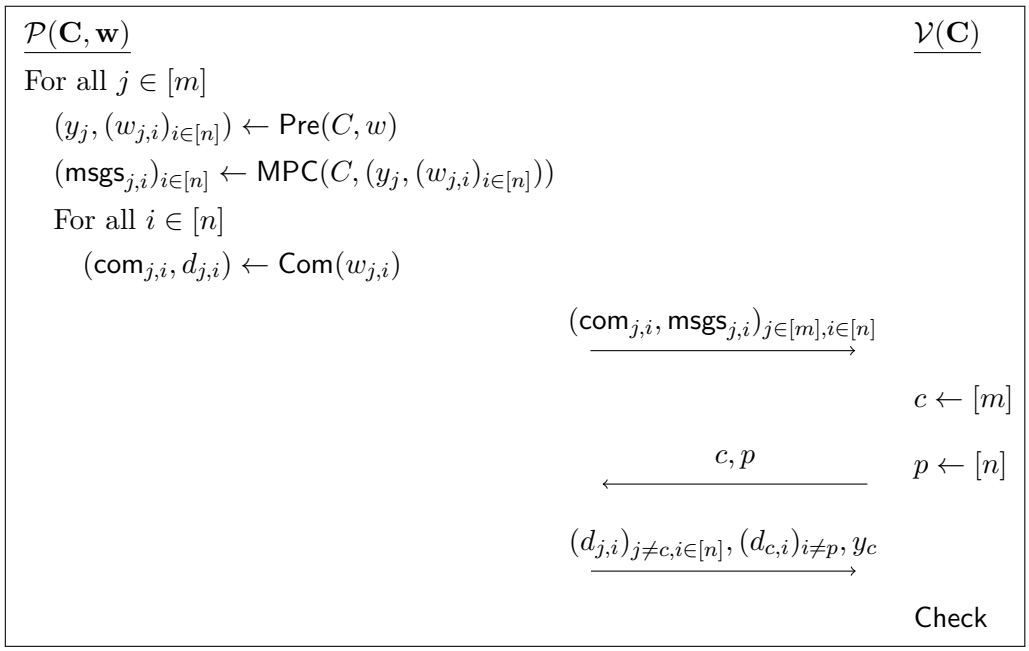

Figure 4.4: A Σ-protocol. Check is defined below.

Check gets as input the whole view of the verifier. It opens all $\mathsf{com}_{c,i}$ for $i \in [n]$ to get the private inputs for the parties. For each $j \neq c$, the it checks whether the masks $\mu_{\alpha,\beta}$ for the AND-gates are consistent with the other masks, i.e. if $\mu_{\alpha,\beta} = \mu_\alpha \oplus \mu_\beta$. Then, the Check uses the private inputs of pre-processing output c and y_c to simulate every non-challenged party of the c-th iteration of the MPC protocol himself, i.e. $\mathsf{MPC}(C, (y_c, (w_{c,i})_{i\in[n]}))$. Note that it does not know $w_{c,p}$, thus he does not simulate party p. Instead, whenever a simulated party expects a message from p, Check uses the messages provided by $\mathsf{msgs}_{c,p}$. During this, he checks whether the computed messages match the $\mathsf{msgs}_{c,i}$. If they do and the output of all opened parties is 1, the verifier accepts. With the protocol formally described, we can state and prove some properties it has, namely that it is a Σ-protocol.

Theorem 4.3 *For any circuit C consisting of only AND-gates and XOR-gates, the protocol from Figure 4.4 is a Σ-protocol for the relation $R(C, w) = 1 \Leftrightarrow C(w) = 1$.*

Lemma 4.4 *For any circuit C consisting of only AND-gates and XOR-gates, the Σ-protocol from Figure 4.4 is correct.*

$\mathcal{S}(C)$

1: Choose $c \leftarrow [m], p \leftarrow [n]$.
2: For $j \in [m]$:
3: $\quad (y_j, (w_{j,i}, \mathsf{msgs}_{j,i})_{i\in[n]}) \leftarrow \mathcal{S}_\Pi([n]\backslash\{p\}, C, 1)$
4: Compute $\mathsf{com}_{j,i}$ as the honest prover would for $j \in [m], i \in [n]\backslash\{p\}$.
5: Compute $\mathsf{com}_{j,p} \leftarrow \mathsf{Com}(a)$ for $j \in [m]$ with $a \leftarrow \{0,1\}^{|w_i|}$.
6: Output $\Big((\mathsf{com}_{j,i}, \mathsf{msgs}_{j,i})_{j\in[m],i\in[n]}, \quad c, p,$
$\quad (d_{j,i})_{j\in[m]\backslash\{c\},i\in[n]}, (d_{c,i})_{i\in[n]\backslash\{p\}}, y_c, \mathsf{msgs}_{c,p}\Big)$.

Figure 4.5: A simulator for the Σ-protocol from Figure 4.4

Proof. The underlying MPC protocol is correct, thus the MPC protocol started with the input and messages generated by the prover would output 1 at the end. Furthermore, since the commitment scheme is correct, the sent commitments along with the opening values open to the private inputs that were computed by the prover. This, together with the fact that the protocol is deterministic, means that when the verifier simulates the MPC protocol himself, he gets the same messages and outputs as the prover. Thus, the verifier accepts. □

Lemma 4.5 *For any circuit C consisting of only AND-gates and XOR-gates, the protocol from Figure 4.4 has honest-verifier zero-knowledge, if the commitment scheme used is perfectly hiding and the MPC protocol has $n-1$-privacy.*

Proof. Let C be a circuit consisting of only AND-gates and XOR-gates. Let $\mathcal{S}_\Pi$ be the simulator of the underlying MPC protocol. Construct a honest-verifier zero-knowledge simulator $\mathcal{S}$ as seen in figure Figure 4.5.

Then, $\mathcal{S}$ outputs transcripts with the same distribution as the real protocol, due to the following reasons: The values $\mathsf{com}_{j,i}$ for $j \in [m], i \in [n]\backslash\{p\}$ are computed as the honest prover would, on values that are generated by the pre-processing simulator. $\mathsf{com}_{j,p}$ for $j \in [m]$ is computed honestly on a random value a of appropriate length. Thus, by the perfect hiding property of the commitment scheme, these values have the correct distribution as well. The $\mathsf{msgs}_{j,i}$ values for $j \in [m], i \in [n]$ are generated by a simulator and thus have the correct distribution by definition. c and p are chosen as in the real protocol. For the $d_{j,i}$ we have that the values are either generated honestly on simulator output (for $i \neq p$) or we have the following case: If $j \in [m]\backslash\{c\}$ and $i = p$, the verifier does not get to know y_j, thus he cannot verify whether the a, that the commitment opens to, is consistent with the rest of the pre-processing. Thus, a is consistent with the adversary's view. At last, y_c and $\mathsf{msgs}_{c,p}$ are generated by the MPC simulator and thus have the correct distribution. □

Lemma 4.6 *For any circuit C consisting of only AND-gates and XOR-gates, the protocol from Figure 4.4 has special soundness, if the commitment scheme used is computationally binding.*

Proof. Assume we have two accepting transcripts (a, z, r) and (a, z', r') with $z \neq z'$ and $z = (c, p)$. We construct an extractor $\mathcal{E}$ that differentiates between the cases where either $c \neq c'$ or $c = c', p \neq p'$. In the first case, the extractor takes y_c from the first transcript and $d_{c,i}$ for $i \in [n]$ from the second transcript. In the second case, he takes y_c and $(d_{c,i})_{i\in[n]\setminus\{p\}}$ from the first transcript and $d_{c,p}$ from the second transcript. In either case, the extractor then uses these opening values along with the commitments from the announcement to compute $(w_{c,i})_{i\in[n]}$. At last, he outputs $w = y_c \bigoplus_{i=1}^{n} w_{c,i}$.

What we need to prove now is that this extractor always outputs a valid witness for C, unless the binding of the commitment fails. This is due to the announcement and thus the commitments being equal. If the binding does not fail, we have either $c \neq c'$ or $c = c', p \neq p'$. In the first case, we get from the first transcript the value y_c. Furthermore, we get the $(w_{c,i})_{i\in[n]}$ from the second transcript, which by the non-failed binding of the commitment are the same as for the first transcript. Thus, if we XOR all the secrets together, we get the mask for the input and then can calculate the non-masked input together with y_c. Therefore, we have that $C(w) = 1$ for the w the extractor outputs.

In the second case, the transcripts different sets of private inputs for the same pre-processing output (provided the binding does not fail). If we now take the conjunction of the private inputs, we know all of them, thus we can again unmask y_c and we have that $C(w) = 1$. However, if the binding of the commitment does fail, we cannot be sure whether the w output by the extractor is valid or not, since then the commitment opens to another value. Let $\mathsf{negl}(\cdot)$ be the negligible limit for the success chance of any adversary against the binding of the commitment scheme. Then, we have that

$$\Pr[w \leftarrow \mathcal{E}((a, c, r), (a, c', r')) : C(w) = 1] \leq 1 - n \cdot \mathsf{negl}(\lambda),$$

since we have n points where the binding may fail. □

Then, we can turn this Σ-protocol into a NIZKPoK by either the Fiat-Shamir transform or Unruh's transform.

Construction 4.7 *Let Σ be the Σ-protocol from Figure 4.4. Construct a non-interactive proof system by transforming Σ with the Fiat-Shamir transform, i.e. construct $\mathsf{sFS}(\Sigma)$. By Theorem 3.31 we know that this is a NIZKPoK.*

Construction 4.8 *Let Σ be the Σ-protocol from Figure 4.4. Construct a non-interactive proof system by transforming Σ with Unruh's transform, i.e. construct $\mathsf{Unruh}(\Sigma)$. By Theorem 3.32 we know that this is a non-interactive zero-knowledge simulation-sound online-extractable proof system.*

Note that the Fiat-Shamir transform requires that the Σ-protocol has an exponentially large challenge space in order to produce secure non-interactive proof system. While the Σ-protocol from Figure 4.4 does not have such a challenge space, we can repeat the protocol a certain number of times to achieve a large enough space. Thus, we wlog. assume that Construction 4.7 produces a zero-knowledge simulation-sound extractable non-interactive proof system.

4.1.3 The Group Signature of Katz et al.

In general, when designing a secure group signature scheme, we need something that guarantees authenticity of the signer and a way for the group manager to correctly open a signature. Furthermore, the group signature must not reveal who signed it. In the standard construction (cf. Chapter 1) authenticity is covered by the digital signature and traceability by the encryption scheme, while the proof of knowledge is used to tie the two components together. An adversary cannot identify the signer, as else he would be able to either compute the signing key or the message in the encryption.

To achieve a better efficiency in their scheme compared to the standard construction, Katz et al. replace the digital signature by the validation set of a Merkle tree to guarantee authenticity. Each user is assigned a pair k_0, k_1 of PRF keys as his secret key and a leaf node in the Merkle tree, where the value of that leaf is $f(k_0, 0^\lambda)||f(k_1, 0^\lambda)$. Then, the root of the Merkle tree is the public key of the group signature.

Furthermore, Katz et al. do not use an encryption scheme to open a signature. Instead, they add an image of the PRF to the signature, parameterized by k_0 and the message to be signed. Then, the master secret key consists of all k_0 from the users. The group manager can then trace a signature by trying out all k_0 he knows until he finds one that fits.

To complete the signature, a user cannot show his validation set directly, as else he can be identified. Thus, the second part of the signature is a NIZKPoK proving that his validation set is valid regarding the public Merkle root. Furthermore, the user proves that his k_0 is also used to compute the image of the function, therefore tying together the validation set and the image.

To formally describe this, we first construct a circuit that we use to instantiate the proof of knowledge.

Construction 4.9 *Let $f : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a function. Let $x \in \mathcal{X}, y \in \mathcal{Y}$ and $k_0, k_1 \in \mathcal{K}$ and y^*, s be some values. We define a circuit C_{x,y,y^*} that on input (k_0, k_1, s) computes $y_b = f(k_b, 0^\lambda)$ for $b \in \{0, 1\}$. Then, if $y = f(k_0, x)$ and s is a valid validation set for the value (y_0, y_1) with respect to y^*, the circuit outputs 1.*

Using this, we can construct the group signature scheme of Katz et al.

Construction 4.10 *Let $\mathcal{H} = (\mathcal{H}.\mathsf{Kg}, H)$ be a hash function. Let $\mathcal{F} = (\mathcal{F}.\mathsf{Kg}, f)$ be a keyed function. Then, we can construct a fixed size group signature scheme.*

- $\mathsf{Kg}(1^\lambda, 1^\ell)$: *Generate $k_{hash} \leftarrow \mathcal{H}.\mathsf{Kg}(1^\lambda)$. For each user $i \in [\ell]$, generate*

$k_0^{(i)}, k_1^{(i)} \leftarrow \mathcal{F}.\mathsf{Kg}(1^\lambda)$ *and compute* $y_b^{(i)} \leftarrow f(k_b^{(i)}, 0^\lambda)$ *for* $b \in \{0,1\}$. *Compute a Merkle tree for data points* $(y_0^{(i)}, y_1^{(i)})_{i\in[\ell]}$ *with* $H_{k_{hash}}$. *Let* y^* *be the root of that Merkle tree. Compute the validation set* s_i *for each data point. Return* $\mathsf{gpk} = y^*$, $\mathsf{msk} = (k_0^{(i)})_{i\in[\ell]}$ *and* $\mathsf{usk}_i = (k_0^{(i)}, k_1^{(i)}, s_i)$ *for each user* i.

- $\mathsf{Sign}(\mathsf{usk}_i, m)$: *Parse* $\mathsf{usk}_i = (k_0^{(i)}, k_1^{(i)}, s_i)$. *Compute* $y = f_{k_0^{(i)}}(m)$. *Use the prover of Construction 4.7 to create a non-interactive proof* π *for circuit* $C_{m,y,\mathsf{gpk}}$ *with input* $(k_0^{(i)}, k_1^{(i)}, s_i)$, *where the circuit is defined as in Construction 4.9. Output* $\sigma = (\pi, y)$.

- $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma)$: *Parse* $\sigma = (\pi, y)$. *Let* $C_{m,y,\mathsf{gpk}}$ *be the circuit from Construction 4.9. Use the verifier of Construction 4.7 to check whether the proof* π *is valid for that circuit. If it is, output 1. Else, output 0.*

- $\mathsf{Open}(\mathsf{msk}, m, \sigma)$: *Parse* $\mathsf{msk} = (k_0^{(i)})_{i\in[\ell]}$. *Parse* $\sigma = (\pi, y)$. *If* $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma) = 0$, *output* $\perp$. *Else check for each* $i \in [\ell]$ *whether* $f(k_0^{(i)}, m) = y$. *If there is such an* i, *output the first of them. If no such* i *can be found, output* $\perp$.

Lemma 4.11 *If Construction 4.7 is a correct non-interactive argument, then Construction 4.10 is a correct fixed size group signature.*

Proof. Let λ be a security parameter and $\ell \in \mathbb{N}$. Let $m \in \mathcal{M}$ and $i \in [\ell]$. Let $(\mathsf{gpk}, \mathsf{msk}, (\mathsf{usk}_i)_{i\in[\ell]}) \in [\mathsf{Kg}(1^\lambda, 1^\ell)]$. Then, for $\sigma = (\pi, y) \leftarrow \mathsf{Sign}(\mathsf{usk}_i, m)$, we know that by the correctness of Construction 4.7 the verifier of that proof returns 1. Thus, we also have that $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma) = 1$. Furthermore, we know that $y = f(k_0^{(i)}, m)$. Therefore, $\mathsf{Open}(\mathsf{msk}, m, \sigma)$ will find some $k_0^{(j)}$ such that $y = f(k_0^{(j)}, m)$, either with $j = i$ or $j \neq i$. However, the case that $j \neq i$ happens only with negligible probability, since the keys are generated by the key generation of a pseudo-random function. □

Attack on Traceability

As stated before, Katz et al. intend to use a pseudo-random function to instantiate $\mathcal{F}$. However, it is possible to construct a pseudo-random function that makes it easy to find keys $k \neq k'$ such that $f(k, 0^\lambda) = f(k', 0^\lambda)$ An adversary can use this to create signatures that cannot be traced, as can be seen in the proof of the following theorem.

Theorem 4.12 *There exists a pseudo-random function* PRF', *such that Construction 4.10 does not offer traceability when* $\mathcal{F}$ *is instantiated with* PRF'.

Proof. Let $\mathsf{PRF} = (\mathsf{Kg}, f)$ be some pseudo-random function. Construct an algorithm Kg' that on input 1^λ uses $k \leftarrow \mathsf{Kg}(1^\lambda)$, chooses $x^* \leftarrow \mathcal{X}$ from the associated

input space and computes $y^* \leftarrow f(k, x^*)$. It then outputs $k' = (k, x^*, y^*)$. Furthermore, define a function

$$f'(k', x) = \begin{cases} f(k, x), & x \neq x^* \\ y^*, & x = x^* \end{cases} \text{ for } k' = (k, x^*, y^*).$$

Now, we argue that $\mathsf{PRF}' = (\mathsf{Kg}', f')$ is still a pseudo-random function. Let $\mathcal{A}$ be an adversary that has non-negligible advantage against PRF'. Then, $\mathcal{A}$ has the same advantage against PRF, since for a key generated by Kg', we have that f' behaves exactly as f. Thus, PRF' is a pseudo-random function as well.

However, if we instantiate Construction 4.10 with $\mathcal{F} = \mathsf{PRF}'$ we can attack the traceability of the group signature. In the attack, we corrupt a user i to get his $k_0^{(i)}, k_1^{(i)}$ and s_i, from which we can compute $y_0 = f'(k_0^{(i)}, 0^\lambda)$ and y_1 analogously. Furthermore, we choose $k \leftarrow \mathsf{Kg}(1^\lambda)$ such that $f(k, m) \neq f'(k_0^{(i)}, m)$ for some message m. We can find such a k easily since f' is a pseudo-random function. Note that k is a key of PRF, while the $k_b^{(i)}$ are keys of PRF'. Then, we can construct keys $k_b^* = (k, 0^\lambda, y_b)$ for $b \in \{0, 1\}$ If we use those keys along with the s_i and m as input to the sign algorithm to get a signature $\sigma = (\pi, y)$. This signature is valid, since $f'(k_b^*, 0^\lambda) = y_b$ by definition and it is created by the signing algorithm. However, if the group manager computes $j \leftarrow \mathsf{Open}(\mathsf{msk}, m, y)$, we have that $j \neq i$, since we chose k such that $f(k, m) \neq f'(k_0^{(i)}, m)$. Thus, we created a signature that is not traceable.

□

Restoring Traceability

As we have seen in Theorem 4.12, if Construction 4.10 instantiated with $\mathcal{F}$ being a pseudo-random function, then the construction does not offer necessarily traceability. However, in order to get a secure group signature, we want to have that property. To do that, we require an additional property from the pseudo-random function that prevents our and similar attacks. In our attack we used that it is easy for an adversary, given a key, to find another key such that for some input the function evaluates to the same for either key, i.e. $\Pr[(k', m) \leftarrow \mathcal{A}(k) : f(k, m) = f(k', m) \wedge k \neq k']$ is high. To prevent this, we require that the function is a one-way function if we switch the roles of key and input, formally we require that for all $x \in \mathcal{M}$ and all adversaries $\mathcal{A}$ we have that $\Pr[k \leftarrow \mathsf{Kg}(1^\lambda), k^* \leftarrow \mathcal{A}(1^\lambda, f(k, x)) : f(k, x) = f(k^*, x)]$ is negligible in λ. If we assume that property, we can prove traceability of Construction 4.10.

Lemma 4.13 *If the protocol from Construction 4.7 is a non-interactive proof of knowledge, $\mathcal{H}$ is collision resistant and $f(\cdot, x)$ is a one-way function for all inputs x, then Construction 4.10 offers traceability.*

The intuition why this holds is as follows. In order to create a valid signature, an adversary has to use a validation set s that was issued by the trusted party. If

not, the adversary was able to find a collision of the hash function. Furthermore, the adversary must prove possession of keys such that the values of the pseudo-random function at 0^λ are a leaf in the Merkle tree, such that the used s is a valid validation set. If he can prove possession of such keys he either was able to find a hash collision that occurred when computing the leaf or he knows keys that evaluate to the value in the leaf. Since the adversary wants to output a signature that does not trace to him, he then has to use keys that are either unknown to the trusted party or are from an honest user. However, the one-wayness of the pseudo-random function prevents the adversary from computing such keys.

Proof. Let Π be the group signature from Construction 4.10. Let $\mathcal{E}$ be the knowledge extractor of Construction 4.7 and $\mathcal{S}$ the simulator. Assume that the construction is not traceable. Then, there exists an adversary $\mathcal{A}$ with non-negligible probability to win $\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda)$. From that, we construct adversary $\mathcal{A}_{\mathsf{OWF}}$ against the one-wayness of f.

$\mathcal{A}_{\mathsf{OWF}}(1^\lambda, y^*)$

1: Choose a user $i \leftarrow [\ell]$ for some ℓ and initialize empty sets $\mathbb{M}, \mathbb{C}$.
2: Generate $(\mathsf{gpk}, \mathsf{msk}, (\mathsf{usk}_i)_{i\in[\ell]}) \leftarrow \mathsf{Kg}(1^\lambda, 1^\ell)$ while storing the Merkle tree.
3: Replace y_1 of user i by y^* and recompute the Merkle tree honestly and store it.
4: Simulate $\mathcal{A}^{\mathcal{S}.\mathcal{O}(\cdot)}$ by giving him $\mathsf{gpk}, \mathsf{msk}$. Answer his oracle queries as in the tracing game, except for the following queries involving i:
   - $\mathsf{Sig}(i, m)$: Reply with $\sigma = \left(\mathcal{S}.\mathcal{O}_P(C_{m,f(k_0^{(i)},m),\mathsf{gpk}}), f(k_0^{(i)}, m)\right)$. Add (m, i) to $\mathbb{M}$.
   - $\mathsf{Corrupt}(i)$: Abort.
5: $\mathcal{A}$ outputs m^*, σ^*.
6: If $\mathsf{Vrfy}(\mathsf{gpk}, m^*, \sigma^*) = 1$ and $\mathsf{Open}(\mathsf{msk}, m^*, \sigma^*) = j \notin \mathbb{C}$ and $(m^*, j) \notin \mathbb{M}$ hold, continue, else abort.
7: If $j \neq i$, abort.
8: Use $\mathcal{E}$ on $\mathcal{A}$ to get some (k_0^*, k_1^*, s^*).
9: If s^* matches the Merkle tree, $f(k_0^*, 0^\lambda) = f(k_0^{(i)}, 0^\lambda)$ and $f(k_1^*, 0^\lambda) = y^*$, return k_1^*.

Figure 4.6: An adversary against the one-wayness of f.

Furthermore, we construct adversary $\mathcal{A}_{\mathcal{H}}$ against the collision-resistance of $\mathcal{H}$.

$\mathcal{A}_{\mathcal{H}}(1^\lambda, k)$

1: Initialize empty sets $\mathbb{M}, \mathbb{C}$.
2: Generate $(\mathsf{gpk}, \mathsf{msk}, (\mathsf{usk}_i)_{i\in[\ell]}) \leftarrow \mathsf{Kg}(1^\lambda, 1^\ell)$ for some ℓ, but instead of generating k_{hash}, set $k_{hash} = k$. Additionally, store the Merkle tree.
3: Simulate $\mathcal{A}$ by giving him $\mathsf{gpk}, \mathsf{msk}$. Answer oracle queries as in the tracing game.
4: $\mathcal{A}$ outputs m^*, σ^*.
5: If $\mathsf{Vrfy}(\mathsf{gpk}, m^*, \sigma^*) = 1$ and $\mathsf{Open}(\mathsf{msk}, m^*, \sigma^*) =: i \notin \mathbb{C}$ and $(m^*, i) \notin \mathbb{M}$, use $\mathcal{E}$ to get some (k_0^*, k_1^*, s^*).
6: If s^* matches the Merkle tree and $\nexists j \in [\ell]$ such that $f(k_0^*, 0^\lambda) = f(k_0^{(j)}, 0^\lambda)$ and $f(k_1^*, 0^\lambda) = f(k_1^{(j)}, 0^\lambda)$, then return collision $(f(k_0^*, 0^\lambda), f(k_1^*, 0^\lambda)), (f(k_0^{(i)}, 0^\lambda), f(k_1^{(i)}, 0^\lambda))$.
7: Else, compare s^* to the Merkle tree. Compute and output the collision.

Figure 4.7: An adversary against the collision-resistance of $\mathcal{H}$.

For these two adversaries, we can define an event *Extract*, that happens if the extractor $\mathcal{E}$ outputs a valid witness. Then, we can define two events $OW, Hash$ that can happen after $\mathcal{A}$ outputs a forgery $\sigma^* = (\pi, y)$ for a message m^* in the traceability security game and we successfully extract some (k_0^*, k_1^*, s^*). The first event, OW, happens if every node of s^* output by the extractor can be found at the correct location in the Merkle tree (s^* matches the tree) and $f(k_0^*, 0^\lambda) = f(k_0^{(i)}, 0^\lambda)$ and $f(k_1^*, 0^\lambda) = f(k_1^{(i)}, 0^\lambda)$, where $j = \mathsf{Open}(\mathsf{msk}, m^*, \sigma^*)$. Thus, the adversary used an s^* that was issued and the keys he used do not form a hash collision.

The second event, *Hash*, happens either if s^* does not match the tree or if it matches the tree, but either $f(k_0^*, 0^\lambda) \neq f(k_0^{(i)}, 0^\lambda)$ or $f(k_1^*, 0^\lambda) \neq f(k_1^{(i)}, 0^\lambda)$. Then, the adversary used a hash collision to form the signature. This collision is found either in the tree (if s^* does not match the tree) or when computing the leaf.

From the construction of $\mathcal{A}_{\mathsf{OWF}}$ and $\mathcal{A}_{\mathcal{H}}$ we can see that they perfectly simulate the view of $\mathcal{A}$. While $\mathcal{A}_{\mathcal{H}}$ uses exactly the same steps and computations as the traceability game, $\mathcal{A}_{\mathsf{OWF}}$ uses the zero knowledge simulator to generate proofs (and thus signatures) of a random user i. However, the simulator outputs proofs with the exact same distribution, thus the view of $\mathcal{A}$ is perfectly simulated. Only in the case that $\mathcal{A}$ queries $\mathsf{Corrupt}(i)$ does $\mathcal{A}_{\mathsf{OWF}}$ not simulate the view perfectly, as it then aborts. However, if $\mathsf{Corrupt}(i)$ is queried in the real tracing game and the forgery opens to $j = i$, the adversary loses anyways and if $j \neq i$, we do not

care about the forgery and abort. Thus, we have that

$$\begin{aligned}\Pr[\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract] = \;& \Pr[OW\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract]\\ &+\Pr[Hash\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract]\\ \Leftrightarrow \Pr[\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1] =& (\Pr[Extract|\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1])^{-1}\\ &\cdot\Big(\Pr[OW\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract]\\ &+\Pr[Hash\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract]\Big).\end{aligned}$$

At this point, we wlog. assume that $\Pr[Extract|\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1]\geq\frac{1}{c}$ for some constant $c\in\mathbb{N}$, since this is achievable with standard techniques. Then, we have that

$$\begin{aligned}\Pr[\mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1]\leq & c\cdot\Big(\Pr[OW\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract]\\ &+\Pr[Hash\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract]\Big).\end{aligned}$$

Now, if $OW\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract$ happens, we know that $\mathcal{A}_{\mathsf{OWF}}$ has some k_0^*,k_1^*,s^*, such that the s^* was honestly generated for values $y_0^{(j)},y_1^{(j)}$, where j is the user the signature output by $\mathcal{A}$ opens to. Furthermore, $\mathcal{A}_{\mathsf{OWF}}$ knows some k_0^*,k_1^*, such that $f(k_0^*,0^\lambda)=y_0$ and $f(k_1^*,0^\lambda)=y_1$. Thus, if $\mathcal{A}_{\mathsf{OWF}}$ correctly and independently guessed $i=j$, he computed a pre-image k_1^* for $f(k_1^*,0^\lambda)=y_1^{(i)}=y^*$. Therefore, we have that

$$\begin{aligned}&\Pr[OW\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract]\\ &=\ell\cdot\Pr[k^*\leftarrow\mathsf{PRF.Kg}(1^\lambda),x^*\leftarrow\mathcal{A}_{\mathsf{OWF}}(1^\lambda,f(k^*,0^\lambda)):f(k^*,0^\lambda)=f(x^*,0^\lambda)].\end{aligned}$$

In the case of $Hash\wedge \mathsf{Exp}^{trace}_{\Pi,\mathcal{A}}(\lambda,\ell)=1\wedge Extract$, we can make another case distinction. If we know that s^* matches the Merkle tree and either

$$f(k_0^*,0^\lambda)\neq f(k_0^{(i)},0^\lambda)\text{ or }f(k_1^*,0^\lambda)\neq f(k_1^{(i)},0^\lambda),$$

we have that $(f(k_0^*,0^\lambda),f(k_1^*,0^\lambda))$ is a pre-image of the hash belonging to user i. Thus, we know that

$$H(f(k_0^*,0^\lambda),f(k_1^*,0^\lambda))=H(f(k_0^{(i)},0^\lambda),f(k_1^{(i)},0^\lambda)),$$

but $f(k_0^*,0^\lambda),f(k_1^*,0^\lambda)\neq f(k_0^{(i)},0^\lambda),f(k_1^{(i)},0^\lambda)$. Therefore, $\mathcal{A}_{\mathcal{H}}$ found a collision. In the other case, where s^* does not match the Merkle tree, we can compute a hash collision by Lemma 3.41.

Therefore, we have that

$$\Pr[\mathit{Hash} \wedge \mathsf{Exp}^{\mathit{trace}}_{\Pi,\mathcal{A}}(\lambda,\ell) = 1 \wedge \mathit{Extract}] = \\ \Pr[k \leftarrow \mathcal{H}.\mathsf{Kg}(1^\lambda), (x_1,x_2) \leftarrow \mathcal{A}_{\mathcal{H}}(1^\lambda,k) : H_k(x_1) = H_k(x_2)].$$

Taking all the cases together, we thus have

$$\Pr[\mathsf{Exp}^{\mathit{trace}}_{\Pi,\mathcal{A}}(\lambda,\ell) = 1] \leq \\ c \cdot \Big(\ell \cdot \Pr[k^* \leftarrow [\mathsf{PRF}.\mathsf{Kg}(1^\lambda)], x^* \leftarrow \mathcal{A}_{\mathsf{OWF}}(1^\lambda, f(k^*,0^\lambda)) : f(k^*,0^\lambda) = f(x^*,0^\lambda)] \\ + \Pr[k \leftarrow \mathcal{H}.\mathsf{Kg}(1^\lambda), (x_1,x_2) \leftarrow \mathcal{A}_{\mathcal{H}}(1^\lambda,k) : H_k(x_1) = H_k(x_2)]\Big),$$

which contradicts the assumption that $\mathcal{A}$ has non-negligible advantage. □

Lemma 4.14 *If the protocol from Construction 4.7 is zero-knowledge and* PRF *is a pseudo-random function, then Construction 4.10 offers weak anonymity.*

Proof. Let $\mathcal{S}$ be the zero-knowledge simulator of the proof of knowledge in the group signature. We prove anonymity by a series of games.

- G_0: the original anonymity game.
- G_1: the same as G_0, except proofs π are generated by $\mathcal{S}$.
- G_2: the same as G_1, except $f(k_0^{(1)}, \cdot)$ is replaced by a truly random function.

  ...
- $\mathsf{G}_{\ell+1}$: the same as G_ℓ, except $f(k_0^{(\ell)}, \cdot)$ is replaced by a truly random function.

Now we need to argue, why views of a ppt adversary in consecutive games are computationally indistinguishable. Since $\mathcal{S}$ outputs proofs with the same distribution as honest provers by the zero-knowledge property of the NIZKPoK, games G_0 and G_1 are perfectly indistinguishable. If we assume that G_1 and G_2 are distinguishable by some ppt $\mathcal{A}$, we can construct a decider that is able to break the pseudo-randomness of PRF. That decider simply plays the role of the challenger in G_2, except whenever $f(k_0^{(0)}, x)$ would be evaluated, the decider uses the value $\mathcal{O}(x)$ of his oracle instead.Then, it outputs whatever $\mathcal{A}$ outputs. Thus, G_1 is computationally indistinguishable from G_2. By a similar argument, we can argue that for all $i \in \{2,..,\ell\}$ the games G_i and G_{i+1} are computationally indistinguishable.

Let $\mathsf{negl}(\cdot)$ be an upper bound of an adversary against the pseudo-randomness of PRF. Since in $\mathsf{G}_{\ell+1}$ we have that all signatures $\sigma = (\pi, y)$ are generated by a simulator and a truly random function, their distribution does not depend on b. Therefore, by these two facts and the sequence of games we know that

$$\Pr[b' = b | b = 0] - \Pr[b' = b | b = 1] \leq \ell \cdot \mathsf{negl}(\lambda).$$

Thus, any adversary wins the anonymity game with only negligible probability. □

Then, the following theorem follows directly from Lemmata 4.13 and 4.14.

Theorem 4.15 *If the protocol from Construction 4.7 is a NIZKPoK, $\mathcal{H}$ is collision resistant and* PRF *is a pseudo-random function that is also a one-way function for any arbitrary but fixed input, then Construction 4.10 is correct and offers traceability and weak anonymity.*

Although this theorem only considers classical adversaries, we can replace the non-interactive proof system by Construction 4.8. Then, since we conjecture that symmetric-key primitives are post-quantum secure, we deduce that Construction 4.10 is post-quantum secure.

4.1.4 Efficiency

The non-interactive proof system from Construction 4.7 as it is defined is not very efficient. One reason is that in the simulated MPC protocol there are many messages being sent and each such message contributes to the size of the proof. Furthermore, the many repetitions of the Σ-protocol to get a exponentially large challenge space also increases the proof size. However, Katz et al. present several steps to optimize their construction, with which the size of the proof can be greatly reduced.

In their paper [10], Katz et al. not only construct a group signature scheme based on their proof of knowledge, but also construct a digital signature scheme with a similar idea. Due to the optimizations mentioned above, they show evaluations supporting their claim that their digital signature scheme is competitive in signature size and verification time compared to the currently best known alternatives. Furthermore, they show that the proof of knowledge they use for the signature is the most efficient compared to two constructions of other papers, but only for circuits that have between 300 and 100000 AND-gates. Since the main difference between their digital signature scheme and their group signature scheme is the relation that is used for the proof of knowledge, they expect that their group signature scheme is very efficient as well. In fact, they claim their construction is the most efficient currently known one.

While the proof size is expected to be small in comparison (due to the evaluations from before), the technique by which Katz et al.'s scheme offers traceability leads to drawbacks. First, their construction has a fixed size, meaning it is not easy to add new members to the group beyond the old size without invalidating old signatures. But creating a very large group in the beginning is also not a good solution. This is due to the fact that both the runtime of the signing algorithm and the verification algorithm have a runtime of $\mathcal{O}(\log n)$, where n is the size of the group.

Another drawback of their construction is that opening a signature takes time $\mathcal{O}(n)$, as the group manager has to iterate through all group members to find the correct one. This is obviously worse than in the standard construction of sign,

encrypt and proof (cf. Chapter 1), where the group size has no influence on the running time of the opening algorithm, as it simply decrypts.

4.2 Conclusion Katz

To understand the results of Katz et al.'s approach better, we discuss their techniques in a more abstract way.

The main point to take away from Katz et al.'s construction is the ability to construct proofs of knowledge for any circuit while using only symmetric-key primitives (cf. Theorem 4.3). With this, we are able to construct proofs of knowledge for any NP-relation, as the circuit we use for the proof is simply the algorithm verifying a witness. This gives us a very powerful and flexible tool. When constructing a group signature in a way similar to the standard technique, it allows us to use any type of certificate , as long it can be verified as valid. Furthermore, the proof of knowledge hides which certificate is used.

However, the size of the proof increases with the size of the certificate. Therefore, Katz et al. use a Merkle tree in place of the signature. A certificate for a Merkle tree, which is the validation set, has a size of $\log \ell$ hash values. Furthermore, verification is simple, as one has to make $\log \ell$ computations of the hash function, as well as one comparison. This benefits the construction of Katz et al. insofar that the size of the proof of knowledge scales with the complexity of the verification circuit. However, the drawback of using the Merkle tree is having only a fixed-size group signature scheme.

Another technique Katz et al. use is replacing the encryption scheme of the standard technique by a pseudo-random function that is also a one-way function in respect to the key. Thus, the group manager can easily trace by finding which key was used to compute the y. This is possible, since we use a proof of knowledge that works for any circuit. By also proving that we know the key that was used to compute y and linking it to the certificate, we can guarantee the verifier that we "encrypted", i.e. embed our identity, honestly. However, then the running time of the opening algorithm is linear in the number of group members, where in the standard technique it is constant. Depending on the use case of the group signature scheme, this may be a big drawback or a little one. If the tracing algorithm is used only rarely, an expensive opening algorithm does not have a big impact.

4.3 Extension to a Dynamic Group Signature

As mentioned before, Construction 4.10 is only a static group signature. However, with techniques similar to the previous construction, one can extend the idea to a dynamic group signature. In particular, we replace the Merkle tree that was used for authenticity with a signature scheme as in the standard group signature

construction. Since the proof of knowledge of Katz et al. allows us to create proofs for any NP-relation, we can use a standard technique to construct a digital signature. For that, we define the following circuit.

Construction 4.16 *Let $\mathcal{F} = (\mathcal{F}.\mathsf{Kg}, f)$ be a keyed function. Define the circuit C^{sig}_{pk} that on input $\mathsf{sk} \in [\mathcal{F}.\mathsf{Kg}(1^\lambda)]$ computes $f_{\mathsf{sk}}(0^\lambda)$ and checks whether this value is equal to pk. If it is, the circuit outputs 1, else 0.*

With this circuit, we can then formally define the digital signature.

Construction 4.17 *Let $\mathcal{F} = (\mathcal{F}.\mathsf{Kg}, f)$ be a keyed function.*

- $\mathsf{Kg}(1^\lambda)$*: Choose* $\mathsf{sk} \leftarrow \mathcal{F}.\mathsf{Kg}(1^\lambda)$*. Compute* $\mathsf{pk} = f(\mathsf{sk}, 0^\lambda)$*. Output* $(\mathsf{sk}, \mathsf{pk})$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$*: Use the prover of Construction 4.7 to create a non-interactive proof σ for the circuit C^{sig}_{pk} from Construction 4.16 with input* sk*. When computing the challenge with the Fiat-Shamir transform, also append m to the input of the hash function. Output σ.*
- $\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$*: Use the verifier of Construction 4.7 to verify whether σ is valid for the circuit C^{sig}_{pk}. When computing the challenge with the Fiat-Shamir transform, also append m to the input of the hash function. Output whatever the verifier outputs.*

The intuition behind this construction is that to sign a message, we prove ownership of the function key used to compute the public key. To make the signature dependent on the message, we include it when generating the challenge for the underlying Σ-protocol.

Lemma 4.18 *If $\mathcal{F}$ is a one-way function with regards to the key for inputs of form 0^λ, then Construction 4.17 is existentially unforgeable under chosen message attack.*

We omit the proof for this lemma, as the technique of using a non-interactive proof of knowledge in this way is commonly used. Furthermore, Katz et al. [10] and Chase et al. [5] claim similar results.

With this digital signature, we can continue with our original goal of constructing a dynamic group signature. For that, we again first define a circuit. Since our goal was to replace the Merkle tree with digital signature scheme, this circuit checks whether a signature is valid. As in Construction 4.9, the circuit also checks whether the given x, y, k_0 fulfill $y = f(k_0, x)$ for some function f.

Construction 4.19 *Let $\Pi' = (\mathsf{Kg}', \mathsf{Sign}', \mathsf{Vrfy}')$ be the digital signature from Construction 4.17. Let $f : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a function. Define the circuit $C_{x,y,\mathsf{pk}}$ that on input (k_0, k_1, σ') outputs 1 if $y = f(k_0, x)$ and $\mathsf{Vrfy}'(\mathsf{pk}, (k_0 || f(k_1, 0^\lambda)), \sigma') = 1$. Else it outputs 0.*

Then, we can use circuit in the construction of the group signature scheme. The signing works similar to Construction 4.10: we compute the image of a pseudo-random function with the k_0 key and include a proof of knowledge for a circuit, but this time the circuit from Construction 4.19. Thus, the very algorithm also

works very similar, as it has to check for validity of the proof. Since the new construction is a dynamic group signature, instead of being static, we have to adjust the key generation a bit and construct a joining protocol.

Construction 4.20 *Let* $\Pi' = (\mathsf{Kg}', \mathsf{Sign}', \mathsf{Vrfy}')$ *be the digital signature from Construction 4.17. Let* $\mathcal{F} = (\mathcal{F}.\mathsf{Kg}, f)$ *be a keyed function. Construct a dynamic group signature scheme.*

- $\mathsf{Setup}(1^\lambda)$*: Generate* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}'(1^\lambda)$*. Set* $\mathsf{gpk} = \mathsf{pk}$ *as the public key,* $\mathsf{isk} = \mathsf{sk}$ *as issuer secret key and* $\mathsf{osk} = \emptyset$ *as the tracing key. Output* $(\mathsf{gpk}, \mathsf{isk}, \mathsf{osk})$*.*
- $\mathsf{UKg}(1^\lambda, \mathsf{gpk})$*: Generate two keys* $k_0, k_1 \leftarrow \mathcal{F} : \mathsf{Kg}(1^\lambda)$ *and compute* $y_1 = f(k_1, 0^\lambda)$*. Set* $\mathsf{usk} = (k_0, k_1)$*,* $\mathsf{upk} = y_1$*.*
- $\mathsf{Join}(\mathsf{usk}_i) \leftrightarrow \mathsf{Issue}(\mathsf{upk}_i, \mathsf{isk})$*: The user parses* $(k_0, k_1) = \mathsf{usk}_i$*. He then sends* k_0 *as well as a NIZKPoK that he knows* k_1 *such that* $f(k_1, 0^\lambda) = \mathsf{upk}_i$ *to the issuer. If the proof is valid, the group manager responds with* $\sigma' \leftarrow \mathsf{Sign}'(\mathsf{isk}, (k_0, \mathsf{upk}_i))$ *and sets* $\mathsf{reg}_i = k_0$*. The user outputs* $\mathsf{cert}_i = \sigma'$*.*
- $\mathsf{Sign}(\mathsf{usk}_i, \mathsf{cert}_i, m)$*: Parse* $\mathsf{usk}_i = (k_0, k_1, \sigma')$*. Compute* $y = f(k_0, m)$*. Use the prover of Construction 4.7 to create a non-interactive proof* π *for the circuit* $C_{m,y,\mathsf{gpk}}$ *with input* $(k_0, k_1, \mathsf{cert}_i)$*, where the circuit is defined as in Construction 4.19. Output* $\sigma = (\pi, y)$*.*
- $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma)$*: Parse* $\sigma = (\pi, y)$*. Let* $C_{m,y,\mathsf{gpk}}$ *be the circuit from Construction 4.19. Use the verifier of Construction 4.7 to check whether the proof* π *is valid for that circuit. If it is, output 1. Else, output 0.*
- $\mathsf{Open}(\mathsf{osk}, m, \sigma)$*: Parse* $\sigma = (\pi', y)$*. If* $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma) = 1$*, check for each* $\mathsf{reg}_i \neq \perp$ *in* reg *whether* $f(\mathsf{reg}_i, m) = y$*. Let* reg_j *be the first such key. Compute a NIKZPoK* π *for knowing* reg_j *such that* $f(\mathsf{reg}_j, m) = y$*. Output* (j, π)*.*
- $\mathsf{Judge}(\mathsf{gpk}, j, \mathsf{upk}_j, m, \sigma, \pi)$*: Parse* $\sigma = (\pi', y)$*. If* π *is valid for the circuit* $f(\cdot, m) = y$*, output 1, else 0.*

We construct the mentioned NIZKPoKs with Construction 4.7.

As always, we are interested in the security of such constructions. However, due to time constraints, we cannot give a full proof. Instead, we state a conjecture and give a short argument, why the conjecture should hold.

Conjecture 4.21 *If* Π' *is a EUF-CMA secure signature scheme,* $\mathcal{F} = (\mathcal{F}.\mathsf{Kg}, f)$ *is pseudo-random function and* $f(\cdot, x)$ *is a one-way function for all inputs* x*, then Construction 4.20 is a dynamic group signature scheme that is correct and offers anonymity and traceability.*

It can easily be seen that correctness and anonymity should hold. The former follows from the correctness of the digital signature scheme and the fact that an honest signature contains a $y = f(k_0^{(i)}, m)$ for some $k_0^{(i)}$ which is contained in osk,

thus the opening manager can output the corresponding i, similar to the proof of Lemma 4.11.

The proof for anonymity works very similar to the proof of Lemma 4.14, since in both the construction of Katz et al. and the new construction, the signatures consist of a proof of knowledge and the image of a pseudo-random function. Thus, to write a formal proof, one simply has to adapt the proof of Lemma 4.14 to the dynamic group signature definition.

To win the traceability game, an adversary has to produce a signature that is valid, but opens to nobody, or the proof output by Open has to be invalid. To ensure that $\mathsf{Open}(\mathsf{osk}, m^*, \sigma^*) = \perp$, the adversary has to find a key k^*, such that there exists no key $k_0^{(i)}$ among all users, such that $f(k^*, m^*) = f(k_0^{(i)}, m^*)$. While this alone is easy, the adversary also has to produce a valid proof that uses k^* as a witness. Thus, he either has to use a honestly generated cert_i, but then he also needs that $f(k^*, 0^\lambda) = f(k_0^{(i)}, 0^\lambda)$. However, this is hard for him, as $f(\cdot, 0^\lambda)$ is a one-way function. The other possibility is that the adversary creates his own certificate with which he creates the proof. This also hard for him, as the digital signature scheme is unforgeable. Therefore, Open finds a k' in the registry, such that $f(k', m) = y$ and computes a valid proof. Thus, our construction is traceable.

An adversary can win the non-frameability game in two ways. Either he forges a signature that opens to an honest user or he creates a proof that attests that some signature opens to an honest user. For the second possibility, an adversary has to find a k^*, such that $f(k^*, m) = y$, where m, y are from the signature that he outputs. Then, he can claim that any user signed the message by creating the proof. But finding such a k^* is hard, as $f(\cdot, m)$ is a one-way function. On the other hand, an adversary has to guess the user secret key k_1 of an honest user to forge a signature, else he cannot create the proof needed for the signature. However, since $f(\cdot, 0^\lambda)$ is a one-way function this is hard.

5 Lattice Based Group Signatures

Another approach to construct quantum-secure cryptographic primitives is using lattices and related problems. This is advantageous, as lattices require only multiplication and addition of matrices and vectors to be implemented. Furthermore, progress in solving lattice problems is slow and currently there is no known algorithm for solving them efficiently for useful parameters (cf. [15]).

This chapter is structured the following way. First, we formally explain what lattices and related problems are. Then, we define a collection of constructions whose security is based on lattice problems. After that, we use those constructions as building blocks to construct the group signature itself and explain its security. At last, we discuss the techniques of the group signature.

5.1 Basic Definitions for Lattice-Based Cryptography

A lattice is similar to a vector space, in that for each lattice there exists a basis. Elements of the lattice are then linear combinations of the basis vectors, however all coefficients must be integers. In the two-dimensional space, this creates a structure that resembles a lattice if lattice points are connected by the basis vectors, thus the name.

Definition 5.1 ([15]) *A lattice* $\mathcal{L}(\mathbf{b}_1, ., , \mathbf{b}_n)$ *is defined by n linearly independent vectors* $\mathbf{b}_1, .., \mathbf{b}_n \in \mathbb{R}^n$ *as* $\mathcal{L}(\mathbf{b}_1, ., , \mathbf{b}_n) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$. *We can alternatively write* $\mathcal{L}(\mathbf{B})$*, if we set* $\mathbf{B}$ *to the matrix with the* $\mathbf{b}_i$ *as columns. We call* $\mathbf{b}_1, .., \mathbf{b}_n$ *or* $\mathbf{B}$ *a basis for lattice* $\mathcal{L}(\mathbf{B})$.

It is important to note that a lattice does not possess a unique basis. In fact, it is possible to compute a different basis for the same lattice by computing $\mathbf{B}' = \mathbf{BU}$, where $U \in \mathbb{Z}^{n \times n}$ with $det(\mathbf{U}) = \pm 1$.

One central computational problem of lattices that is important for cryptography is finding a short vector given a lattice basis. In fact, the adversary needs to output a vector whose length is smaller than the length of the shortest vector times some factor.

Definition 5.2 *In the* γ-approximate shortest vector problem (SVP_γ)*, an adversary* $\mathcal{A}$ *given a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}(\mathbf{B})$ *has to find a non-zero vector* $\mathbf{v}$*, such that* $\|\mathbf{v}\| \leq \|\mathbf{v}_0\| \cdot \gamma$*, where* $\mathbf{v}_0 \in \mathcal{L}(B)$ *is the shortest non-zero vector of* $\mathcal{L}(\mathbf{B})$.

Note that the adversary is given an arbitrary basis. This means he needs to be able to solve any instance, even the hardest, if he wants to solve the problem.

There are many approaches to solving this problem, for example the LLL-algorithm [11]. While this algorithm is one of the best currently known algorithms, it has exponential runtime if γ is a polynomial in n. Conversely, if we restrict the algorithm to a polynomial runtime, it only guarantees to find a vector with length within an exponential factor of the shortest one. Due to this and slow progress in finding better algorithms (cf. [15]), the SVP_γ problem is conjectured to be hard.

Conjecture 5.3 ([15]) *Any* quantum *ppt adversary $\mathcal{A}$ has only negligible success probability of solving SVP_γ, if γ is some polynomial in n, where n is the dimension of the lattice.*

It is important to note that this conjecture holds even for quantum adversaries. This means that a scheme that bases its security on lattice problems, such as SVP_γ, is considered to be post-quantum secure.

The SVP mentioned above is an approximation problem. There are several variants of this problem (and of other lattice problems). For example, there exists a decision variant, where an adversary given a lattice basis and some rational number has to decide whether the norm of the shortest non-zero vector of the lattice is smaller than the given number or not. Another variation is a relaxation of the decisional variant, a so-called promise problem. In a promise problem, an adversary is given an input that either satisfies a Yes condition or a No condition. The conditions need to be mutually exclusive, but in difference to the decisional variant, the conditions need not be exhaustive, i.e. there may exist instances that are neither a Yes or No instance. An adversary then has to decide, whether the input satisfies the Yes or No condition. For inputs that satisfy neither condition, there are no requirements for what the adversary outputs. Thus, formally the promise version of SVP_γ looks as follows.

Definition 5.4 ([12]) *In the* shortest vector promise problem (GapSVP_γ)*, an adversary $\mathcal{A}$, given a basis $\mathbf{B}$ of a lattice $\mathcal{L}(\mathbf{B})$ and a $d \in \mathbb{Q}$, has to decide between* Yes *inputs with $\|\mathbf{v}_0\| \leq d$ and* No *inputs with $\|\mathbf{v}_0\| > \gamma \cdot d$, where $\mathbf{v}_0 \in \mathcal{L}(B)$ is the shortest non-zero vector of $\mathcal{L}(\mathbf{B})$.*

While one can base the security of cryptographic constructions on SVP or GapSVP, there exists another lattice-related problem that is useful. The problem, called *shortest integer solution* problem, deals with q-ary lattices.

Definition 5.5 *A q-ary lattice $\Lambda_q^\perp(\mathbf{A})$ is defined by a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ as $\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}^m : \mathbf{A}\mathbf{y} = 0 \mod q\}$.*

The SIS problem then asks for a short non-zero vector that belongs to the lattice. As the problem is an average-case problem, the adversary gets a uniformly random lattice. Formally, it is defined the following way.

Definition 5.6 ([12]) *In the* shortest integer solution *problem* ($\mathsf{SIS}_{q,m,\beta}$) *an adversary, given an integer q, a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\beta \in \mathbb{R}$, has to find a non-zero vector $\mathbf{z} \in \mathbb{Z}^m \setminus \{0\}$ such that $\mathbf{A}\mathbf{z} = 0 \mod q$ and $\|\mathbf{z}\| \leq \beta$.*

There exists a similar problem to SIS, called SIS'. This differs from SIS in so far that for the output $\mathbf{z}$ of the adversary it additionally has to hold that $\mathbf{z} \in \mathbb{Z}^m \setminus 2\mathbb{Z}^m$.

Although SIS' is an average-case problem, there exists a reduction to the worst-case problem SVP.

Theorem 5.7 ([12]) *There exist parameters β, m, q, γ such that there is a polynomial time reduction from solving GapSVP_γ in the worst case to solving $\mathsf{SIS}'_{q,m,\beta}$ in the average case.*

This also means that the reduction reduces to SIS if q is odd.

The existence of such a reduction implies that solving SIS in the average case is at least as hard as solving SVP in the worst case. This in turn means that we can base the security of our constructions on the SIS problem instead, which is advantageous since q-ary lattices are easier to use and implement, as they only use integer arithmetic.

There exists another problem related to lattices, where an adversary has to distinguish between two different distributions. In one case, the adversary gets a random vector, while in the other case he gets a random lattice point on which an error vector is added, where the latter is drawn from a known error distribution.

Definition 5.8 ([15]) *In the* learning with errors *problem* ($\mathsf{LWE}_{m,q,\chi}$)*, that is parametrized by integers n, m, q and a probability distribution χ on $\mathbb{Z}_q$, an adversary is given a uniformly chosen matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ as well as a vector $\mathbf{v}$. This vector is either chosen uniformly at random from $\mathbb{Z}_q^m$ or set to be $\mathbf{v} = \mathbf{As} + \mathbf{e}$, where $s \leftarrow \mathbb{Z}_q^n$ is chosen uniformly at random and $\mathbf{e} \leftarrow \chi^m$. The adversary then has to distinguish between those two cases.*

If we use the truncated m-dimensional continuous Gaussian distribution as the error distribution, we can relate LWE to GapSVP.

Definition 5.9 *Let $D_s(x) = \frac{1}{s} \cdot \exp(-\pi(\frac{x}{s})^2)$ be the density function of the truncated one-dimensional continuous Gaussian distribution over $\mathbb{R}$, where $s \in \mathbb{R}^+$ and $x \in \mathbb{R}$ with $|x| < s \cdot \omega(\sqrt{\log n})$.*

Definition 5.10 ([8]) *The distribution Ψ_α^m over $[0, q)$ is defined by the following algorithm: Choose $\eta_1, .., \eta_m \leftarrow D_\alpha$, set $e_i = q \cdot \eta_i \mod q$ for $i \in [m]$, and output $\mathbf{e} = (e_1, .., e_m)^T$.*

Formally, the relation between LWE and GapSVP can be expressed by the following reduction.

Theorem 5.11 ([14]) *There exist parameters $\alpha, \gamma, \zeta, q, m$ such that there is a polynomial time reduction from solving $\mathsf{GapSVP}_{\zeta,\gamma}$ in the worst case (with overwhelming probability) to solving $\mathsf{LWE}_{m,q,\Psi_\alpha^m}$.*

Note that in this theorem, GapSVP has an additional parameter ζ. The definition of $\mathsf{GapSVP}_{\zeta,\gamma}$ is very similar to GapSVP_γ, but has some additional restrictions on which bases $\mathbf{B}$ can be used for the lattice. Still, this theorem implies that the LWE problem is hard if GapSVP is hard.

5.2 The Group Signature of Gordon et al.

The group signature scheme of Gordon et al. [8] follows the standard construction of a group signature (cf. Chapter 1) which consists of a signature scheme, an encryption scheme and a proof of knowledge linking both together. Furthermore, they base the security of these building blocks on lattice assumptions, thus the group signature scheme is secure if these assumptions hold.

In the following, we first define the building blocks Gordon et al. used. After that, we present their group signature scheme and explain their techniques.

5.2.1 Building Blocks

A very important building block is an algorithm called TrapSample. It outputs two matrices $\mathbf{A}$ and $\mathbf{T}$. The idea behind this algorithm that it is hard for an adversary given $\mathbf{A}$ and a vector $\mathbf{v}$ to win the LWE game. However, if one also knows $\mathbf{T}$, then it is easy to distinguish whether $\mathbf{v}$ is a uniformly random vector or a perturbed lattice point. In fact, if $\mathbf{v} = \mathbf{As} + \mathbf{e}$, one can even compute $\mathbf{s}$ and $\mathbf{e}$.

Lemma 5.12 ([1]) *There exists a ppt algorithm* TrapSample*, that on input* $1^n, 1^m, q$ *outputs matrices* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ *and* $\mathbf{T} \in \mathbb{Z}^{m \times m}$*, such that*

- *the distribution of* $\mathbf{A}$ *is statistically indistinguishable from that of a uniformly random matrix from* $\mathbb{Z}^{n \times m}$*,*
- *the columns of* $\mathbf{T}$ *form a basis of* $\Lambda^{\perp}(\mathbf{A})$*,*
- $\|\mathbf{T}\| = \mathcal{O}(n \log q)$ *and* $\left\|\tilde{\mathbf{T}}\right\| \leq C \cdot \sqrt{n \log q}$*,*

where $q \geq 2, m \geq 8n \log q$ *and* $C < 40$ *is some constant.*

This idea works, since $\mathbf{T}$ consists only of short, orthogonal column vectors. Then, one can use Babai's rounding algorithm [15], which on input $\mathbf{T}$ and $\mathbf{v} = \mathbf{As}+\mathbf{e}$ is defined as $\mathbf{T}\lfloor \mathbf{T}^{-1}\mathbf{v} \rceil$, to compute $\mathbf{s}$. One can use TrapSample and Babai's rounding algorithm to construct an encryption scheme, where a ciphertext is $\mathbf{v} = \mathbf{As}+\mathbf{e}$ with $\mathbf{s} = G(m)$ modeled as a random oracle [12], where G somehow encodes m to a vector. Decrypting is then using Babai's algorithm.

With the knowledge of a $\mathbf{T}$ output by TrapSample, it is possible to construct another interesting algorithm, called PreSample. This algorithm is able to invert a lattice point, i.e. given an $\mathbf{A}$ and $\mathbf{u}$ is able to compute an $\mathbf{e}$ such that $\mathbf{Ae} = \mathbf{u}$.

Lemma 5.13 *There exists a ppt algorithm* PreSample*, that on input* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*,* $\mathbf{T} \in \mathbb{Z}^{m \times m}$ *that were output by* TrapSample *and some* s *as well as* $\mathbf{u} \in \mathbb{Z}_q^n$ *outputs a vector* $\mathbf{e} \in \mathbb{Z}_q^m$*. This vector is distributed uniformly at random conditioned on* $\mathbf{Ae} = \mathbf{u}$*.*

We can then combine TrapSample and PreSample to create a signature scheme. The idea is to choose a public key $\mathbf{A}$ and secret key $\mathbf{T}$ with TrapSample. Then, the signer can sign a message by computing a pre-image of the message with

PreSample. Security relies on the fact that, assuming SIS is hard, it is hard for an adversary to compute such a pre-image without knowing $\mathbf{T}$.

Construction 5.14 ([7]) *Let $q = \mathsf{poly}(n)$ be prime. Let $m \geq 5n \log q$. Let $H : \{0,1\}^* \to \mathbb{Z}_q^n$ be a random oracle. The GPV signature scheme then is defined as follows.*

- $\mathsf{Kg}(1^\lambda)$*: Compute* $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSample}(1^\lambda, 1^m, q)$*. Set* $\mathsf{sk} = \mathbf{T}, \mathsf{pk} = \mathbf{A}$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$*: Output* $\mathbf{e} \leftarrow \mathsf{PreSamp}(\mathbf{A}, \mathbf{T}, H(m))$*.*
- $\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$*: If* $\mathbf{Ae} = H(m) \mod q$ *and* $\|\mathbf{e}\| \leq s\sqrt{m}$*, return 1, else 0.*

Lemma 5.15 ([7]) *If $\mathcal{H}$ is modeled as a random oracle and $\mathsf{SIS}_{q,m,2s\sqrt{m}}$ is hard, then Construction 5.14 is a EUF-CMA secure signature scheme.*

Another building block that Gordon et al. use is the algorithm OrthoSample. This algorithm has a similar output to TrapSample, but it gets as additional input a matrix $\mathbf{B}$. The matrix $\mathbf{A}$ output by the algorithm then guarantees that $\mathbf{AB}^T = \mathbf{0} \mod q$.

Lemma 5.16 ([8]) *There exists a ppt algorithm* OrthoSample*, that on input $1^n, 1^m, q$ and a matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$ outputs matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{T} \in \mathbb{Z}^{n \times m}$, such that*

- *the distribution of $\mathbf{A}$ is statistically indistinguishable from that of a uniformly random matrix from $\mathbb{Z}^{n \times m}$ conditioned on $\mathbf{AB}^T = \mathbf{0} \mod q$*
- *the columns of $\mathbf{T}$ form a basis of $\Lambda^\perp(\mathbf{A})$*
- *each column of $\mathbf{T}$ is distributed independently according to $D_{\Lambda^\perp(\mathbf{A}),s}$*

where $q \geq 2, m \geq n + 8n \log q$, $C < 40$ some constant, $s = C \cdot \sqrt{n \log q} \cdot \omega(\log m)$ and $D_{\Lambda^\perp(\mathbf{A}),s}$ is D_s with support restricted to $\Lambda^\perp(\mathbf{A})$.

Gordon et al. further prove that it is possible to generate $(\mathbf{B}, \mathbf{S}) \leftarrow \mathsf{TrapSample}$ and $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{OrthoSample}(\mathbf{B})$ and use $\mathbf{A}, \mathbf{T}$ as the keys for the GPV signature scheme. Then, we can use $\mathbf{B}, \mathbf{S}$ as the keys for our encryption scheme component. Thus, the signature scheme and encryption scheme used to construct the group signature scheme are already related, since $\mathbf{AB}^T = \mathbf{0}$.

However, we still need a proof of knowledge to ensure that a signer encrypted a valid signature. For this, Gordon et al. define a gap language and use results from other work [13, 6] to construct the proof for this language.

Definition 5.17 *The language $L_{s,\gamma}$ is a gap language that is defined by L_{Yes} and L_{No} as follows:*

$$L_{\mathsf{Yes}} = \left\{ \begin{pmatrix} \mathbf{B}_1, .., \mathbf{B}_N \\ \mathbf{z}_1, .., \mathbf{z}_N \end{pmatrix} \middle| \exists \mathbf{c} \in \mathbb{Z}_q^n \ \exists i \in [N] : \left\| \mathbf{z}_i - \mathbf{B}_i^T \mathbf{c} \right\| \leq s\sqrt{m} \right\}$$

$$L_{\mathsf{No}} = \left\{ \begin{pmatrix} \mathbf{B}_1, .., \mathbf{B}_N \\ \mathbf{z}_1, .., \mathbf{z}_N \end{pmatrix} \middle| \forall \mathbf{c} \in \mathbb{Z}_q^n \ \forall i \in [N] : \left\| \mathbf{z}_i - \mathbf{B}_i^T \mathbf{c} \right\| > \gamma \cdot s\sqrt{m} \right\}$$

A witness for $L_{s,\gamma}$ is an s and an i satisfying a Yes *instance.*

Since this is a gap language, a proof of knowledge does not need to guarantee anything for instances that are neither a Yes instance or No instance. Furthermore, Gordon et al. do not need a zero-knowledge proof of knowledge. Instead, it suffices to have a witness-indistinguishable proof. This means that the distribution of the proof is independent from the witness used.

Lemma 5.18 ([8]) *There is a non-interactive witness-indistinguishable proof system for the language $L_{c,\gamma}$ in the random oracle model, where $\gamma \geq \mathcal{O}(\sqrt{\frac{m}{\log m}})$.*

It is important to note that in their construction, Gordon et al. use the Fiat-Shamir transform to get a non-interactive proof system from a Σ-protocol. As we have mentioned before (cf. Section 3.10), there exist results that lead to believe that the Fiat-Shamir transform produces proof systems that are not secure against a quantum adversary. Therefore, although the lattice problems are conjectured to be post-quantum secure, a construction using the Fiat-Shamir transform may not be post-quantum secure due to the transform. However, we propose that if we use Unruh's transform instead, the proof system mentioned above is also post-quantum secure.

5.2.2 Construction

With these building blocks defined, we can construct the static group signature scheme. We generate our keys with TrapSample and OrthoSample to set up an encryption component and signature scheme respectively, one for each group member. Then, to sign a message a group member i uses the digital signature scheme to sign the hash of the message, i.e. $\mathbf{e}_i \leftarrow \mathsf{PreSample}(\mathbf{A}_i, \mathbf{T}_i, H(m))$, and encrypts it by computing $\mathbf{z}_i = \mathbf{B}_i^T\mathbf{w} + \mathbf{e}_i$ for a uniformly random $\mathbf{w}$. At this point, the group signature does not offer anonymity, as the signature would be valid only for the public key that belongs to the correct signer. To circumvent this issue, the signer also creates dummy signatures, one for each other group member. This dummy signature is simply an $\mathbf{e}_j$ that satisfies $\mathbf{A}_j\mathbf{e}_j = H(m)$. For such a dummy signature to be valid, $\mathbf{e}_j$ would have to be short, however computing such a vector without length restriction is not hard. The group member i then encrypts the $\mathbf{e}_j$ as well. He also prepares a proof that one of the $\mathbf{e}_k$ is short. This means that at least one of the $\mathbf{e}_k$ is a valid signatures is valid, but an adversary cannot check which.

To verify a signature, one needs to check whether the proof is valid and whether $\mathbf{A}_k\mathbf{z}_k = H(m)$. This way, the verifier checks whether each dummy signature fulfills the necessary condition that $\mathbf{A}_k\mathbf{e}_k = H(m)$ and that there exists a real signature. Note that a verifier can check the necessary condition in this way, since we have that $\mathbf{A}\mathbf{B}^T = \mathbf{0}$. To open a group signature, the opening manager simply decrypts the $\mathbf{z}_k$ and outputs the index for which the decrypted vector is small. By construction, this is true for the real signature, but with high probability not for the dummy signatures.

Formally, the construction looks as follows.

Construction 5.19 *Let λ be the security parameter. Let $q = \mathsf{poly}(\lambda)$, $m \geq 8\lambda \log q, s \geq C\sqrt{\lambda \log q} \cdot \omega(\sqrt{\log m})$. Let $\mathcal{H} = (\mathcal{H}.\mathsf{Kg}, H)$ be a hash function with $H : \{0,1\}^* \to \mathbb{Z}_q^n$.*

- $\mathsf{Kg}(1^\lambda, 1^\ell)$*: Compute* $(\mathbf{B}_1, \mathbf{S}_1), .., (\mathbf{B}_\ell, \mathbf{S}_\ell) \leftarrow \mathsf{TrapSample}(1^\lambda, 1^m, q)$ *as well as* $(\mathbf{A}_i, \mathbf{T}_i) \leftarrow \mathsf{OrthoSample}(1^\lambda, 1^m, q, \mathbf{B}_i)$*. Compute* $k_{hash} \leftarrow \mathcal{H}.\mathsf{Kg}(1^\lambda)$*. Let* $\mathsf{gpk} = ((\mathbf{A}_i, \mathbf{B}_i)_{i=1}^\ell, k_{hash})$*. Let* $\mathsf{msk} = (\mathbf{S}_i)_{i=1}^\ell$ *and* $\mathsf{usk}_i = \mathbf{T}_i$*. Output* $\mathsf{gpk}, \mathsf{msk}, (\mathsf{usk}_i)_{i=1}^\ell$*.*

- $\mathsf{Sign}(\mathsf{usk}_i, m)$*: Choose* $r \leftarrow \{0,1\}^\lambda$*. Compute* $\mathbf{h}_j = H_{k_{hash}}(m||r||j)$ *for* $j \in [\ell]$*. Compute* $\mathbf{e}_i \leftarrow \mathsf{PreSample}(\mathbf{A}_i, \mathbf{T}_i, s, \mathbf{h}_i)$*. For* $j \in [\ell] \backslash \{i\}$ *choose* $e_j \in \mathbb{Z}_q^m$ *uniformly at random conditioned on* $\mathbf{A}_j \mathbf{e}_j = \mathbf{h}_j \mod q$*. For* $k \in [\ell]$*, choose* $\mathbf{w}_k \leftarrow \mathbb{Z}_q^n$ *and compute* $\mathbf{z}_k = \mathbf{B}_k^T \mathbf{w}_k + \mathbf{e}_k \mod q$*. Construct a NIWI proof* π *for* $L_{s,\gamma}$ *with witness* $(\mathbf{w}_i, i)$*. Output* $\sigma = (r, \mathbf{z}_1, .., \mathbf{z}_\ell, \pi)$*.*

- $\mathsf{Vrfy}(\mathsf{gpk}, m, \sigma)$*: Parse* $\sigma = (r, \mathbf{z}_1, .., \mathbf{z}_\ell, \pi)$*. If* $\mathbf{A}_j \mathbf{z}_j = H_{k_{hash}}(m||r||j)$ *for all* $j \in [\ell]$ *and* π *is correct, output 1, else 0.*

- $\mathsf{Open}(\mathsf{msk}, m, \sigma)$*: If* $\mathsf{Vry}(\mathsf{gpk}, m, \sigma) = 1$*, parse* $\sigma = (r, \mathbf{z}_1, .., \mathbf{z}_\ell, \pi)$*. For* $j \in [\ell]$*, compute* $\mathbf{e}_j = \mathbf{z}_j - \mathbf{B}_j^T \left(\mathbf{S}_j \lfloor \mathbf{S}_j^{-1} \mathbf{z}_j \rceil\right)$*. Output the smallest* j*, for which it holds that* $\|\mathbf{e}_j\| \leq s\sqrt{m}$*.*

Lemma 5.20 *The static group signature from Construction 5.19 is correct and offers traceability and CPA-anonymity, if H is modeled as a random oracle.*

In this thesis, we do not give a full proof for this lemma, as it can be found in [8]. However, we explain the intuition why the three properties hold.

Correctness

For a honestly generated signature σ by user i, we know that $\mathbf{A}_j \mathbf{e}_j = H_{k_{hash}}(m||r||j)$ holds by construction for $j \in [\ell] \setminus \{i\}$. We also know that this equation holds for $j = i$, due to the properties of $\mathsf{PreSample}$. By definition of $\mathsf{OrthoSample}$ we know that $\mathbf{A}_j \mathbf{B}_j^T = \mathbf{0}$, thus we have that $\mathbf{A}_j \mathbf{z}_j = \mathbf{A}_j \mathbf{B}_j^T \mathbf{w}_j + \mathbf{A}_j \mathbf{e}_j = \mathbf{A}_j \mathbf{e}_j$. Therefore, the first check of Vrfy is true.

The second check holds, because we know that $\mathbf{e}_i = \mathbf{z}_k - \mathbf{B}_i^T \mathbf{w}_j$ is short due to the properties of $\mathsf{PreSample}$. Thus, the prover gets a valid witness as input and π verifies as true.

For correctness, we also need that an honestly generated signature opens to the correct user. As before, we know that $\mathbf{e}_j$ is short. Therefore, Babai's rounding algorithm [15] is able to correctly invert $\mathbf{z}_i$ to $\mathbf{w}_i$, since we know that $\mathbf{S}_i$ also fulfills the requirements of Babai's algorithm. Thus, he also computes the correct $\mathbf{e}_i$, as $\mathbf{e}_i = \mathbf{z}_i - \mathbf{B}_i^T \mathbf{w}_i$ by definition. Since it is highly probable that the other $\mathbf{e}_j$ that are computed by Open are not short, the signature opens to i.

Anonymity

An adversary looking at a signature is able to see a random value r, the $\mathbf{z}_j$ and a proof π. Since the random value is chosen uniformly at random, it does not help the adversary in distinguishing who signed the signature.

When we look at two signatures σ and σ' that were signed by i and j respectively, the only difference is in the computation of $\mathbf{z}_i$ and $\mathbf{z}_j$. Let $\mathbf{z}_i$ be the value from σ, while $\mathbf{z}'_i$ is from σ'. Then, we know that $\mathbf{z}_i$ was computed with an $\mathbf{e}_i$ that was output by PreSample, while $\mathbf{z}'_i$ was computed with an $\mathbf{e}_i$ that was chosen uniformly at random conditioned on $\mathbf{A}_i\mathbf{e}'_i = \mathbf{h}'_i$. From this, Gordon et al. are able to construct a distinguisher for LWE, as the distributions of $\mathbf{z}_i$ and $\mathbf{z}'_i$ are statistically close to $\mathbf{v} = \mathbf{A}\mathbf{s} + \mathbf{e}$ from the LWE game and a uniformly random $\mathbf{v}$ respectively. Since by Definition 5.11 we know that LWE is hard if GapSVP is hard, an adversary cannot distinguish between the encryption of a dummy signature and a real one. This in turn means that an adversary cannot distinguish between two statements for the proof π, where one instances was created by user i and the other by user $j \neq i$. Then, by the witness indistinguishability of the proof, the distribution of the proof in both instances looks the same for the adversary. Thus, he cannot distinguish who created a signature and the scheme offers anonymity.

Traceability

In order to output a valid signature that opens to a non-corrupted user, an adversary has to find a $\mathbf{z}_j$, such that $\mathbf{e}_j = \mathbf{z}_j - \mathbf{B}_j^T \left(\mathbf{S}_j \lfloor \mathbf{S}_j^{-1}\mathbf{z}_j \rceil\right) \leq s\sqrt{m}$ and j is uncorrupted, as well as a valid proof π. However, such an $\mathbf{e}_j$ also fulfills $\mathbf{A}_j\mathbf{e}_j = \mathbf{h}_j \mod q$, since the forgery is valid. But this means, that the adversary was able to create a forgery for the GPV signature scheme, which is a contradiction.

5.2.3 Techniques

As mentioned before, Gordon et al. follow the standard technique of combining a signature scheme, an encryption scheme and a proof of knowledge for tying the previous two together to get a group signature scheme. An important technique they use is computing keys for the digital signature scheme with OrthoSample. This way, they relate the (public) keys of the encryption scheme and digital signature scheme to each other. Due to this, verification is made easier, since $\mathbf{A}\mathbf{B}^T = \mathbf{0}$ now holds. Furthermore, the proof of knowledge needs less functionality, i.e. the statement that needs to be proven is simpler. As mentioned before, Gordon et al. only need to prove that a short vector was encrypted, instead of also proving that the encryption was done honestly.

The drawback to using OrthoSample is that Gordon et al. have to use multiple instances of the digital signature scheme and encryption scheme, one for each user. Thus, the size of the signatures is in $\mathcal{O}(\ell)$, which is a heavy penalty.

A thing to note is that the encryption scheme that Gordon et al. use is not

semantically secure. As mentioned before, it "encrypts" a vector $\mathbf{e}$ computing the ciphertext $\mathbf{z} = \mathbf{A}\mathbf{s} + \mathbf{e}$ for a uniformly random $\mathbf{s}$. Although this reminds of an LWE instance, the $\mathbf{e}$ is not random. Even if we want to encrypt a message m and thus compute $\mathbf{e} \coloneqq H(m)$ (with a random oracle) first, $\mathbf{e}$ is the same if we encrypt twice. If we computed $\mathbf{e} \coloneqq H(m||r)$ for a uniformly random r instead, Micciancio et al. [15] claim that this yields a secure encryption scheme. However, this shows us that a secure encryption scheme is not required to build a group signature scheme. In fact, Gordon et al. require a function f, such that the distribution of $f(\mathbf{e})$ where $\mathbf{e}$ is drawn uniformly at random is computationally indistinguishable from the distribution of $f(\mathbf{e})$ where $\mathbf{e}$ is drawn from a certain distribution. Furthermore, they also need that there exists a way to invert, i.e. "decrypt", $f(\mathbf{e})$, when in possession of a secret value.

Bibliography

[1] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, 2011.

[2] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. Quantum attacks on classical proof systems: The hardness of quantum rewinding. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 474–483. IEEE, 2014.

[3] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *Topics in Cryptology – CT-RSA 2005*, volume 3376, pages 136–153. Springer Berlin Heidelberg, 2005.

[4] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 626–643. Springer, 2012.

[5] 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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*. ACM Press, 2017.

[6] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*, pages 174–187. Springer, 1994.

[7] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[8] S Dov Gordon, Jonathan Katz, and Vinod Vaikuntanathan. A group signature scheme from lattice assumptions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–412. Springer, 2010.

[9] 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.

[10] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. Cryptology ePrint Archive, Report 2018/475, 2018. `https://eprint.iacr.org/2018/475`.

[11] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[12] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

[13] Daniele Micciancio and Salil P Vadhan. Statistical zero-knowledge proofs with efficient provers: Lattice problems and more. In *Annual International Cryptology Conference*, pages 282–298. Springer, 2003.

[14] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, page 333. ACM Press, 2009.

[15] Oded Regev. Lattice-based cryptography. In *Annual International Cryptology Conference*, pages 131–141. Springer, 2006.

[16] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[17] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 755–784. Springer, 2015.