UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Efficient Verifier-Local Revocation for Anonymous Credentials

Master's Thesis

by
Jan Bobolz
jbobolz@mail.uni-paderborn.de

Thesis Supervisor:
Prof. Dr. rer. nat. Johannes Blömer
and
Dr. rer. nat. Volker Krummel

November 02, 2015

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 singemäß übernommen worden sind, sind als solche gekennzeichnet.

City, Date

Signature

Contents

1 Introduction **1**

2 Foundation and Notation **5**
2.1 Notation . . . 5
2.2 Basics of (pairing-based) cryptography . . . 6
2.3 Proofs of knowledge . . . 7
2.4 Forking lemma . . . 9
2.5 Ring signatures . . . 10
2.6 Group signatures . . . 12
2.7 Assumptions . . . 15
2.8 Constructions . . . 16

3 Accumulators **19**
3.1 Definition . . . 19
3.2 The necessity of witness updates . . . 22
3.3 The Camenisch et al. accumulator . . . 23
3.4 Anonymously proving inclusion in the accumulator . . . 26

4 A ring signature scheme from the Camenisch et al. accumulator **29**
4.1 A protocol to prove revocation status . . . 29
4.1.1 Signing accumulator identities . . . 29
4.1.2 Combining the signature and accumulator protocol . . . 32
4.2 Constructing the ring signature scheme . . . 36
4.2.1 Construction . . . 36
4.2.2 Correctness . . . 39
4.2.3 Anonymity . . . 41
4.2.4 Unforgeability . . . 43

5 Revocation for group signatures **57**
5.1 Defining group signatures with accumulator revocation . . . 57
5.1.1 Syntax definition . . . 58
5.1.2 Security definitions . . . 61
5.2 A generic construction of group signature schemes with accumulator revocation . . . 66
5.2.1 Construction . . . 67
5.2.2 Correctness . . . 68
5.2.3 Anonymity . . . 69

5.2.4 Traceability 72
5.2.5 Unforgeability 72
5.2.6 Unforgeability of epoch information 75
5.2.7 Performance 77

6 Anonymous credential systems **79**
6.1 Definition of anonymous credential systems 80
6.1.1 Syntax 80
6.1.2 Anonymity 82
6.1.3 Soundness 85
6.2 Anonymous credential systems with revocation 87
6.2.1 Syntax 88
6.2.2 Anonymity 90
6.2.3 Soundness 91

7 Conclusion **95**

Bibliography **97**

1 Introduction

Today, authentication over the Internet typically involves submitting some identifying information. This can be a user name and a password, or a public key certificate. After the user's identity is established, the authenticator admits or denies access to some resource. However, it is often not necessary for the authenticator to know the user's actual identity. For example, the operator of an anonymous paid discussion platform only needs to know that the authenticating user has a valid subscription to the service. The actual identity should be hidden from the operator and only known to some billing agency. Additionally, as a user, using the same (certified) identity for multiple services enables these services to share collected data associated with the user's identity to create extensive profiles without user consent.

For these reasons, systems that allow users to authenticate anonymously become increasingly interesting. One simple example of such schemes are the well-known *group signatures*, where a user signs a message (proving authenticity) as part of a group of users. The receiver (verifier) can check that the message originated from some user of the group, but cannot determine which user signed the message. Only a special entity, the *group manager* is able to trace signatures to a signer. In our discussion forum example above, paying users may receive a group signature key with which they can sign their messages. The forum software can simply check signature validity to decide whether or not to post a message, removing the need for users to authenticate with their identity. Still, in case of legal dispute, the group manager (e.g., law enforcement) can uncover who wrote a message.

Another example of anonymous authentication can be found in *anonymous credential systems* [Lys02], where users are issued credentials with certain attributes that they can use to authenticate themselves with services. A service knows its users only under pseudonyms and when a user shows a credential to a service, he has fine-grained control over what exactly he wants to disclose about his attributes and identity. Overall, users stay anonymous unless, for example, they choose to reveal an attribute that is unique to them. In particular, users cannot be tracked between multiple services (as the services know the user under different, unlinkable pseudonyms).

However, anonymity complicates certain desired functionalities. In this thesis, we are interested in *revocation*. In case of misuse or, for example, if a subscription expires, a user's credentials or signatures should be declared invalid. In contexts where users are not anonymous, this can be easily accomplished by announcing "*signatures this user* or *this credential* should not be considered valid anymore". But this by itself does not solve the problem in an anonymous context since by design, verifiers cannot distinguish signers or credentials.

There exist revocation approaches using such lists as described above, notably [BS04]

for group signatures, or proving in zero-knowledge that a credential is not on a revocation list (using a generic proof protocol construction [GMW87]). However, these approaches are not practical for large revocation lists since the cost to check revocation status is at least linear in the number of revocation entries. For this reason, the use of cryptographic *accumulators* for revocation is interesting (e.g., [CKS09]). Basically, an accumulator maps a set of revocable identities to a constant-size digest. If an identity is included in the accumulator, then there exists a *witness* that convinces anyone knowing the digest that the identity is indeed included. Here, the verification time is independent of the number of values in the accumulator. If an identity is not included, then it should be computationally infeasible to compute a witness.

In this thesis, we are concerned with the accumulator introduced by Camenisch et al. that is based on bilinear maps [CKS09]. Its algebraic structure allows for efficient protocols to prove inclusion in the accumulator, which makes it usable in anonymity-preserving contexts. However, the question arises under which circumstances an accumulator revocation mechanism can be added to existing constructions and what exact revocation semantics can be achieved by this.

In this thesis, we examine and formally define the revocation semantics that we can expect from accumulator revocation mechanisms in group signatures and anonymous credential systems. Furthermore, at the example of group signature schemes, we answer the question of requirements for existing constructions to allow for accumulator revocation with the Camenisch et al. construction mentioned above.

Our main contributions are as follows:

- We formally define new revocation semantics for group signatures that can be realized with accumulators. As it turns out, the notion of revoking *signing rights*, as opposed to retroactively revoking all the user's signatures, is more adequate in an accumulator context (Chapter 5).
- We answer the question about which group signature schemes can be extended to allow for accumulator revocation by giving a generic construction that works for *any* secure group signature scheme (Chapter 5).
- We introduce a ring signature scheme where the ring of signers is managed through the Camenisch et al. accumulator [CKS09]. The methods used to design the ring signature scheme serve as an abstract guideline of how to employ that accumulator for anonymous revocation, and the ring signature scheme will serve as a concrete building block for our generic group signature construction with accumulator revocation (Chapter 4).
- We define revocation semantics for anonymous credential systems. To do this, we give a formal definition of credential systems based on experiments (as opposed to ideal worlds as in [Lys02]) and extend that definition with revocation semantics (Chapter 6).

As a noteworthy minor result, we give an impossibility result for accumulators that do not require witness updates (Observation 3.4).

The thesis is structures as follows. In Chapter 2, we define the notation used in this thesis and give basic definitions for the used schemes. In Chapter 3, we define accumulator schemes and discuss one particular construction introduced by Camenisch et al. [CKS09], which we will use throughout the thesis. Afterwards, in Chapter 4, we construct a ring signature scheme using the accumulator. This ring signature scheme will be used as a building block in Chapter 5, where we define group signature schemes with accumulator revocation and give a generic construction for such a scheme. Finally, in Chapter 6, we examine revocation for anonymous credential systems.

2 Foundation and Notation

This chapter introduces the basic notation and definitions used throughout the thesis.

2.1 Notation

- For a map $\phi : X \to Y$, the *image* of ϕ is $\mathrm{im}(\phi) := \phi(X) := \{y \in Y \mid \exists x \in X : \phi(x) = y\}$.
- $\mathbb{N}$ is the set of natural numbers ($0 \notin \mathbb{N}$) and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.
- $[n]$ is the set of natural numbers less than $n+1$, i.e. $[n] = \{1, \ldots, n\}$.
- For $n \in \mathbb{N}$ we set $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$, that is the ring of integers modulo n.
- 1^k, when used as input for algorithms, denotes the string of k ones (i.e. $1^k \in \{1\}^k$).
- For sets S, S', the expression $S \subset S'$ denotes strict inclusion, i.e. $S \subseteq S' \wedge S \neq S'$.
- For a set S, $2^S := \{X \mid X \subseteq S\}$ denotes the powerset of S.
- For a finite set S we write $s \xleftarrow{R} S$ if s is picked uniformly at random from S.
- For a probabilistic algorithm $\mathcal{A}$, we write $Z \leftarrow \mathcal{A}(\cdot)$ if Z is a random variable taking on the return value of $\mathcal{A}$. And if $Z \leftarrow \mathcal{A}(x)$, we write $[\mathcal{A}(x)] := [Z] := \{y \mid \Pr[Z = y] > 0\}$.
- For two probabilistic, interacting algorithms (in terms of interactive Turing machines) $\mathcal{A}, \mathcal{B}$, the expression $\mathcal{A}(x) \leftrightarrow \mathcal{B}(y)$ denotes a random variable that takes on the transcript of exchanged messages when $\mathcal{A}$ is run with input x and $\mathcal{B}$ is run with input y.
- Similarly, $\mathcal{A}(x) \overset{o}{\leftrightarrow} \mathcal{B}(y)$ denotes a random variable that takes on the value of $(\tau, o_{\mathcal{A}}, o_{\mathcal{B}})$, where τ is the protocol's transcript as above and $o_{\mathcal{A}}, o_{\mathcal{B}}$ denote the output of $\mathcal{A}$ and $\mathcal{B}$, respectively.
- $\Pr[x_1 \leftarrow X_1, \ldots, x_n \leftarrow X_n : \phi(x_1, \ldots, x_n)]$ denotes the probability that $\phi(x)$ is true in the probability space where x_i is chosen according to the random variable X_i for $i \in [n]$.
- $x \mapsto f(x)$ denotes a mapping (the domain and codomain are apparent from the context) that maps any value x from the domain to $f(x)$.

- If $f : A \times B \to C$ is a mapping, then for any $a \in A$, $f(a, \cdot)$ denotes the mapping $b \mapsto f(a, b)$ $(B \to C)$.

- As usual, we assume that arguments of algorithms are encoded in a reasonable manner. For example, for a finite group $\mathbb{G}$ and an algorithm $\mathcal{A}$, the expression $\mathcal{A}(\mathbb{G})$ does not mean that $\mathbb{G}$ is passed as a list of all its elements. Rather, a description of $\mathbb{G}$ is passed (in particular, $\mathcal{A}$ is not a polynomial-time algorithm if it iterates over all group elements).

2.2 Basics of (pairing-based) cryptography

This section introduces basic definitions widely used in modern cryptography and especially in the context of pairing-based cryptography.

For our security definitions, we require the notion of *negligible functions*.

Definition 2.1 (Negligible functions)**.** A function $\mu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is called *negligible* if $\forall c \in \mathbb{N}\ \exists n_0 \in \mathbb{N}\ \forall n > n_0$,

$$\mu(n) \leq n^{-c}$$

A typical requirement is that any probabilistic polynomial-time attacker should succeed in breaking a scheme only with *negligible* probability, i.e. with increasing security parameter, the probability of a successful attack should approach zero faster than the inverse of any polynomial.

In pairing-based cryptography, *bilinear maps* between finite groups play a central role. We define *bilinear groups* as follows:

Definition 2.2 (Bilinear group)**.** A bilinear group is a tuple $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$ where $\mathbb{G}$ and $\mathbb{G}_\mathrm{T}$ are groups of prime order p, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$ is a map which is

- Bilinear: $e(g^a, g^b) = e(g, g)^{ab}$ for any $g \in \mathbb{G}, a, b \in \mathbb{Z}_p$.

- Non-degenerate: $\mathrm{im}(e) \neq \{1\}$.

- Efficiently computable: there is an efficient algorithm that evaluates e.

e is often also called a *pairing*.

Examples for bilinear groups are elliptic curve groups with the Weil pairing.

2.3 Proofs of knowledge

A useful tool for (anonymous) authentication is the notion of interactive protocols that are zero-knowledge and a proof of knowledge. They allow provers to convince verifiers that they possess a *witness* corresponding to some shared input between the verifier and the prover. In particular, the protocol does not leak any information about the witness used by the prover, which will later correspond to the verifier staying anonymous.

In this context, Σ protocols [Sch15] play a large role.

Definition 2.3 (Σ protocol [Sch15])**.** Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be an NP-relation, i.e. $L_R := \{x \mid \exists w : (x,w) \in R\} \in \text{NP}$.

A Σ protocol for R is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. Both parties receive common input $x \in \{0,1\}^*$ and $\mathcal{P}$ gets additional (private) input w. The prover computes and sends the first message t, the verifier responds with a challenge message c chosen uniformly at random from some set C, and the prover answers with a message s. $\mathcal{V}$ either accepts or rejects the transcript (t,c,s).

A Σ protocol also fulfills the following properties:

- **Completeness**: If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol for $(x,w) \in R$, the resulting transcript is accepting.
- **Special soundness**: There exists a probabilistic polynomial-time extractor algorithm that, given $x \in L_R$ and a pair of transcripts $(t,c,s),(t,c',s')$ that are accepting for x and where $c \neq c'$, computes a witness w with $(x,w) \in R$.
- **Special honest-verifier zero-knowledge**: There exists a probabilistic polynomial-time simulator algorithm that, given $x \in L_R$ and a challenge $c \in C$, produces a transcript (t,c,s) distributed like $\mathcal{V}(x) \leftrightarrow \mathcal{P}(x,w)$ for all w with $(x,w) \in R$.

In this thesis, we employ a popular notation for proofs of knowledge introduced by Camenisch and Stadler [CS97].

Definition 2.4 (Proof of knowledge notation)**.** $PK\{w \mid \phi(x,w)\}$, where ϕ is a predicate, denotes a Σ-protocol for the relation $\{(x,w) \mid \phi(x,w)\}$. The domains of x and w will be apparent from the context.

The question arises how to realize such Σ protocols defined through this notation. For this, there is a generic Σ protocol construction based on the Schnorr protocol that works for all occurrences (in this thesis) of the notation defined in Definition 2.4. The special case $m = 1$ was described in [Bra97].

Construction 2.5. Let $\mathbb{G}_1, \ldots, \mathbb{G}_m$ be groups of the same prime order p. Consider m equations $A_j = \prod_{i=1}^{n} g_{j,i}^{x_i}$ for $j \in [m]$, where $A_j, g_{j,i} \in \mathbb{G}_j$ for all $i \in [n], j \in [m]$.
On common input $(\mathbb{G}_1, \ldots, \mathbb{G}_m, A_1, \ldots, A_m, ((g_{j,i})_{i=1}^{n})_{j=1}^{m})$ and additional prover input $(x_1, \ldots, x_n) \in \mathbb{Z}_p^n$, the following protocol between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ can be executed:

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
choose $(t_1, \ldots, t_n) \xleftarrow{R} \mathbb{Z}_p^n$		
for each $j \in \{1, \ldots, m\}$,		
calculate $T_j = \prod_{i=1}^{n} g_{j,i}^{t_i}$		
	$\xrightarrow{T_1, \ldots, T_m}$	
		$c \xleftarrow{R} \mathbb{Z}_p$
	$\xleftarrow{c}$	
for each $i \in \{1, \ldots, n\}$,		
calculate $s_i = x_i \cdot c + t_i$		
	$\xrightarrow{s_1, \ldots, s_n}$	
		accept if
		$\forall j : \prod_i g_{j,i}^{s_i} \stackrel{!}{=} A_j^c T_j$

Lemma 2.6. *Construction 2.5 is a Σ protocol for the relation*

$$R = \{((\mathbb{G}_1, \ldots, \mathbb{G}_m, A_1, \ldots, A_m, ((g_{j,i})_{i=1}^{n})_{j=1}^{m}), (x_1, \ldots, x_n)) \mid \forall i \in [n]\ \forall j \in [m] : A_j, g_{j,i} \in \mathbb{G}_j \wedge A_j = \prod_{i=1}^{n} g_{j,i}^{x_i}\}$$

where $\mathbb{G}_i, A_i, g_{j,i}$ are as defined in the construction.

Proof. We need to prove *completeness*, *special soundness*, and *special honest-verifier zero-knowledge*.
Completeness

If both parties follow the protocol, then the verifier always accepts. This is the case because

$$\prod_i g_{j,i}^{s_i} = \prod_i g_{j,i}^{x_i \cdot c + t_i} = \prod_i (g_{j,i}^{x_i})^c \cdot \prod_i g_{j,i}^{t_i} = A_j^c \cdot T_j$$

for each $j \in [m]$ and hence, the verifier accepts.

Special soundness

We define an extractor that can extract a witness $w = (x_1, \ldots, x_n)$ given $(\mathbb{G}_1, \ldots, \mathbb{G}_m, A_1, \ldots, A_m, ((g_{j,i})_{i=1}^n)_{j=1}^m)$ and two accepting transcripts $((T_1, \ldots, T_m), c, (s_1, \ldots, s_n))$ and $((T_1, \ldots, T_m), c', (s'_1, \ldots, s'_n))$ with $c \neq c'$. This extractor computes $x_i = (s_i - s'_i)/(c - c')$ for each $i \in [n]$ and returns $(x_1, \ldots, x_n)$.

Because the two transcripts are accepting, it holds that $\prod_i g_{j,i}^{s_i} = A_j^c T_j$ and $\prod_i g_{j,i}^{s'_i} = A_j^{c'} T_j$ for all $j \in [m]$. Dividing both equalities yields

$$\prod_i g_{j,i}^{s_i - s'_i} = A_j^{c-c'}$$

and hence

$$\prod_i g_{j,i}^{\frac{s_i - s'_i}{c-c'}} = A_j$$

using that $c - c' \neq 0$. Consequently, setting $x_i = (s_i - s'_i)/(c - c')$ results in a valid witness $(x_1, \ldots, x_n)$.

Special honest-verifier zero-knowledge

Interactions between honest $\mathcal{V}$ and $\mathcal{P}$ can be simulated as follows: Given $(\mathbb{G}_1, \ldots, \mathbb{G}_m, A_1, \ldots, A_m, ((g_{j,i})_{i=1}^n)_{j=1}^m)$ and $c \in \mathbb{Z}_p$, choose $(s_1, \ldots, s_n) \xleftarrow{R} \mathbb{Z}_p^n$ and set $T_j = \prod_i g_{j,i}^{s_i} / A_j^c$ for $j \in [m]$. The simulator outputs the transcript $((T_1, \ldots, T_m), c, (s_1, \ldots, s_n))$.

The set of transcripts generated this way is equal to the set of accepting transcripts between two honest parties, namely $\{((T_1, \ldots, T_m), c, (s_1, \ldots, s_n)) \mid \forall j : \prod_i g_{j,i}^{s_i} = A_j^c T_j\}$. Furthermore, the simulated transcripts for a fixed c are distributed uniformly in this set because the s_i are uniformly distributed and the other elements are determined through that. In honest transcripts for a fixed c, the s_i are also distributed uniformly and independently because of the relationship $s_i = x_i \cdot c + t_i$ where $t_i \xleftarrow{R} \mathbb{Z}_p$. Hence, our simulator outputs transcripts with the same distribution as the original protocol. □

Construction 2.5 can only demonstrate knowledge of some vector of exponents (as opposed to elements of a group $\mathbb{G}$). In order to solve this, often a proof of knowledge of some group element g is constructed in the following form: First, choose random $r \xleftarrow{R} \mathbb{Z}_p$ and send a blinded g formed as $\mathcal{G} = g\tilde{h}^r$ (for some generator $\tilde{h}$), then run some Σ-protocol proving knowledge of the value r that can be used to de-randomize $\mathcal{G}$ to a value g with the desired properties.

2.4 Forking lemma

Typically, the proof technique for proof of knowledge protocols involves the following argument. If a prover $\mathcal{P}$ is able to pass the protocol (in this example, a Σ protocol) without being given a witness, $\mathcal{P}$ can be rewound to the point where it receives the

challenge, and be run from that point with a new random challenge. If $\mathcal{P}$ has non-negligible success probability, it will likely also pass the second challenge, giving us two transcripts (T, c, s) and (T, c', s'). In the likely case that $c \neq c'$, the (special) soundness property gives us a way to compute a witness for the protocol from just the common input (which should be hard).

The argument that rewinding the algorithm likely supplies two such transcripts is encapsulated in the (generalized) *forking lemma* as described in [BN06]. Prior to that, a different version of this lemma, for the special case of signatures derived from the Fiat-Shamir heuristic, was introduced by Pointcheval and Jacques [PS00].

Lemma 2.7 (Forking lemma [BN06])**.** *Let $q \in \mathbb{N}$ and let H be a set, $|H| \geq 2$. Let $\mathcal{A}$ be a randomized algorithm that takes input $(x, h) \in \{0,1\}^* \times H^q$ and outputs $(J, \sigma) \in (\{0, \ldots, q\}) \times \{0,1\}^*$. Let $\mathfrak{G}$ be an instance generator for x. Let μ be the probability that $\mathcal{A}$ outputs some (J, σ) with $J \neq 0$, i.e. $\mu = \Pr[x \leftarrow \mathfrak{G}, h \xleftarrow{R} H^q, (J, \sigma) \leftarrow \mathcal{A}(x, h) : \ J \neq 0]$.*

Consider the algorithm $F_{\mathcal{A}}(x)$.

- *$F_{\mathcal{A}}$ generates random bits $\omega_{\mathcal{A}}$ for $\mathcal{A}$ and chooses $h = (h_1, \ldots, h_q) \xleftarrow{R} H^q$.*
- *It then runs $\mathcal{A}(x, h)$ with random bits $\omega_{\mathcal{A}}$ to obtain (I, σ). If $I = 0$, $F_{\mathcal{A}}$ outputs* fail *and aborts.*
- *It chooses new $h'_I, \ldots, h'_q \xleftarrow{R} H$ and sets $h' := (h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q)$.*
- *It then runs $\mathcal{A}(x, h')$ with the same random bits $\omega_{\mathcal{A}}$ to obtain (I', σ'). If $I = 0$, $F_{\mathcal{A}}$ outputs* fail *and aborts.*
- *If $I = I'$ and $h_I \neq h'_I$, F_A outputs σ, σ'. Otherwise, it outputs* fail *and aborts.*

Then $\Pr[x \leftarrow \mathfrak{G} : \ F_{\mathcal{A}}(x) \neq \text{fail}] \geq \mu(\frac{\mu}{q} - \frac{1}{|H|})$

We omit the proof and refer to [BN06] for details.

To apply this lemma for the scenario described above, assume $\mathcal{A}$ is a prover with success probability μ for a fixed security parameter. In this case, x is the randomly chosen common input of the protocol, H is the verifier's challenge space and $q = 1$ (corresponding to the one challenge $h_1 \in H$). Let $\mathcal{B}$ be an algorithm that simulates the protocol, choosing the verifier's challenge uniformly at random, with $\mathcal{A}$ in the prover role, which results in a transcript σ. $\mathcal{B}$ outputs $(I\sigma)$, where $I = 1$ if σ is an accepting transcript, $I = 0$ otherwise. Then the rewinder algorithm $F_{\mathcal{B}}$ outputs two transcripts σ, σ' with different challenges $h_1 \neq h'_1$ with probability at least $\mu(\mu - 1/|H|)$.

2.5 Ring signatures

In (identity-based) ring signature schemes, messages are signed by a set V (*ring*) of signers. When signing a message, the signer i can choose the signer ring V freely, as long as $i \in V$. The informal security guarantees are that user i should not be able to

create a signature for a ring V with $i \notin V$, and verifiers should not be able to distinguish potential signers $i, i' \in V$.

Our definition of ring signature schemes and their security is based on [BKM06]. We changed the definition to fit so-called *identity-based* ring signatures, where a ring, for which signatures are created, is not a set of public keys but a set of identities.

Definition 2.8 ((Identity-based) ring signature scheme)**.** A ring signature scheme consists of three polynomial-time algorithms:

- $\text{KeyGen}(1^\lambda, 1^n)$ is a probabilistic algorithm that generates a public key pk, a set of member identities U with $|U| = n$, and member keys $(sk[i])_{i \in U}$. It outputs $(pk, U, (sk[i])_{i \in U})$.
- $\text{Sign}_{sk[i]}(m, V)$ is a probabilistic algorithm that outputs a signature σ for a group $V \subseteq U$
- $\text{Verify}_{pk}(m, V, \sigma)$ is a deterministic algorithm that returns 0 or 1.

We say that such a scheme is *correct* if for all $\lambda, n \in \mathbb{N}$ and for all $(pk, U, (sk[i])_{i \in U}) \in [\text{KeyGen}(1^\lambda, 1^n)]$, $V \subseteq U$ and $i \in V$,

$$\Pr[\text{Verify}_{pk}(m, V, \text{Sign}_{sk[i]}(m, V)) = 1] = 1$$

In the following, we define security for ring signature schemes. We start with *anonymity*, which is the property that signers stay anonymous inside the ring of signers. We consider *anonymity against full key exposure*, which is the strongest definition in [BKM06] where an adversary is given the randomness used to create the keys and has to distinguish two signers. Here, we introduce an even stronger notion, which we call *anonymity against early full key exposure*.

Definition 2.9 (Anonymity against early full key exposure)**.** A ring signature scheme Π admits *anonymity (against early full key exposure)* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\text{Adv}^{\text{anon}}_{\Pi,\mathcal{A}}(\lambda, n) := |\Pr[\text{Exp}^{\text{anon}-1}_{\Pi,\mathcal{A}}(\lambda, n) = 1] - \Pr[\text{Exp}^{\text{anon}-0}_{\Pi,\mathcal{A}}(\lambda, n) = 1]| \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiments $\text{Exp}^{\text{anon}-b}_{\Pi,\mathcal{A}}(\lambda, n)$ for $b \in \{0, 1\}$ work as follow:

$(pk, U, (sk[i])_{i \in U}) \leftarrow \text{KeyGen}(1^\lambda, 1^n)$ using randomness $\omega \xleftarrow{R} \{0,1\}^{poly(\lambda)}$.

$\mathcal{A}$ is given pk, U, and ω.

Eventually $\mathcal{A}$ outputs two identities $(i_0, i_1) \in U^2$, a set V^* with $\{i_0, i_1\} \subseteq V^* \subseteq U$ and a message m^*.

$\mathcal{A}$ is given the randomness ω used to create the keys. It is also given a signature $\sigma^* \leftarrow \mathrm{Sign}_{sk[i_b]}(m^*, V^*)$ created using $sk[i_b]$.

In the end, $\mathcal{A}$ outputs a bit b'. The output of the experiment is b'.

In this experiment, $\mathcal{A}$ is handed a signature created by one of two possible signers. $\mathcal{A}$, knowing all keys (and even the randomness ω used to created them), needs to distinguish the signers. In contrast to [BKM06], we hand the adversary the randomness ω *before* it has to choose the ring V^* and the two identities i_0, i_1 to distinguish. This also eliminates the need for access to a signing oracle.
We now consider unforgeability, which is the property that guarantees that a valid signature was created by some user in the ring. The following definition of unforgeability is based on *Unforgeability w.r.t. insider corruption* from [BKM06].

Definition 2.10 (Unforgeability w.r.t. insider corruption)**.** A ring signature scheme Π has *unforgeable signatures (w.r.t. insider corruption)* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n) := \Pr[\mathrm{Exp}^{\mathrm{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n)$ is defined as follows:

$(pk, U, (sk[i])_{i \in U}) \leftarrow \mathrm{KeyGen}(1^\lambda, 1^n)$.

$\mathcal{A}$ is given pk, U, oracle access to $\mathrm{Sign}_{sk[\cdot]}(\cdot, \cdot)$, and oracle access to $i \mapsto sk[i]$.

Eventually, $\mathcal{A}$ outputs a message m^*, a set $V^* \subseteq U$ and a signature σ^*.

If $\mathrm{Verify}_{pk}(m^*, V^*, \sigma^*) = 1$ and $\mathcal{A}$ has neither queried $\mathrm{Sign}_{sk[\cdot]}(m^*, V^*)$ nor $sk[i]$ for any $i \in V^*$, the experiment outputs 1. Otherwise it outputs 0.

In this definition, $\mathcal{A}$ may adaptively *corrupt* some set of users by accessing the $sk[i]$ oracle. $\mathcal{A}$ should not be able to sign a message for a ring where none of his corrupted users are in. Oracle access to Sign models that an attacker may indeed see arbitrary valid signatures, which should not help in creating signatures on any other message or ring.

2.6 Group signatures

In group signatures, a user signs messages on behalf of the group. In contrast to ring signatures, this group is not chosen ad-hoc when signing a message, but is fixed at setup

(assuming a definition of group signatures without revocation). For the group, there exists a special entity, the *group manager* that is given a special secret key $gmsk$. With this key, the group manager is able to uncover who created a signature (e.g., in case of dispute). Verifiers without $gmsk$ can be sure that a valid signature was created by some user of the group, but cannot determine which user.

Our definition for group signature schemes and their security is based on [BMW03].

Definition 2.11 (Group signature scheme)**.** A group signature scheme consists of four polynomial-time algorithms:

- $\text{KeyGen}(1^\lambda, 1^n)$ is a probabilistic algorithm that generates a set U of identities (with $|U| = n$), a public key gpk, the group manager's secret key $gmsk$, and member secret keys $(sk[i])_{i \in U}$. It outputs $(U, gpk, gmsk, (sk[i])_{i \in U})$.
- $\text{Sign}_{sk[i]}(m)$ is a probabilistic algorithm that outputs a signature σ.
- $\text{Verify}_{gpk}(m, \sigma)$ is a deterministic algorithm that returns 0 or 1.
- $\text{Open}_{gmsk}(m, \sigma)$ is a deterministic algorithm that returns an identity $i \in U$ or the failure symbol $\perp$.

A group signature scheme is *correct* if for all $\lambda, n \in \mathbb{N}, (U, gpk, gmsk, (sk[i])_i) \in [\text{KeyGen}(1^\lambda, 1^n)], i \in U$, and all messages m,

$$\Pr[\text{Verify}_{gpk}(m, \text{Sign}_{sk[i]}(m)) = 1] = 1$$

and

$$\Pr[\text{Open}_{gmsk}(m, \text{Sign}_{sk[i]}(m)) = i] = 1$$

In the following, we define security of group signature schemes [BMW03]. We start with anonymity, which is the requirement that verifiers cannot distinguish signers.

Definition 2.12 (Full anonymity)**.** A group signature scheme Π has *full anonymity* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\text{Adv}^{\text{anon}}_{\Pi,\mathcal{A}}(\lambda, n) := |\Pr[\text{Exp}^{\text{anon}-1}_{\Pi,\mathcal{A}}(\lambda, n) = 1] - \Pr[\text{Exp}^{\text{anon}-0}_{\Pi,\mathcal{A}}(\lambda, n) = 1]| \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiments $\text{Exp}^{\text{anon}-b}_{\Pi,\mathcal{A}}(\lambda, n)$ for $b \in \{0, 1\}$ work as follow:

$(U, gpk, gmsk, (sk[i])_{i \in U}) \leftarrow \text{KeyGen}(1^\lambda, 1^n)$.

$\mathcal{A}$ is given $U, gpk, (sk[i])_{i \in U}$ and oracle access to $\text{Open}_{gmsk}(\cdot, \cdot)$.

Eventually $\mathcal{A}$ outputs two identities $(i_0, i_1) \in U^2$ and a message m^*.

$\mathcal{A}$ is given a signature $\sigma^* \leftarrow \mathrm{Sign}_{sk[i_b]}(m^*)$ created using $sk[i_b]$.

$\mathcal{A}$ continues to have oracle access to $\mathrm{Open}_{gmsk}(\cdot,\cdot)$.

In the end, $\mathcal{A}$ outputs a bit b'. If $\mathcal{A}$ has queried $\mathrm{Open}_{gmsk}(m^*,\sigma^*)$ after receiving the signature, the experiment outputs 0, otherwise it outputs b'.

Note that in contrast to the corresponding anonymity definition for ring signatures (Definition 2.9), here $\mathcal{A}$ cannot be given the randomness used to run KeyGen, as it would allow it to compute $gmsk$, which can always be used to disable anonymity. We do however give $\mathcal{A}$ all user keys and oracle access to Open for any signature but the challenge signature which it tries to distinguish (if it tries to query Open for σ^*, the experiment outputs 0).

We now consider *full traceability*, which states that the group manager should always be able to trace a valid signature to its correct signer.

Definition 2.13 (Full traceability)**.** A group signature scheme Π has *full traceability* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{trace}}_{\Pi,\mathcal{A}}(\lambda,n) := \Pr[\mathrm{Exp}^{\mathrm{trace}}_{\Pi,\mathcal{A}}(\lambda,n)=1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{trace}}_{\Pi,\mathcal{A}}(\lambda,n)$ is defined as follows:

$(U, gpk, gmsk, (sk[i])_{i=1}) \leftarrow \mathrm{KeyGen}(1^\lambda, 1^n)$.

$\mathcal{A}$ is given $U, gmsk, gpk$, oracle access to $\mathrm{Sign}_{sk[\cdot]}(\cdot)$, and oracle access to $i \mapsto sk[i]$.

Eventually, $\mathcal{A}$ outputs a message m^* and a signature σ^*.

If $\mathrm{Verify}_{gpk}(m^*,\sigma^*) = 1$ and $\mathrm{Open}_{gmsk}(m^*,\sigma^*) = \perp$, the experiment outputs 1.

If $\mathrm{Verify}_{gpk}(m^*,\sigma^*) = 1$ and $\mathrm{Open}_{gmsk}(m^*,\sigma^*) = i \in U$, and $\mathcal{A}$ has neither queried $\mathrm{Sign}_{sk[i]}(m^*)$ nor $sk[i]$, the experiment outputs 1.

Otherwise it outputs 0.

Here, $\mathcal{A}$ is given oracle access to $sk[\cdot]$ (which models corruption) and wins the experiment if it is able to output a valid signature that cannot be traced by Open to any user, or that is traced to an uncorrupted user i (without querying the Sign for that signature).

Finally, we define unforgeability, which states that it should be hard to create valid signatures without being given any secret key.

Definition 2.14 (Unforgeability)**.** A group signature scheme Π has *unforgeable signatures* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{forge}}_{\Pi,\mathcal{A}}(\lambda, n) := \Pr[\mathrm{Exp}^{\mathrm{forge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{forge}}_{\Pi,\mathcal{A}}(\lambda, n)$ is defined as follows:

$(U, gpk, gmsk, (sk[i])_{i=1}^{n}) \leftarrow \mathrm{KeyGen}(1^{\lambda}, 1^{n})$.

$\mathcal{A}$ is given U, gpk and oracle access to $\mathrm{Sign}_{sk[\cdot]}(\cdot)$.

Then, $\mathcal{A}$ outputs a message m^* and a signature σ^*.

If $\mathrm{Verify}_{gpk}(m^*, \sigma^*) = 1$ and $\mathcal{A}$ has not queried $\mathrm{Sign}_{sk[\cdot]}(m^*)$, the experiment outputs 1, otherwise it outputs 0.

Note that unforgeability immediately follows from full traceability.

2.7 Assumptions

In this section, we introduce several assumptions that have not been proven yet (and because these assumptions typically imply $P \neq NP$, it seems nontrivial to prove any of them in the standard model).

Definition 2.15 (n-DHE assumption [CKS09])**.** Let $\mathfrak{G}(\lambda)$ be an instance generator that outputs a description of a prime-order group $\mathbb{G}$. The n-DHE assumption for $\mathfrak{G}$ states that for all probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function μ such that

$$\Pr[\mathbb{G} \leftarrow \mathfrak{G}(\lambda), g \xleftarrow{R} \mathbb{G} \setminus \{1\}, \gamma \leftarrow \mathbb{Z}_{|\mathbb{G}|} : \mathcal{A}((g^{\gamma^i})_{i=0; i \neq n+1}^{2n}) = g^{\gamma^{n+1}}] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$

Definition 2.16 (n-SDH assumption [BB04])**.** Let $\mathfrak{G}(\lambda)$ be an instance generator that outputs a description of a prime-order group $\mathbb{G}$. The n-SDH assumption (*strong Diffie-Hellman assumption*) for $\mathfrak{G}$ states that for all probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function μ such that

$$\begin{aligned}\Pr[&\mathbb{G} \leftarrow \mathfrak{G}(\lambda), g \xleftarrow{R} \mathbb{G} \setminus \{1\}, \delta \leftarrow \mathbb{Z}_{|\mathbb{G}|} : \\ &\exists c \in \mathbb{Z}_p^* : \ \mathcal{A}(g, (g^{\delta^i})_{i=1}^{n}) = (g^{1/(\delta+c)}, c)] \leq \mu(\lambda)\end{aligned}$$

for all $\lambda \in \mathbb{N}$.

Definition 2.17 (n-HSDHE assumption [CKS09])**.** Let $\mathfrak{G}(\lambda)$ be an instance generator that outputs a description of a prime-order group $\mathbb{G}$. The n-HSDHE assumption (*hidden strong Diffie-Hellman exponent assumption*) for $\mathfrak{G}$ states that for all probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function μ such that

$$\Pr[\mathbb{G} \leftarrow \mathfrak{G}(\lambda), g, u \xleftarrow{R} \mathbb{G} \setminus \{1\}, \gamma, \delta \leftarrow \mathbb{Z}_{|\mathbb{G}|} : \exists c \in \mathbb{Z}_p \setminus \{\gamma^i \mid i \in [n]\} : \mathcal{A}(g, g^\delta, u, (g^{1/(\delta+\gamma^i)}, g^{\gamma^i}, u^{\gamma^i})_{i=1}^n, (g^{\gamma^i})_{i=n+2}^{2n}) = (g^{1/(\delta+c)}, g^c, u^c)] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$. For the unlikely event that $\delta + \gamma^i = 0$, we set $0^{-1} := 0 \in \mathbb{Z}_p$ in the expressions above.

Note that n-HSDHE implies n-SDH for any instance generator.

2.8 Constructions

In this section, we introduce some basic constructions used throughout the thesis.

First, a signature scheme based on Boneh-Boyen signatures and proven secure in [Oka06].

Construction 2.18 (Boneh-Boyen signature variant [Oka06])**.**

- $\mathrm{KeyGen}(1^\lambda)$ generates a bilinear group $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$ of prime order p and generators $h, h_0, h_1, h_2 \xleftarrow{R} \mathbb{G} \setminus \{1\}$. It then sets $sk \xleftarrow{R} \mathbb{Z}_p$, $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), h, h_0, h_1, h_2, h^{sk})$ and outputs (pk, sk).
- $\mathrm{Sign}_{sk}(m)$ for a message $m \in \mathbb{Z}_p$, the algorithm picks $c, s \xleftarrow{R} \mathbb{Z}_p$. It then computes $\sigma = (h_0 h_1^m h_2^s)^{1/(sk+c)}$ and outputs (σ, c, s) as a signature.
- $\mathrm{Verify}_{pk}(m, (\sigma, c, s))$ checks that $e(h_0 h_1^m h_2^s, h) \stackrel{!}{=} e(\sigma, pk \cdot h^c)$ and returns 1 if the equation holds, 0 otherwise.

The following scheme is a weakly secure signature scheme introduced in [BB04].

Construction 2.19 (A weakly secure short signature scheme [BB04])**.**

- KeyGen(1^λ) generates a bilinear group $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p) \leftarrow \mathfrak{G}(\lambda)$ of prime order p and a generator $g \xleftarrow{R} \mathbb{G}$. It chooses a random $\delta \xleftarrow{R} \mathbb{Z}_p$ and sets $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), g, g^\delta)$ and $sk = (\delta, pk)$, and outputs (pk, sk).
- $\mathrm{Sign}_{sk}(m)$ for a message $m \in \mathbb{Z}_p$, the algorithm outputs $\sigma = g^{1/(\delta+m)}$ as a signature (or $\sigma = 1$ if $\delta + m = 0$).
- $\mathrm{Verify}_{pk}(m, \sigma)$ checks that $e(\sigma, g^\delta \cdot g^m) \stackrel{!}{=} e(g, g)$ and returns 1 if the equation holds (or if $\sigma = g^\delta \cdot g^m = 1$), 0 otherwise.

This signature scheme is secure against weak chosen message attacks [BB04] (i.e. attackers fix their messages for signature queries before they are given the public key g, g^δ) under the $(n+1)$-SDH assumption (where n is the maximum number of message queries an attacker may do). Formally,

Definition 2.20. Let $n \in \mathbb{N}$. Construction 2.19 has *weak unforgeability* if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{weakforge}}_{\Pi,\mathcal{A}}(\lambda) := \Pr[\mathrm{Exp}^{\mathrm{weakforge}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{weakforge}}_{\Pi,\mathcal{A}}(\lambda)$ is defined as follows:

$(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p) \leftarrow \mathfrak{G}(\lambda)$. $\mathcal{A}$ is given $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$.

$\mathcal{A}$ outputs messages $m_1, \ldots, m_n \in \mathbb{Z}_p$.

The experiment chooses $g \xleftarrow{R} \mathbb{G}, \delta \xleftarrow{R} \mathbb{Z}_p$, sets $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), g, g^\delta)$ and $sk = (\delta, pk)$.

The experiment computes $\sigma_i \leftarrow \mathrm{Sign}_{sk}(m_i)$ for $i \in [n]$ and hands $pk, \sigma_1, \ldots, \sigma_n$ to $\mathcal{A}$.

Then, $\mathcal{A}$ outputs a message m^* and a signature σ^*.

If $\mathrm{Verify}_{pk}(m^*, \sigma^*) = 1$ and $m^* \notin \{m_1, \ldots, m_n\}$, the experiment outputs 1, otherwise it outputs 0.

In the formulation of weak unforgeability given in [BB04], $\mathcal{A}$ is not even given the group $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$ before having to output messages. However, their proof does not depend on this. We choose the version above as we require it for Lemma 4.14 later. This change is also the reason that we do not define weak unforgeability for arbitrary signature schemes.

3 Accumulators

Accumulator schemes allow condensing a set V into an (short) accumulator value acc_V. For any value $i \in V$, one can efficiently compute a witness $wit_{V,i}$ that attests the fact that $i \in V$ to some verifier algorithm Verify on input $acc_V, i, wit_{V,i}$. For any value $i \notin V$, it should be computationally infeasible to produce a witness wit such that $\text{Verify}(acc_V, i, wit) = 1$.

The main advantage is that the accumulator value acc_V is typically significantly shorter than an explicit representation of the set V. As a subset of n elements, V requires the representation of V requires n bit, whereas acc_V should be sublinear (e.g., logarithmic) in n. This typically also translates to good verification time: most of the computational work can be done while creating the witness, then Verify should be fast.

We will employ accumulators for revocation. The overall idea is that some entity maintains a set V of all valid identities i, and periodically publishes the current acc_V to verifiers. Valid users can efficiently compute a witness to prove to a verifier that they were not revoked. They only need to do this once whenever the set V changes. For revoked users $i \notin V$, creating a valid witness should be computationally infeasible.

In this chapter, we will first define accumulator schemes (Section 3.1), then give a general result about the necessity of computing new witnesses whenever the accumulator changes (Section 3.2), and finally explain the accumulator construction used in this thesis (Section 3.3, [CKS09]).

3.1 Definition

Our definition of accumulator schemes is based on [DHS15] (however, in their terminology, we restrict our definition to non-universal, bounded accumulators). Note that there are different varieties of accumulator schemes in which the keys required for certain operations differ. in the following definitions, we use K as a placeholder for either pk or sk, depending on the scheme.

Definition 3.1 (Accumulator scheme)**.** An *accumulator scheme* consists of the following polynomial-time algorithms. For each of these algorithms, K is a placeholder for either pk or sk, depending on the scheme.

- $\text{Gen}(1^\lambda, 1^n)$ is a probabilistic algorithm that generates a public key pk, a secret key sk, and a public set of identities U with $|U| = n$, and returns (pk, sk, U).
- $\text{AccCreate}_K(V)$ is a (probabilistic) algorithm that, for a set $V \subseteq U$, outputs an accumulator acc.

- $\text{WitCreate}_K(V, i)$ is a deterministic algorithm that outputs a witness $wit_{V,i}$
- $\text{Verify}_{pk}(acc, i, wit)$ is a deterministic algorithm that outputs 0 or 1.

An accumulator scheme is *correct* if for all $\lambda, n \in \mathbb{N}$, all $(pk, sk, U) \in [\text{Gen}(1^\lambda, 1^n)]$, $V \subseteq U$, $i \in V$,

$$\Pr[\text{Verify}_{pk}(\text{AccCreate}_K(V), i, \text{WitCreate}_K(V, i)) = 1] = 1$$

If AccCreate_K is deterministic, we say that the accumulator scheme is *deterministic*. We say it is *probabilistic* otherwise. We also write $wit_{V,i} := \text{WitCreate}_K(V, i)$ for the witness of i with respect to V and if AccCreate is deterministic, we write $acc_V := \text{AccCreate}_K(V)$.

In this thesis, we mainly deal with an accumulator where witnesses and accumulator values can be computed from public information (i.e. $K = pk$ for both WitCreate and AccCreate). However, we offer one general statement about the nature of accumulators with respect to witness updates (Observation 3.4) that holds in the more general case.

We now define security of an accumulator scheme. Namely, it should be infeasible to compute a witness for some non-accumulated value.

Definition 3.2 (Accumulator scheme security)**.** An accumulator scheme Π is *secure (collision free)* if for all probabilistic polynomial-time algorithms $\mathcal{A}$, and all $n \in \mathbb{N}$, there exists a negligible function μ such that

$$\Pr[\text{Exp}^{\text{witforge}}_{\mathcal{A},\Pi}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where $\text{Exp}^{\text{witforge}}_{\mathcal{A},\Pi}(\lambda, n)$ is defined as follows:

> $(pk, sk, U) \leftarrow \text{Gen}(1^\lambda, 1^n)$.
>
> $\mathcal{A}$ is given pk, U, and oracle access to $\text{AccCreate}_K(\cdot)$ and $\text{WitCreate}_K(\cdot, \cdot)$.
>
> Eventually, $\mathcal{A}$ outputs a tuple (V, acc, i, wit).
>
> The experiment returns 1 if $acc \in [\text{AccCreate}_K(V)]$, $\text{Verify}_{pk}(acc, i, wit) = 1$, and $i \notin V \subseteq U$. Otherwise it returns 0.

Note that oracle access to $\text{AccCreate}_K(\cdot)$ and $\text{WitCreate}_K(\cdot, \cdot)$ is only required if $K = sk$, i.e. the scheme requires the secret key to either create an accumulator or a witness. Otherwise, $\mathcal{A}$ can compute accumulators or witnesses itself. In particular, if both accumulator values and witnesses can be created from public keys, $\mathcal{A}$ is simply given public parameters and challenged to output a forgery without any oracle access.

$\mathcal{A}$ wins the game if it can find an accumulator forgery (V, acc, i, wit) such that $i \notin V$ but acc is an accumulator for V (i.e. $acc \in [\text{AccCreate}_K(V)]$) and wit is accepted by Verify as a witness for i.

One shortcoming of accumulators is that witnesses depend on the accumulated set and hence need to be recomputed whenever the accumulator changes, which typically takes time linear in the size of the accumulated set. Also, the time for creating an accumulator is at least linear in the size of the accumulated set. As it turns out, accumulator schemes can often be defined such that adding or removing a value from the accumulator, as well as updating a witness can be done more efficiently (often in constant time/with only as many group operations as changes to the set). Accumulator schemes that support this are called *dynamic* accumulator schemes. In the following, we offer a definition for *deterministic* dynamic accumulators. It is also possible to define probabilistic dynamic accumulator schemes. However, we are only dealing with deterministic accumulators in this thesis, so we omit the details.

Definition 3.3 (Deterministic dynamic accumulator scheme)**.** A *deterministic dynamic accumulator scheme* is an deterministic accumulator scheme with the following additional algorithms. For each of these algorithms, K denotes either pk or sk, depending on the scheme.

- $\text{AccAdd}_K(acc, i)$ is a deterministic algorithm that outputs an updated accumulator acc'.
- $\text{AccDelete}_K(acc, i)$ is a deterministic algorithm that outputs an updated accumulator acc'.
- $\text{WitUpdate}_K(V, V', i, wit)$ is a deterministic algorithm that outputs an updated witness $wit_{V',i}$

We require that

$$\text{WitUpdate}_K(V, V', i, \text{WitCreate}_K(V, i)) = \text{WitCreate}_K(V', i)$$

for all $i \in V \cap V'$. Furthermore, we require that for all $i \notin V$,

$$\text{AccAdd}_K(acc_V, i) = \text{AccCreate}_K(V \cup \{i\})$$

and for all $i \in V$,

$$\text{AccDelete}_K(acc_V, i) = \text{AccCreate}_K(V \setminus \{i\})$$

Note that according to this definition, AccAdd, AccDelete, and WitUpdate can be functionally expressed by original accumulator operations and hence do not need to factor into the definitions of security and correctness.

3.2 The necessity of witness updates

The major drawback of accumulators is the need to update and distribute new witnesses whenever the accumulator changes. Hence, the question may arise whether it is possible to create an accumulator that does not suffer from this drawback. Unfortunately, the following observation establishes that witness updates are necessary for efficient accumulator constructions.

Observation 3.4 (Accumulators without witness updates)**.** Any secure accumulator scheme for n values, where $wit_{V,i} = wit_{V',i}$ for all $V, V' \subseteq U$ (i.e. witnesses do not need to be updated), requires at least n bits to encode accumulators.

Essentially, accumulators rely on computational hardness for their security – witnesses for values $i \notin V$ may exist but are hard to compute. Without witness updates, if a witness for $i \notin V$ exists, it is easy to compute as $wit_{U,i}$. Hence, such witnesses must not exist at all, which results in the lower bound of n for the accumulator length.

Proof of Observation 3.4. For convenience of notation, we assume AccCreate to be deterministic. However, the proof directly translates to the probabilistic case.

Consider an accumulator scheme without witness updates. We proceed to show that if this accumulator scheme uses at most $n-1$ bits to encode accumulators, it is insecure:

Let $(pk, sk, U) \in [\mathrm{Gen}(1^\lambda, 1^n)]$ and let $Acc := \mathrm{im}(\mathrm{AccCreate}_K)$ be the set of possible accumulators (note that $|Acc| \leq 2^{n-1}$ because by assumption, each accumulator can be represented by $n-1$ bits). We define

$$\phi : Acc \to 2^U, acc \mapsto \{i \in U \mid \mathrm{Verify}(acc, i, wit_{U,i}) = 1\}$$

to map accumulators to their implicitly accumulated sets [1]. Since $|\mathrm{im}(\phi)| \leq |Acc| \leq 2^{n-1}$, at most half of all 2^n subsets of U have a preimage. Any $V \notin \mathrm{im}(\phi)$ has no proper accumulator representation. In particular, $\phi(acc_V) \neq V$, and because the accumulator scheme is correct, $\phi(acc_V) \supset V$.

We construct an attacker $\mathcal{A}$ against the accumulator. $\mathcal{A}$ is given (pk, sk, U). It first queries $wit_{U,i}$ for all $i \in U$. Then, it chooses a set $V \subseteq U$ uniformly at random. $\mathcal{A}$ queries for acc_V and computes $\phi(acc_V)$. If $\mathcal{A}$ finds an index $i \in \phi(acc_V) \setminus V$, it returns $(V, acc_V, i, wit_{U,i})$, which is a valid witness forgery because $wit_{U,i} = wit_{V,i}$. Otherwise, $\mathcal{A}$ fails.

According to our analysis above,

$$\Pr[V \notin \mathrm{im}(\phi)] \geq 1/2$$

[1] Note that this definition is only meaningful in our setting without witness updates as $wit_{U,i}$ is not usually required to work for any accumulator acc.

and if this event occurs, then $\phi(acc_V) \supset V$ and in that case, $\mathcal{A}$ is able to find a valid witness forgery.

It follows that our probabilistic polynomial-time algorithm $\mathcal{A}$ succeeds in forging a witness with probability at least $1/2$ and hence, accumulators without witness updates and with accumulator length at most $n-1$ are not secure. □

Note that an accumulator scheme with accumulator length n can be trivially constructed by setting $acc_V := V$ (written as an n-bit vector) for any $V \subseteq \{1, \ldots, n\}$. Consequently, without witness updates, accumulator schemes are trivial.

Even though witness updates are necessary, schemes can attempt to minimize the effort needed for updating witnesses. In the next section, we describe the accumulator scheme introduced by Camenisch et al. [CKS09], which features efficient witness updates that do not rely on any secret information and hence could be carried out by untrusted third parties if necessary.

3.3 The Camenisch et al. accumulator

The following construction of a deterministic dynamic accumulator was introduced in [CKS09].

Construction 3.5 (Camenisch et al. dynamic accumulator)**.**

- $\mathrm{Gen}(1^\lambda, 1^n)$ generates a bilinear group $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$ of prime order $p > 2^\lambda$ and computes
  - $g \xleftarrow{R} \mathbb{G} \setminus \{1\}$, $\gamma \xleftarrow{R} \mathbb{Z}_p^*$
  - $z = e(g,g)^{\gamma^{n+1}}$
  - $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), (g^{\gamma^i})_{i=0; i \neq n+1}^{2n}, z)$
  - $sk = \gamma$
  - $U = \{1, \ldots, n\}$

  It returns (pk, sk, U). We write $g_i := g^{\gamma^i}$.
- $\mathrm{AccCreate}_{pk}(V)$ outputs $acc_V = \prod_{j \in V} g_{n+1-j}$
- $\mathrm{WitCreate}_{pk}(V, i)$ outputs $wit_{V,i} = \prod_{j \in V \setminus \{i\}} g_{n+1-j+i}$
- $\mathrm{Verify}_{pk}(acc, i, wit)$ outputs 1 if $e(g_i, acc)/e(g, wit) = z$
- $\mathrm{AccAdd}_{pk}(acc, i)$ outputs $acc' = acc \cdot g_{n+1-i}$
- $\mathrm{AccDelete}_{pk}(acc, i)$ outputs $acc' = acc / g_{n+1-i}$
- $\mathrm{WitUpdate}_{pk}(V, V', i, wit_{V,i})$ if $i \in V'$, outputs $wit_{V',i} = wit_{V,i} \cdot \frac{\prod_{j \in V' \setminus V} g_{n+1-j+i}}{\prod_{j \in V \setminus V'} g_{n+1-j+i}}$

In the remainder of the thesis, we may refer to i, γ^i, or g_i as an *identity* in the accumulator. We use the term interchangeably, since there is a strong relation between the three objects.

To attain a better understanding of this definition, consider the following example.

Example 3.6. Let $n = 5$ and $V = \{1, 2, 5\}$. Then the accumulator $acc_V = \prod_{j \in V} g_{n+1-j}$ $= g_1 \cdot g_4 \cdot g_5$ (cf. the green and blue fields in the first row of Figure 3.3. The number in a field represents the identity that contributes the value (e.g., the green field g_4 is included in acc_V because $2 \in V$)). We will focus on identity $i = 2$ (so that the green fields represent the entries contributed by this identity). For i, the witness $wit_{V,i}$ is $\prod_{j \in V}^{j \neq i} g_{n+1-j+i} = g_3 \cdot g_7$ (cf. the blue fields in the second row of Figure 3.3). One can view $wit_{V,i}$ as the accumulator *shifted* by i indices, excluding the secret g_{n+1}. The key to understanding the Verify algorithm is that applying the bilinear map $e(g_i, acc_V)$ allows shifting the accumulator in a similar manner as for the witnesses (but yielding a result in $\mathbb{G}_T$ and now *including* index $n + 1$). More specifically, $e(g_i, acc_V) = e(g^{\gamma^i}, \prod_{j \in V} g^{\gamma^{n+1-j}}) = \prod_{j \in V} e(g, g)^{\gamma^{n+1-j+i}} = e(g, g)^{\gamma^3} \cdot e(g, g)^{\gamma^{n+1}} \cdot e(g, g)^{\gamma^7}$ (cf. the fourth row blue and green fields in Figure 3.3).

Now, we divide the shifted accumulator $e(g_i, acc_V)$ (cf. fourth row) by the projected witness $e(g, wit_{V,i})$ (cf. third row) in $\mathbb{G}_T$. Because by construction, the two overlap at all positions but $n + 1$, the quotient yields $z = e(g, g)^{\gamma^{n+1}}$, which is exactly what Verify checks.

Intuitively, if $i \notin V$, the green fields in Figure 3.3 would disappear from the accumulator and shifted accumulator. Hence, in the quotient that is checked to be $e(g, g)^{\gamma^{n+1}}$, the index $n + 1$ is not contributed by the shifted accumulator and so it must somehow be present in the witness. However, the n-DHE assumption (Definition 2.15) states that it is hard to compute g_{n+1} in $\mathbb{G}$, so no probabilistic polynomial-time algorithm $\mathcal{A}$ should be able to create an accepting witness for $i \notin V$ (except with negligible probability).

We now formally prove correctness and security of this scheme [CKS09].

Theorem 3.7. *Construction 3.5 is a* correct *deterministic (dynamic) accumulator scheme (Definitions 3.1 and 3.3) that is* secure *(Definition 3.2) under the n-DHE assumption (Definition 2.15).*

Proof. **Correctness**

Let $(pk, sk, U) \in [\text{Gen}(1^\lambda, 1^n)], V \in U$. Let $i \in V$. We prove $\text{Verify}_{pk}(acc_V, wit_{V,i}) = 1$.

	1	2	3	4	5	$n+1$	7	8	9	$2n$
acc_V	5			2	1					
$wit_{V,i}$			5			/	1			
$e(g, wit_{V,i})$			5			/	1			
$e(g_i, acc_V)$			5			2	1			

Figure 3.1: An example illustrating Construction 3.5. Here, $n = 5, V = \{1, 2, 5\}$, and $i = 2$. The first row represents the accumulator $acc_V = g_1 \cdot g_4 \cdot g_5$. Each field is annotated with the identity that contributed the value (e.g., the green field in the first row is g_4 and it belongs to $i = 2$). The second illustrates the witness $wit_{V,i} = g_3 \cdot g_7$. The third row contains the witness projected into $\mathbb{G}_T$ and the fourth row is the shifted accumulator $e(g_i, acc_V) = e(g, g)^{\gamma^3} \cdot e(g, g)^{\gamma^{n+1}} \cdot e(g, g)^{\gamma^7}$. Note that rows 3 and 4 overlap on all positions except $n + 1$.

That is

$$\begin{aligned} & e(g_i, acc_V)/e(g, wit_{V,i}) \\ =& \frac{e(g^{\gamma^i}, \prod_{j\in V} g^{\gamma^{n+1-j}})}{e(g, \prod_{j\in V\setminus\{i\}} g^{\gamma^{n+1-j+i}})} \\ =& \frac{e(g,g)^{\sum_{j\in V} g^{\gamma^{n+1-j+i}}}}{e(g,g)^{\sum_{j\in V\setminus\{i\}} g^{\gamma^{n+1-j+i}}}} \\ =& e(g,g)^{\gamma^{n+1}} = z \end{aligned}$$

Security

From a probabilistic polynomial-time attacker $\mathcal{A}$ against the accumulator, we construct $\mathcal{A}'$ against the n-DHE assumption.

$\mathcal{A}'$ receives as input a description of the bilinear group $(\mathbb{G}, \mathbb{G}_T, e, p) \leftarrow \mathfrak{G}(\lambda)$, an element $g \xleftarrow{R} \mathbb{G} \setminus \{1\}$, $\gamma \xleftarrow{R} \mathbb{Z}_p$, and $g_i := g^{\gamma^i}$ for $i \in [2n] \setminus \{n + 1\}$. $\mathcal{A}'$ computes $z = e(g_1, g_n) = e(g, g)^{\gamma^{n+1}}$ and hands $pk = ((\mathbb{G}, \mathbb{G}_T, e, p), (g_i)_{i=1, i\neq n+1}^{2n}, z)$ and $U = [n]$ to $\mathcal{A}$.

$\mathcal{A}'$ can answer the oracle queries of $\mathcal{A}$ using pk [2].

Eventually, $\mathcal{A}$ outputs (V, acc, i, wit). If $i \notin V, acc = acc_V$, and $\text{Verify}_{pk}(acc, i, wit) =$

[2]Indeed, we may assume that $\mathcal{A}$ does not query the oracles as it could easily compute the results itself.

1 (i.e. $e(g_i, acc_V)/e(g, wit) = e(g, g)^{\gamma^{n+1}}$), then $\mathcal{A}'$ computes and outputs

$$g^{\gamma^{n+1}} = g_{n+1} \stackrel{!}{=} \frac{\prod_{j \in V} g_{n+1-j+i}}{wit} \tag{3.1}$$

the last equality holds because

$$\begin{aligned}
& e(g_i, acc_V)/e(g, wit) = e(g, g)^{\gamma^{n+1}} \\
\Leftrightarrow & e(g^{\gamma^i}, acc_V) \cdot e(g, wit^{-1}) = e(g, g^{\gamma^{n+1}}) \\
\Leftrightarrow & e(g, acc_V^{\gamma^i}) \cdot e(g, wit^{-1}) = e(g, g^{\gamma^{n+1}}) \\
\Leftrightarrow & e(g, (\prod_{j \in V} g_{n+1-j})^{\gamma^i}) \cdot e(g, wit^{-1}) = e(g, g^{\gamma^{n+1}}) \\
\Leftrightarrow & e(g, \prod_{j \in V} g_{n+1-j+i}) \cdot e(g, wit^{-1}) = e(g, g^{\gamma^{n+1}}) \\
\Leftrightarrow & e(g, \prod_{j \in V} g_{n+1-j+i}/wit) = e(g, g^{\gamma^{n+1}}) \\
\Leftrightarrow & \prod_{j \in V} g_{n+1-j+i}/wit = g^{\gamma^{n+1}}
\end{aligned}$$

using that all groups involved are prime order groups and e is non-degenerate. $\mathcal{A}$ can compute the right hand side value of (3.1) because $i \notin V$ and so the unknown value g_{n+1} is not part of the numerator's product (all other $g_{n+1-j+i}$ are given by n-DHE).

$\mathcal{A}'$ outputs $g^{\gamma^{n+1}}$ if and only if $\mathcal{A}$ succeeds in computing an accumulator forgery. From the n-DHE assumption, it follows that the probability for $\mathcal{A}$ to compute an accumulator forgery must be negligible in the security parameter. □

3.4 Anonymously proving inclusion in the accumulator

The goal of this thesis is to explore applications of accumulators for revocation in schemes that feature anonymity, such as group signatures or credential systems. For this, it is imperative that inclusion in the accumulator can be demonstrated without revealing the accumulated identity or the witness (since in these schemes, either would identify the user).

This can be solved through an interactive protocol to prove knowledge of an accumulator witness $wit_{V,i}$ that certifies that i is accumulated in acc_V without revealing $wit_{V,i}$ or i. Indeed, such a protocol exists:

Construction 3.8. The prover has an identity g_i and a witness wit for Construction 3.5. We also assume some publicly known random generator $\tilde{h} \in \mathbb{G} \setminus \{1\}$. The prover chooses random $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$ and computes

- a blinded identity $\mathcal{G} = g_i \tilde{h}^{r_1}$
- a blinded witness $\mathcal{W} = wit \cdot \tilde{h}^{r_2}$

and sends these values to the verifier. Then, the prover engages the verifier in the following proof:

$$PK\{(r_1, r_2) : e(\mathcal{G}, acc_V)/e(g, \mathcal{W}) = z \cdot e(\tilde{h}, acc_V)^{r_1}/e(g, \tilde{h})^{r_2}\}$$

Through rearranging of terms, on can see that a witness (r_1, r_2) of the $PK\{\dots\}$ protocol allows derandomizing $\mathcal{G}$ and $\mathcal{W}$ to values $g^* := \mathcal{G}\tilde{h}^{-r_1}, wit^* := \mathcal{W}\tilde{h}^{-r_2}$ that fulfill the accumulator verification equation $e(g^*, acc_V)/e(g, wit^*) = z$.

However, this protocol is not very useful on its own. Because g_i is completely hidden to the verifier, a revoked user i may simply choose some $g_{i'}$ where $i' \in V$ and run the protocol with $wit_{V,i'}$. Hence, schemes that use this protocol will need to tie g_i to some other value (that only user i would know). For example, this tie can be a signature on g_i that is only given to user i. This is further discussed at the example of building a ring signature scheme from the accumulator in Section 4.1.1.

4 A ring signature scheme from the Camenisch et al. accumulator

In this chapter, we will construct a ring signature scheme (Definition 2.8). The ring of signers V for a signature will be internally represented by an accumulator value acc_V (Construction 3.5). This construction serves as a demonstration of how to build anonymity-preserving protocols using the accumulator. Furthermore, the ring signature scheme will serve as a basis for group signatures with accumulator revocation.

The overall idea of our proposed scheme is the following, using terminology from the accumulator (Chapter 3). User i receives a signature on "his" accumulator value g_i as a secret key. We design a protocol for a user to prove knowledge of such a signature on some g_i and an accumulator witness that certifies accumulation of that particular i. A signature for a set V and user $i \in V$ is then created by computing the corresponding accumulator acc_V and a witness $wit_{V,i}$ from public information, then creating a signature of knowledge using the Fiat-Shamir heuristic on the protocol.

Hence, we require

- a proof protocol for an accumulator witness (already covered in Section 3.4)
- a signature scheme for signing g_i that has an efficient protocol for proving knowledge of a signature without revealing g_i (Section 4.1.1)
- the composition of both protocols (Section 4.1.2)
- construction and security proof of the ring signature scheme derived from the combined protocol and the Fiat-Shamir heuristic (Section 4.2)

4.1 A protocol to prove revocation status

In this section, we describe the protocol that can be used to demonstrate (1) knowledge of an identity i included in the accumulator, and (2) knowledge of a signature on i. First, we will explain the signature scheme used for signing accumulator identities and a protocol for proving knowledge of such signatures (Section 4.1.1). Then, we will compose the two protocols and prove the properties of that protocol that will be used for security of the ring signature scheme that we construct (Section 4.1.2).

4.1.1 Signing accumulator identities

As described in Section 3.4, the proof protocol for knowledge of an accumulated value is not particularly useful in itself: No information is leaked about what value i is proven

to be accumulated. This is a desirable property for our ring signature scheme since we want to prove inclusion of the signer's identity in the accumulator without revealing said identity. However, since no information is leaked about the signer, user i may simply prove inclusion of some $i' \in V$, regardless of its own inclusion in the accumulator. For this reason, we need to bind the value i used in the accumulator proof to the actual signer's identity. We do this by giving each user i a signature σ_i on the accumulator identity as part of their secret key during setup. In this setting, the user proves knowledge of some value i that is (1) included in the accumulator, and (2) signed with σ_i. This section deals with choosing a suitable signature scheme.

For the sake of clarity, we already define the public and secret keys for the ring signature scheme and reference these variables throughout this section. The keys involved in our ring signature scheme are of the following form:

- $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), (g^{\gamma^i})_{i=0;i\neq n+1}^{2n}, z, u, g^\delta, h, \tilde{h})$, where $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$, $(g^{\gamma^i})_{i=0;i\neq n+1}^{2n}$, z are public parameters for the accumulator (Construction 3.5), $u, h \xleftarrow{R} \mathbb{G} \setminus \{1\}$ and $\delta \xleftarrow{R} \mathbb{Z}_p$ for the signatures on accumulator identities (Section 4.1.1), and $\tilde{h} \xleftarrow{R} \mathbb{G} \setminus \{1\}$ for commitments.
- $sk[i] = (i, g_i, u_i, \sigma_i, pk)$, where $g_i = g^{\gamma^i}$ as in the accumulator construction, $u_i = u^{\gamma^i}$ (for n-HSDHE, as discussed later), and $\sigma_i = h^{1/(\delta+\gamma^i)}$ (a signature on γ^i, cf. Section 4.1.1).

We identified the following informal requirements for a suitable signature scheme signing accumulator identities, which will be briefly explained:

- The message space should be $\mathcal{M} = \mathbb{G}$ (for signing $g_i = g^{\gamma^i}$) or $\mathcal{M} = \mathbb{Z}_p$ (for signing γ^i)[1].
- Since only a fixed number of messages g_i or γ^i are securely signed in the setup phase, a *weakly* secure signature scheme suffices (cf. Definition 2.20).
- There must exist an efficient proof protocol demonstrating knowledge of a hidden signature on a hidden message.
- That protocol (previous point) must not require user's knowledge of γ^i but only g^{γ^i} [2].
- Without a signature, the protocol should be hard to pass for a prover. For $\mathcal{M} = \mathbb{G}$, this is easily achieved by requiring that the protocol is a proof of knowledge of g_i and a signature on g_i. In the case that $\mathcal{M} = \mathbb{Z}_p$, however, this is potentially problematic. Since the prover must not learn the signed message $m = \gamma^i$, we

[1] As an alternative to signing γ^i, one may consider signing $i \in \mathbb{Z}_p$. However, the complex relationship between g_i used in the accumulator proof and the signed i ($g_i = g^{\gamma^i}$ for secret γ) would make linking i to g_i prohibitively difficult.

[2] Knowing i and γ^i would allow a user to compute $\gamma = (\gamma^i)^{1/i}$, which allows creating accumulator witnesses for arbitrary values

cannot expect to be able to extract m from a successful prover, which we need to claim a proper signature forgery. As a consequence, the security of the protocol cannot be directly reduced to the security of the signature scheme.

- The proof protocol should be "and"-composable with the accumulator proof protocol (Construction 3.8) such that the identity used there and the signature's message are properly connected.

This list of requirements indicates that it would be preferable to use a signature scheme with message space $\mathcal{M} = \mathbb{G}$ as this sidesteps the problems associated with users not having access to γ^i. However, we were unable to find such a signature scheme. For this reason, we follow the constructions in [CKS09] and use a variant of weakly-secure Boneh-Boyen signatures (Construction 2.19). The message space of the weakly-secure Boneh-Boyen signature scheme is $\mathbb{Z}_p$ and it signs γ^i as $\sigma_i := h^{1/(\delta+\gamma^i)}$ with the secret key δ.

In the following, we will state the required protocol for demonstrating knowledge of the ring signature scheme's secret key $sk[i]$. Afterwards, we will explain the rationale of the proof and how the problems associated with signing γ^i instead of g_i are solved through it.

Consider the following proof protocol used to demonstrate knowledge of a secret key (slightly modified from [CKS09]):

Construction 4.1. Given $sk[i] = (i, g_i, u_i, \sigma_i, pk)$, the prover chooses random r_1, r_3, r_4, $open \xleftarrow{R} \mathbb{Z}_p$ and computes

- the blinded identities $\mathcal{G} = g_i \tilde{h}^{r_1}$ and $\mathcal{U} = u_i \tilde{h}^{r_4}$
- the blinded signature $S = \sigma_i \tilde{h}^{r_3}$
- the commitment $B = g^{r_1} \tilde{h}^{open}$

and sends these values to the verifier. Then, the prover sets $mult = r_1 r_3, tmp = open \cdot r_3$ and engages the verifier in the following proof:

$$
\begin{aligned}
PK\{&(r_1, r_3, r_4, open, mult, tmp) : \\
&e(\mathcal{G}, u)/e(g, \mathcal{U}) = e(\tilde{h}, u)^{r_1} e(g, \tilde{h})^{-r_4} && (4.1)\\
&\wedge e(g^\delta \cdot \mathcal{G}, S) = e(g, h) \cdot e(g^\delta \cdot \mathcal{G}, \tilde{h})^{r_3} \cdot e(\tilde{h}, \tilde{h})^{-mult} \cdot e(\tilde{h}, S)^{r_1} && (4.2)\\
&\wedge B = g^{r_1} \tilde{h}^{open} \wedge B^{r_3} = g^{mult} \tilde{h}^{tmp} && (4.3)\\
\}&
\end{aligned}
$$

Roughly speaking[3], in this protocol, (4.1) ensures that $\mathcal{G}$ and $\mathcal{U}$ derandomize to values

[3] For details, consult Lemma 4.4 which deals with the composition of this protocol with the accumulator value proof.

$g^* = \mathcal{G}\tilde{h}^{-r_1}, u^* = \mathcal{U}\tilde{h}^{-r_4}$ such that $\exists x \in \mathbb{Z}_p : \ g^* = g^x \wedge u^* = u^x$ (i.e. they have the same discrete logarithm). (4.3) ensures that $mult = r_1 r_3$ (assuming the discrete logarithm of $\tilde{h}$ is hard to compute). Finally, (4.2) ensures that $\sigma^* := S\tilde{h}^{-r_3}$ is a valid signature on the discrete logarithm of g^* (i.e. $e(g^\delta \cdot g^*, \sigma^*) = e(g, h)$).

Note that a witness in this protocol does not contain the signed message γ^i of σ_i (as discussed above, this is impossible to expect since γ^i must be kept secret). It does, however, allow computing values g^*, u^* and a signature $\sigma^* = h^{1/(m^*+\delta)}$ on the common discrete logarithm m^* of g^* and u^*. This does not allow extracting a signature forgery on m^* (as m^* cannot be efficiently computed from g^* or u^* by assumption). However, assume we know the discrete logarithm ℓ of $g = h^\ell$. Then we *can* compute an n-HSDHE tuple (Definition 2.17) from the result, namely $(g^{1/(\delta+m^*)}, g^{m^*}, u^{m^*})$, where $g^{1/(\delta+m^*)}$ can be computed as $(\sigma^*)^\ell$.

Loosely interpreted, the protocol shows knowledge of a signature σ on some message m where a prover forging a new signature does not necessarily know m, but was able to compute g^m and u^m for two unrelated bases g, u.

Overall, we have found a way to sign the secret γ^i such that there exists a proof protocol that does not depend on γ^i but only g^{γ^i}.

4.1.2 Combining the signature and accumulator protocol

We now state the protocol that combines Construction 3.8 (which guarantees knowledge of an accumulator witness for some potential identity g^*) and Construction 4.1 (which guarantees that g^* is the user's g_i by proving knowledge of a signature on g^*). The composition itself is straight-forward: the blinded values in the beginning are the union of the ones in the respective protocols and the constraints in the subsequent Σ protocol are composed by conjunction. The link between the two protocols is the shared $\mathcal{G}$ (blinded accumulator identity) and its derandomization value r_1.

Construction 4.2 (Proof of knowledge of a signature on an accumulated identity)**.** Let $pk, (sk[i])_{i=1}^n$ be as defined in the beginning of this chapter. To prove knowledge of a secret key $sk[i]$ and a witness wit for inclusion of g_i in an accumulator acc, the prover chooses random $r_1, r_2, r_3, r_4, open \xleftarrow{R} \mathbb{Z}_p$ and computes

- a blinded identity $\mathcal{G} = g_i\tilde{h}^{r_1}$ and $\mathcal{U} = u_i\tilde{h}^{r_4}$
- a blinded witness $\mathcal{W} = wit \cdot \tilde{h}^{r_2}$
- a blinded signature $S = \sigma_i\tilde{h}^{r_3}$
- a commitment $B = g^{r_1}\tilde{h}^{open}$

and sends these values to the verifier. Then, the prover sets $mult = r_1 r_3, tmp = open \cdot r_3$ and engages the verifier in the following proof protocol, which we will call Σ in the

remainder of this chapter.

$$\begin{aligned}\Sigma = PK\{&(r_1, r_2, r_3, r_4, open, mult, tmp) : \\ &e(\mathcal{G}, u)/e(g,\mathcal{U}) = e(\tilde{h}, u)^{r_1} e(g, \tilde{h})^{-r_4} \\ &\wedge e(g^\delta \cdot \mathcal{G}, S) = e(g, h) \cdot e(g^\delta \cdot \mathcal{G}, \tilde{h})^{r_3} \cdot e(\tilde{h}, \tilde{h})^{-mult} \cdot e(\tilde{h}, S)^{r_1} \\ &\wedge B = g^{r_1} \tilde{h}^{open} \wedge B^{r_3} = g^{mult} \tilde{h}^{tmp} \\ &\wedge e(\mathcal{G}, acc)/e(g, \mathcal{W}) = z \cdot e(\tilde{h}, acc)^{r_1}/e(g, \tilde{h})^{r_2} \\ \}&\end{aligned}$$

For the sake of clarity when applying the Fiat-Shamir heuristic, we now explicitly apply Construction 2.5 to Σ and state the overall protocol with all details: Let

$$\Phi^{(1)}_{pk}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, t_1, \ldots, t_7) := (T_1, \ldots, T_5)$$

with

- $T_1 = e(\tilde{h}, u)^{t_1} \cdot e(g, \tilde{h})^{-t_4}$
- $T_2 = e(g^\delta \cdot \mathcal{G}, \tilde{h})^{t_3} \cdot e(\tilde{h}, \tilde{h})^{-t_6} \cdot e(\tilde{h}, S)^{t_1}$
- $T_3 = g^{t_1} \cdot \tilde{h}^{t_5}$
- $T_4 = g^{t_6} \cdot \tilde{h}^{t_7} \cdot B^{-t_3}$
- $T_5 = e(\tilde{h}, acc)^{t_1}/e(g, \tilde{h})^{t_2}$

and

$$\Phi^{(2)}_{pk}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C}) := (T_1, \ldots, T_5)$$

with

- $T_1 = e(\tilde{h}, u)^{s_1} \cdot e(g, \tilde{h})^{-s_4} \cdot (e(\mathcal{G}, u)/e(g,\mathcal{U}))^{-\mathcal{C}}$
- $T_2 = e(g^\delta \cdot \mathcal{G}, \tilde{h})^{s_3} \cdot e(\tilde{h}, \tilde{h})^{-s_6} \cdot e(\tilde{h}, S)^{s_1} \cdot (e(g^\delta \cdot \mathcal{G}, S)/e(g, h))^{-\mathcal{C}}$
- $T_3 = g^{s_1} \cdot \tilde{h}^{s_5} \cdot B^{-\mathcal{C}}$
- $T_4 = g^{s_6} \cdot \tilde{h}^{s_7} \cdot B^{-s_3}$
- $T_5 = e(\tilde{h}, acc)^{s_1}/e(g, \tilde{h})^{s_2} \cdot (e(\mathcal{G}, acc_V)/e(g, \mathcal{W})/z)^{-\mathcal{C}}$

Then the protocol works as follows:

Prover $\mathcal{P}$ on input $sk[i] = (i, g_i, u_i, \sigma_i, pk), acc, wit$		Verifier $\mathcal{V}$ on input pk, acc
choose $r_1 \ldots, r_4, open \xleftarrow{R} \mathbb{Z}_p$ set $\mathcal{G} = g_i \tilde{h}^{r_1}$, $\mathcal{U} = u_i \tilde{h}^{r_4}$, $\mathcal{W} = wit \cdot \tilde{h}^{r_2}$, $S = \sigma_i \tilde{h}^{r_3}$, $B = g^{r_1} \tilde{h}^{open}$ $mult = r_1 \cdot r_3, tmp = open \cdot r_3$ choose $(t_1, \ldots, t_7) \xleftarrow{R} \mathbb{Z}_p^7$ $(T_1, \ldots, T_5) := \Phi_{pk}^{(1)}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, t_1, \ldots, t_7)$		
	$\xrightarrow{(\mathcal{G},\mathcal{U},\mathcal{W},S,B,T_1,\ldots,T_5)}$	
		$\mathcal{C} \xleftarrow{R} \mathbb{Z}_p$
	$\xleftarrow{\mathcal{C}}$	
calculate $s_1 = r_1 \cdot \mathcal{C} + t_1$ $s_2 = r_2 \cdot \mathcal{C} + t_2$ $s_3 = r_3 \cdot \mathcal{C} + t_3$ $s_4 = r_4 \cdot \mathcal{C} + t_4$ $s_5 = open \cdot \mathcal{C} + t_5$ $s_6 = mult \cdot \mathcal{C} + t_6$ $s_7 = tmp \cdot \mathcal{C} + t_7$		
	$\xrightarrow{s_1,\ldots,s_7}$	
		accept if $\Phi_{pk}^{(2)}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$ $\stackrel{!}{=} (T_1, \ldots, T_5)$

The correctness of this protocol follows from the fact that Σ derived from Construction 2.5 is correct. For more details, consult the proof of Lemma 4.10 (which shows correctness of the signature version of this protocol), as it also directly applies to this protocol.

Note that this construction is *not* formally a Σ-protocol because the prover takes private input $sk[i], wit$ but proves knowledge of $(r_1, r_2, r_3, r_4, open, mult, tmp)$ – and the two sets of witnesses turn out not to be of equal size, which rules out a one-to-one correspondence of witnesses that could be used to convert the protocol to a Σ-protocol. The reason behind this is that the construction to guarantee $mult = r_1 \cdot r_3$ through

$B = g^{r_1}\tilde{h}^{open} \wedge B^{r_3} = g^{mult}\tilde{h}^{tmp}$ can be circumvented by breaking the discrete logarithm problem[4]. This gives rise to witnesses in the inner Σ protocol that do not correspond to valid $sk[i], wit$.

However, the protocol still has the necessary properties to apply the Fiat-Shamir heuristic: it can be simulated (by choosing random $\mathcal{G},\mathcal{U},\mathcal{W},S,B \xleftarrow{R} \mathbb{G}$ and then running the simulator for Σ) and without knowing $sk[i], wit$, it is hard to pass the protocol. The following lemma motivates the latter property (corresponding to special soundness).

Lemma 4.3. *Given two accepting transcripts* $((\mathcal{G},\mathcal{U},\mathcal{W},S,B,T_1,\ldots,T_5),\mathcal{C},(s_1,\ldots,s_7))$ *and* $((\mathcal{G},\mathcal{U},\mathcal{W},S,B,T_1,\ldots,T_5),\mathcal{C}',(s'_1,\ldots,s'_7))$ *where* $\mathcal{C} \neq \mathcal{C}'$*, one can efficiently compute an element*

$$((pk, acc, \mathcal{G},\mathcal{U},\mathcal{W},S,B),(r_1,r_2,r_3,r_4,open,mult,tmp)) \in R$$

where R *is the NP-relation of* Σ *in Construction 4.2, i.e.*

$$\begin{aligned} R = \{&((pk, acc, \mathcal{G},\mathcal{U},\mathcal{W},S,B),(r_1,r_2,r_3,r_4,open,mult,tmp)) \in (\{pk\} \times \mathbb{G}^6) \times \mathbb{Z}_p^7 : \\ &\wedge e(\mathcal{G},u)/e(g,\mathcal{U}) = e(\tilde{h},u)^{r_1}e(g,\tilde{h})^{-r_4} \\ &\wedge e(g^\delta \cdot \mathcal{G},S) = e(g,h) \cdot e(g^\delta \cdot \mathcal{G},\tilde{h})^{r_3} \cdot e(\tilde{h},\tilde{h})^{-mult} \cdot e(\tilde{h},S)^{r_1} \\ &\wedge B = g^{r_1}\tilde{h}^{open} \wedge B^{r_3} = g^{mult}\tilde{h}^{tmp} \\ &\wedge e(\mathcal{G},acc)/e(g,\mathcal{W}) = z \cdot e(\tilde{h},acc)^{r_1}/e(g,\tilde{h})^{r_2} \\ \}& \end{aligned}$$

Proof. Follows immediately from the special soundness property of Σ: given the two transcripts as above, run the extractor of Σ (cf. Lemma 2.5) on input $(pk, acc, \mathcal{G},\mathcal{U},\mathcal{W},S,B)$ and the two transcripts $((T_1,\ldots,T_5),\mathcal{C},(s_1,\ldots,s_7))$ and $((T_1,\ldots,T_5),\mathcal{C}',(s'_1,\ldots,s'_7))$ (which are accepting by construction). □

The following lemma explains the protocol closer by demonstrating that extracted witnesses allow computing interesting values, which we will later use to argue that provers can break one of the schemes involved.

Lemma 4.4. *Given*

$$((pk, acc, \mathcal{G},\mathcal{U},\mathcal{W},S,B),(r_1,r_2,r_3,r_4,open,mult,tmp)) \in R$$

with R *like in Lemma 4.3, one can efficiently compute either*

- *the discrete logarithm* l^* *of* $\tilde{h} = g^{l^*}$

or the following

1. *a signature* σ^* *with* $e(g^\delta \cdot g^*,\sigma^*) = e(g,h)$ *for some* $g^* \in \mathbb{G}$

[4]Details can be found in the upcoming Lemma 4.4.

2. u^* *with* $\exists x \in \mathbb{Z}_p : \ g^* = g^x = u^x = u^*$

3. *a witness* wit^* *with* $e(g^*, acc)/e(g, wit^*) = z$

Proof. Given $((pk, acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, r_2, r_3, r_4, open, mult, tmp)) \in R$, we distinguish two cases:

- If $mult \neq r_1 \cdot r_3$, then we can compute the discrete logarithm $l^* = (r_1 \cdot r_3 - mult)/(tmp - open \cdot r_3)$ of $\tilde{h}$ to base g. From a valid witness we get that $g^{r_1 \cdot r_3} \tilde{h}^{open \cdot r_3} = g^{mult} \tilde{h}^{tmp}$ and hence $\tilde{h} = g^{(r_1 \cdot r_3 - mult)/(tmp - open \cdot r_3)}$ (note that $tmp - open \cdot r_3 \neq 0$ because $r_1 \cdot r_3 - mult \neq 0$ and $g, \tilde{h}$ are generators).
- If $mult = r_1 \cdot r_3$, then we can compute derandomized values $g^* := \mathcal{G} \cdot \tilde{h}^{-r_1}$, $u^* = \mathcal{U}\tilde{h}^{-r_4}$, $wit^* := \mathcal{W} \cdot \tilde{h}^{-r_2}$, $\sigma^* := S\tilde{h}^{-r_3}$ such that the required equations are fulfilled:
  1. In a valid witness for Σ, it holds that $e(g^\delta \cdot \mathcal{G}, S) = e(g, h) \cdot e(g^\delta \cdot \mathcal{G}, \tilde{h})^{r_3} \cdot e(\tilde{h}, \tilde{h})^{-mult} \cdot e(\tilde{h}, S)^{r_1}$, hence $e(g^\delta \cdot g^*, \sigma^*) = e(g, h)$ (using $mult = r_1 \cdot r_3$).
  2. In a valid witness for Σ, it holds that $e(\mathcal{G}, u)/e(g, \mathcal{U}) = e(\tilde{h}, u)^{r_1} e(g, \tilde{h})^{-r_4}$, so for $x \in \mathbb{Z}_p$ with $g^* = g^x$ we have $e(g, u^*) = e(g^*, u) = e(g^x, u) = e(g, u^x)$ and hence $u^* = u^x$.
  3. In a valid witness for Σ, $e(\mathcal{G}, acc)/e(g, \mathcal{W}) = z \cdot e(\tilde{h}, acc)^{r_1}/e(g, \tilde{h})^{r_2}$, which implies $e(g^*, acc)/e(g, wit^*) = z$.

□

Finally, we note that our (composed) protocol (Construction 4.2) can be simulated for any given $\mathcal{C} \in \mathbb{Z}_p$: One simply chooses $\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B \xleftarrow{R} \mathbb{G}$ randomly and then runs the simulator for the inner protocol Σ, i.e. choose random $s_1, \ldots, s_7 \xleftarrow{R} \mathbb{Z}_p$ and compute the missing $T_1, \ldots, T_5$ of the first message as $\Phi^{(2)}_{pk}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$.

4.2 Constructing the ring signature scheme

4.2.1 Construction

We now introduce our ring signature scheme that is based on the accumulator. The scheme is built using the Fiat-Shamir heuristic on the protocol above (Construction 4.2), which proves knowledge of a secret key where the identity is accumulated.

Note that in order to guarantee unforgeability, we require that to set up the ring signature scheme for n' users, the underlying accumulator is set up for $n := \frac{n'(n'+1)}{2}$ users. Because the accumulator scheme is secure for any number of users, this does affect asymptotic guarantees. We refer to the discussion at the end of this chapter for the reason behind this change and to Observation 4.6 for how this affects the size of the public parameters.

Construction 4.5 (Accumulator-based ring signature scheme)**.**

- KeyGen$(1^\lambda, 1^{n'})$ sets $n := \frac{n'(n'+1)}{2}$ and does the following:
  - Run Gen$(1^\lambda, 1^n)$ from the Camenisch et al. accumulator (Construction 3.5) to obtain $pk_{acc} = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), (g^{\gamma^i})_{i=0;i\neq n+1}^{2n}, z)$ and $sk_{acc} = \gamma$.
  - Pick $\delta \xleftarrow{R} \mathbb{Z}_p$ as the secret key for signatures on identities (Construction 2.19), $h \xleftarrow{R} \mathbb{G} \setminus \{1\}$ as the base for the signatures, and $u \xleftarrow{R} \mathbb{G} \setminus \{1\}$ for the reduction to n-HSDHE as discussed in Section 4.1.1.
  - For every $i \in \{1, \ldots, n'\}$, compute a signature $\sigma_i := h^{1/(\delta+\gamma^i)}$ on γ^i and compute $g_i := g^{\gamma^i}$ and $u_i := u^{\gamma^i}$.
  - Choose a hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$, which we model as a random oracle, and an additional $\tilde{h} \xleftarrow{R} \mathbb{G} \setminus \{1\}$ for commitments in the underlying protocol.
  - $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), (g^{\gamma^i})_{i=0;i\neq n+1}^{2n}, z, u, g^\delta, h, \tilde{h})$.
  - $U = [n']$.
  - $sk[i] = (i, g_i, u_i, \sigma_i, pk)$ for each $i \in U$.
  - Output $(pk, U, (sk[i])_{i\in U})$.
- $\mathrm{Sign}_{sk[i]}(m, V)$ signs m according to the Fiat-Shamir heuristic on Construction 4.2:
  - Compute the accumulator $acc_V = \prod_{j\in V} g_{n+1-j}$ and the witness $wit_{V,i} = \prod_{j\in V\setminus\{i\}} g_{n+1-j+i}$ (using public information).
  - Compute the first part of the first message of the protocol: Choose random $r_1 \ldots, r_4, open \xleftarrow{R} \mathbb{Z}_p$ and set $\mathcal{G} = g_i\tilde{h}^{r_1}$, $\mathcal{U} = u_i\tilde{h}^{r_4}$, $\mathcal{W} = wit_{V,i} \cdot \tilde{h}^{r_2}$, $S = \sigma_i\tilde{h}^{r_3}$, $B = g^{r_1}\tilde{h}^{open}$. Then set $mult = r_1 \cdot r_3$ and $tmp = open \cdot r_3$.
  - Compute the second part of the first protocol message: Choose $(t_1, \ldots, t_7) \xleftarrow{R} \mathbb{Z}_p^7$ and calculate $\Phi_{pk}^{(1)}(acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, t_1, \ldots, t_7) := (T_1, \ldots, T_5)$.
  - Compute the challenge $\mathcal{C} = \mathcal{H}(m, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, T_1, \ldots, T_5)$.
  - Compute the response of the protocol: $s_1 := r_1 \cdot \mathcal{C} + t_1$, $s_2 := r_2 \cdot \mathcal{C} + t_2$, $s_3 := r_3 \cdot \mathcal{C} + t_3$, $s_4 := r_4 \cdot \mathcal{C} + t_4$, $s_5 := open \cdot \mathcal{C} + t_5$, $s_6 := mult \cdot \mathcal{C} + t_6$, $s_7 := tmp \cdot \mathcal{C} + t_7$.
  - The signature is $(\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$.
- $\mathrm{Verify}_{pk}(m, V, (\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C}))$ checks the signature of knowledge as follows:
  - Compute $acc_V = \prod_{j\in V} g_{n+1-j}$ (from public information).
  - Recreate the commitment $(T_1, \ldots, T_5) := \Phi_{pk}^{(2)}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$.
  - Output 1 if $\mathcal{C} = \mathcal{H}(m, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, T_1, \ldots, T_5)$, otherwise output 0.

In this construction, it may seem like the size of the public parameters suffers from our need to set up the accumulator scheme for $n = n'(n'+1)/2$ users instead of only $n' = |U|$ users.

Observation 4.6. The public key can be shortened to

$$pk = ((\mathbb{G}, \mathbb{G}_T, e, p), \underline{(g^{\gamma^i})_{i=n+1-n';i\neq n+1}^{n+n'}}, z, u, g^{\delta}, h, \tilde{h})$$

The observation follows immediately because only the g_i where $i \in [n+1-n', n+n'] \setminus \{n+1\}$ are used to carry out the Sign and Verify operations. With this change, we bring down the size of pk by reducing the number of g_i elements from $2n-1$ to merely $2n'-1$ group elements. The construction remains obviously secure (since any attacker against the reduced-key scheme can be trivially used to attack the full-key scheme). We state the ring signature scheme with full $(g_i)_{i\in[2n]\setminus\{n+1\}}$ because it makes the relationship of the accumulator and the ring signature scheme slightly more explicit in the security proofs.

We also make the following observation.

Observation 4.7. Signatures of Construction 4.5 can be created and verified without knowledge of V as long as acc_V and (for signing) $wit_{V,i}$ are known.

As a consequence, one may also define the signing and verifying operations with accumulator parameters, i.e. $\text{Sign}_{sk[i]}(m, acc_V, wit_{V,i})$ and $\text{Verify}_{pk}(m, acc_V, \sigma)$.

The observation follows directly from the fact that V is only used in Sign and Verify to compute acc_V (and $wit_{V,i}$). Later, we will use this fact to embed the scheme in a context where the set V is not necessarily known to signers and verifiers. There, we also prove that the ring signature scheme is still unforgeable in a slightly modified sense if Sign operates directly on the accumulator instead of V (Definition 5.12 and Lemma 5.13).

Now that we established the special properties of this construction, we will proceed to prove correctness, anonymity against early full key exposure, and unforgeability.

Theorem 4.8. *Construction 4.5 is a* correct *ring signature scheme (Definition 2.8) that fulfills* anonymity *(Definition 2.9) in the random oracle model, and has* unforgeable signatures *(Definition 2.10) under the n-HSDHE assumption and n-DHE assumption in the random oracle model.*

Proof. The proof is divided into several parts and results from Lemma 4.10 (correctness), Lemma 4.11 (anonymity), and Lemma 4.14 (unforgeability). □

For the remainder of the chapter, for $\sigma = (\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$, we define functions Ψ_1, Ψ_2, Ψ_3 corresponding to the first, second, and third message in the underlying

protocol (Construction 4.2), respectively.

- $\Psi_1(\sigma, acc, pk) := (\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, T_1, \ldots, T_5)$, where $(T_1, \ldots, T_5) = \Phi_{pk}^{(2)}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$
- $\Psi_2(\sigma) := \mathcal{C}$
- $\Psi_3(\sigma) := (s_1, \ldots, s_7)$

If pk is unambiguous in the current context, we may simply write $\Psi_1(\sigma, acc)$ instead of $\Psi_1(\sigma, acc, pk)$.

The following observation establishes a link between signatures and the underlying protocol.

Observation 4.9. For any $\sigma \in \mathbb{G}^5 \times \mathbb{Z}_p^7 \times \mathbb{G}$ and $V \subseteq U$, $(\Psi_1(\sigma, acc_V), \Psi_2(\sigma), \Psi_3(\sigma))$ is an accepting transcript in the underlying protocol (Construction 4.2).

This observation follows immediately from the fact that the protocol's acceptance check $\Phi_{pk}^{(2)}(acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C}) \stackrel{!}{=} (T_1, \ldots, T_5)$ is fulfilled simply by definition of Ψ_1 (which computes $(T_1, \ldots, T_5)$ using $\Phi_{pk}^{(2)}$).

4.2.2 Correctness

We now begin proving the lemmas required for Theorem 4.8, starting with correctness, which essentially follows from correctness of the underlying protocol (Construction 4.2), which in turn follows from correctness of our Σ protocols (Construction 2.5).

Lemma 4.10. *Construction 4.5 is* correct.

Proof. Consider the signature $\sigma = (\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$ created by the Sign algorithm for m and V, with secret key $sk[i]$ and random values $r_1, \ldots, r_4, open, t_1, \ldots, t_7 \in \mathbb{Z}_p$. We will show that Verify recreates the same $T_1, \ldots, T_5$ that Sign used. Formally, $(T_1', \ldots, T_5') := \Phi_{pk}^{(2)}(acc, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C}) \stackrel{!}{=} \Phi_{pk}^{(1)}(acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, t_1, \ldots, t_7) =: (T_1, \ldots, T_5)$. Then correctness follows immediately (since then both Sign and Verify compute the same hash value, which is exactly what Verify checks).

Note that there is a close relation between the following equations and correctness of the used Schnorr protocol variant (Construction 2.5). However, we still need to argue that, for example, the accumulator check equation in the proof protocol is indeed satisfied by the accumulator witness that Sign computes.

$$
\begin{aligned}
T_1' =& e(\tilde{h}, u)^{s_1} \cdot e(g, \tilde{h})^{-s_4} \cdot (e(\mathcal{G}, u)/e(g, \mathcal{U}))^{-\mathcal{C}} \\
=& e(\tilde{h}, u)^{r_1 \cdot \mathcal{C} + t_1} \cdot e(g, \tilde{h})^{-(r_4 \cdot \mathcal{C} + t_4)} \cdot (e(\mathcal{G}, u)/e(g, \mathcal{U}))^{-\mathcal{C}} \\
=& e(\tilde{h}, u)^{r_1 \cdot \mathcal{C} + t_1} \cdot e(g, \tilde{h})^{-(r_4 \cdot \mathcal{C} + t_4)} \cdot (e(\underline{g_i \tilde{h}^{r_1}}, u)/e(g, \underline{u_i \tilde{h}^{r_4}}))^{-\mathcal{C}} \\
=& e(\tilde{h}, u)^{r_1 \cdot \mathcal{C} + t_1} \cdot e(g, \tilde{h})^{-(r_4 \cdot \mathcal{C} + t_4)} \cdot \underline{e(g_i, u)^{-\mathcal{C}} e(\tilde{h}, u)^{-r_1 \cdot \mathcal{C}} e(g, u_i)^{\mathcal{C}} e(g, \tilde{h})^{r_4 \cdot \mathcal{C}}} \\
=& \underline{e(\tilde{h}, u)^{t_1} \cdot e(g, \tilde{h})^{-t_4}} \cdot e(g_i, u)^{-\mathcal{C}} e(g, u_i)^{\mathcal{C}} \\
\overset{*}{=}& e(\tilde{h}, u)^{t_1} \cdot e(g, \tilde{h})^{-t_4} \cdot \underline{e(g, u)^{-\gamma^i \mathcal{C}} e(g, u)^{\gamma^i \mathcal{C}}} \\
=& e(\tilde{h}, u)^{t_1} \cdot e(g, \tilde{h})^{-t_4} \\
=& T_1
\end{aligned}
$$

using that $u_i = u^{\gamma^i}$ and $g_i = g^{\gamma^i}$ for $*$

$$
\begin{aligned}
T_2' =& e(g^{\delta} \cdot \mathcal{G}, \tilde{h})^{s_3} \cdot e(\tilde{h}, \tilde{h})^{-s_6} \cdot e(\tilde{h}, S)^{s_1} \cdot (e(g^{\delta} \cdot \mathcal{G}, S)/e(g, h))^{-\mathcal{C}} \\
=& e(g^{\delta} \cdot \mathcal{G}, \tilde{h})^{r_3 \cdot \mathcal{C} + t_3} \cdot e(\tilde{h}, \tilde{h})^{-(r_1 \cdot r_3 \cdot \mathcal{C} + t_6)} \cdot e(\tilde{h}, S)^{r_1 \cdot \mathcal{C} + t_1} \cdot (e(g^{\delta} \cdot \mathcal{G}, S)/e(g, h))^{-\mathcal{C}} \\
=& e(g^{\delta} \cdot \underline{g_i \tilde{h}^{r_1}}, \tilde{h})^{r_3 \cdot \mathcal{C} + t_3} \cdot e(\tilde{h}, \tilde{h})^{-(r_1 \cdot r_3 \cdot \mathcal{C} + t_6)} \cdot e(\tilde{h}, \underline{\sigma_i \tilde{h}^{r_3}})^{r_1 \cdot \mathcal{C} + t_1} \\
& \cdot (e(g^{\delta} \cdot \underline{g_i \tilde{h}^{r_1}}, \underline{\sigma_i \tilde{h}^{r_3}})/e(g, h))^{-\mathcal{C}} \\
=& e(g^{\delta} \cdot g_i \tilde{h}^{r_1}, \tilde{h})^{r_3 \cdot \mathcal{C} + t_3} \cdot e(\tilde{h}, \tilde{h})^{-(r_1 \cdot r_3 \cdot \mathcal{C} + t_6)} \cdot \underline{e(\tilde{h}, \sigma_i)^{r_1 \cdot \mathcal{C} + t_1} e(\tilde{h}, \tilde{h})^{r_1 r_3 \cdot \mathcal{C} + t_1 r_3}} \\
& \cdot \underline{e(g^{\delta} \cdot g_i \tilde{h}^{r_1}, \tilde{h})^{-r_3 \mathcal{C}} e(g^{\delta} \cdot g_i, \sigma_i)^{-\mathcal{C}} e(\tilde{h}, \sigma_i)^{-r_1 \mathcal{C}} e(g, h)^{\mathcal{C}}} \\
=& \underline{e(g^{\delta} \cdot g_i \tilde{h}^{r_1}, \tilde{h})^{t_3}} \cdot e(\tilde{h}, \tilde{h})^{-t_6} \cdot \underline{e(\tilde{h}, \sigma_i)^{t_1}} \cdot e(\tilde{h}, \tilde{h}^{r_3})^{t_1} \cdot e(g^{\delta} \cdot g_i, \sigma_i)^{-\mathcal{C}} \cdot e(g, h)^{\mathcal{C}} \\
=& e(g^{\delta} \cdot g_i \tilde{h}^{r_1}, \tilde{h})^{t_3} \cdot e(\tilde{h}, \tilde{h})^{-t_6} \cdot \underline{e(\tilde{h}, \sigma_i \tilde{h}^{r_3})^{t_1}} \cdot e(g^{\delta} \cdot g_i, \sigma_i)^{-\mathcal{C}} \cdot e(g, h)^{\mathcal{C}} \\
\overset{*}{=}& e(g^{\delta} \cdot g_i \tilde{h}^{r_1}, \tilde{h})^{t_3} \cdot e(\tilde{h}, \tilde{h})^{-t_6} \cdot e(\tilde{h}, \sigma_i \tilde{h}^{r_3})^{t_1} \cdot \underline{1} \\
=& e(g^{\delta} \cdot \underline{\mathcal{G}}, \tilde{h})^{t_3} \cdot e(\tilde{h}, \tilde{h})^{-t_6} \cdot e(\tilde{h}, \underline{S})^{t_1} \\
=& T_2
\end{aligned}
$$

using that σ_i is a valid signature on γ^i, i.e. $e(g^{\delta} \cdot g_i, \sigma_i) = e(g^{\delta} \cdot g^{\gamma^i}, h^{1/(\delta+\gamma^i)}) = e(g, h)$ for $*$.

$$
\begin{aligned}
T_3' =& g^{s_1} \cdot \tilde{h}^{s_5} \cdot B^{-\mathcal{C}} \\
=& g^{r_1 \cdot \mathcal{C} + t_1} \cdot \tilde{h}^{open \cdot \mathcal{C} + t_5} \cdot B^{-\mathcal{C}} \\
=& g^{r_1 \cdot \mathcal{C} + t_1} \cdot \tilde{h}^{open \cdot \mathcal{C} + t_5} \cdot (\underline{g^{r_1} \tilde{h}^{open}})^{-\mathcal{C}} \\
=& g^{t_1} \cdot \tilde{h}^{t_5} \\
=& T_3
\end{aligned}
$$

$$\begin{aligned}
T_4' =& g^{s_6} \cdot \tilde{h}^{s_7} \cdot B^{-s_3} \\
\overset{*}{=}& g^{r_1 \cdot r_3 \cdot \mathcal{C} + t_6} \cdot \tilde{h}^{open \cdot r_3 \cdot \mathcal{C} + t_7} \cdot B^{-(r_3 \cdot \mathcal{C} + t_3)} \\
=& g^{r_1 \cdot r_3 \cdot \mathcal{C} + t_6} \cdot \tilde{h}^{open \cdot r_3 \cdot \mathcal{C} + t_7} \cdot \underline{(g^{r_1} \tilde{h}^{open})}^{-(r_3 \cdot \mathcal{C})} \cdot B^{-t_3} \\
=& g^{t_6} \cdot \tilde{h}^{t_7} \cdot B^{-t_3} \\
=& T_4
\end{aligned}$$

using $mult = r_1 r_3$ for $*$.

$$\begin{aligned}
T_5' =& e(\tilde{h}, acc_V)^{s_1} / e(g, \tilde{h})^{s_2} \cdot (e(\mathcal{G}, acc_V) / e(g, \mathcal{W}) / z)^{-\mathcal{C}} \\
=& e(\tilde{h}, acc_V)^{r_1 \cdot \mathcal{C} + t_1} / e(g, \tilde{h})^{r_2 \cdot \mathcal{C} + t_2} \cdot (e(\mathcal{G}, acc_V) / e(g, \mathcal{W}) / z)^{-\mathcal{C}} \\
=& e(\tilde{h}, acc_V)^{r_1 \cdot \mathcal{C} + t_1} / e(g, \tilde{h})^{r_2 \cdot \mathcal{C} + t_2} \cdot (e(\underline{g_i \tilde{h}^{r_1}}, acc_V) / e(g, \underline{wit_{V,i} \cdot \tilde{h}^{r_2}}) / z)^{-\mathcal{C}} \\
=& e(\tilde{h}, acc_V)^{r_1 \cdot \mathcal{C} + t_1} / e(g, \tilde{h})^{r_2 \cdot \mathcal{C} + t_2} \cdot \underline{e(g_i, acc_V)^{-C} \cdot e(\tilde{h}, acc_V)^{-r_1 \mathcal{C}} \cdot e(g, wit_{V,i})^{\mathcal{C}} \cdot e(g, \tilde{h})^{r_2 \mathcal{C}} \cdot z^C} \\
=& \underline{e(\tilde{h}, acc_V)^{t_1} / e(g, \tilde{h})^{t_2}} \cdot e(g_i, acc_V)^{-C} \cdot e(g, wit_{V,i})^{\mathcal{C}} \cdot z^C \\
\overset{*}{=}& e(\tilde{h}, acc_V)^{t_1} / e(g, \tilde{h})^{t_2} \cdot \underline{1} \\
=& e(\tilde{h}, acc_V)^{t_1} / e(g, \tilde{h})^{t_2} \\
=& T_5
\end{aligned}$$

Using for $*$ that acc_V and $wit_{V,i}$ fulfill the accumulator verification equation $e(g_i, acc_V) / e(g, wit_{V,i}) = z$, which implies that $e(g_i, acc_V)^{-1} \cdot e(g, wit_{V,i}) \cdot z = 1$.

Hence, we proved that $(T_1, \ldots, T_5) = (T_1', \ldots, T_5')$, i.e. that Verify recreates the same T_i values that Sign used. This implies correctness of the scheme as noted in the beginning of the proof. □

4.2.3 Anonymity

We now prove anonymity of our scheme in the random oracle model. Essentially, anonymity follows from the fact that the underlying protocol (Construction 4.2) can be simulated. We can create signatures by computing a simulated protocol transcript, then fixing the corresponding hash value to the transcript's challenge. The resulting signature will then be correct and distributed as expected. Anonymity follows from the fact that this process does not depend on any user secrets and hence the challenge signature created by this process is independent of the identity chosen for signing.

Lemma 4.11. *Construction 4.5 fulfills* anonymity against early full key exposure *(Definition 2.9) in the random oracle model.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time attacker against the anonymity experiment ($\mathrm{Exp}_{\Pi,\mathcal{A}}^{\mathrm{anon}-b}(\cdot,\cdot)$ for $b \in \{0,1\}$) that makes at most $q(\lambda)$ queries to the random oracle and the Sign oracle combined. We construct an algorithm $\mathcal{B}$ that simulates the anonymity experiment independently of b for $\mathcal{A}$. Later we will argue that $\mathcal{B}$'s simulation

is indistinguishable from the real experiment(s) for $\mathcal{A}$ (except with negligible probability) and when the simulation does not fail, $\mathcal{A}$'s output is independent of b, which will give us the desired bound on $\mathcal{A}$'s advantage.

$\mathcal{B}$ on input λ behaves as follows:

1. $(pk, U, (sk[i])_{i \in U}) \leftarrow \text{KeyGen}(1^\lambda, 1^{n'})$ using randomness $\omega \xleftarrow{R} \{0,1\}^{poly(\lambda)}$.
2. $\mathcal{A}$ has oracle access to $\mathcal{H}$, which $\mathcal{B}$ will simulate by maintaining a set $Q \subseteq \{0,1\}^* \times \mathbb{Z}_p$: whenever $\mathcal{A}$ queries $\mathcal{H}(x)$ for some $x \in \{0,1\}^*$, $\mathcal{B}$ checks whether $\exists y : (x,y) \in Q$, in which case it returns y. Otherwise, it chooses $y \xleftarrow{R} \mathbb{Z}_p$, adds (x,y) to Q and returns y.
3. $\mathcal{A}$ is given pk and U and randomness ω from Step 1.
4. Eventually $\mathcal{A}$ outputs two identities $(i_0, i_1) \in U^2$, a set V^* with $\{i_0, i_1\} \subseteq V^* \subseteq U$ and a message m^*.
5. $\mathcal{B}$ chooses $\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B \xleftarrow{R} \mathbb{G}$ and $\mathcal{C} \xleftarrow{R} \mathbb{Z}_p$. It chooses random $s_1, \ldots, s_7 \xleftarrow{R} \mathbb{Z}_p$. It sets $\sigma := (\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$. If there is no y already such that $((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})), y) \in Q$, then $\mathcal{B}$ adds $((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})), \mathcal{C})$ to Q. Then it hands the challenge signature σ to $\mathcal{A}$.
6. Eventually, $\mathcal{A}$ outputs a bit b'. $\mathcal{B}$ also outputs b'.

The point where the simulation may fail to deliver the expected behavior for the view of $\mathcal{A}$ is in Step 5. If $\exists y : ((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})), y) \in Q$ at that point (and $y \neq \mathcal{C}$), the signature produced by $\mathcal{B}$ is invalid.

In both the original experiment and in $\mathcal{B}$, let $Q \subseteq \{0,1\}^* \times \mathbb{Z}_p$ be the set of fixed hash values; i.e. $(x,y) \in Q$ if $\mathcal{A}$ queried the random oracle for x, receiving y as an answer, or it received a signature σ on some m for a set of users V and $(x,y) = ((m, acc_V, \Psi_1(\sigma, acc_V)), \Psi_2(\sigma))$. Similarly, let $Q' \subseteq Q$ be the same set at the point in the execution when $\mathcal{A}$ outputs its challenge message. Let σ be the challenge signature that $\mathcal{A}$ receives for its request m^*, V^*, i_0, i_1. Let fail be the event that $\exists y : ((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})), y) \in Q'$ (i.e. the hash value for the challenge signature was already determined before signing).

In both experiments, Q, σ, and $\mathcal{A}$'s random bits uniquely determine the output of $\mathcal{A}$. It is easy to see that Q is distributed the same in both $\mathcal{B}$ and $\text{Exp}^{\text{anon}-b}_{\Pi,\mathcal{A}}$ (where Q is determined by a proper random oracle), as $\mathcal{B}$ answers fresh queries to $\mathcal{H}$ with values y chosen uniformly at random (and it answers repeat queries with the same hash value). Note that *if* $((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})), \mathcal{C})$ is added to Q in Step 5, $\mathcal{C}$ is also chosen uniformly at random as expected.

If the event fail does not occur, the distribution of the challenge signature $\sigma = (\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$ is the same as in the original experiment: in both cases, $\sigma \xleftarrow{R} \mathbb{G}^5 \times \mathbb{Z}_p^7 \times \mathbb{Z}_p$ by definition of $\mathcal{B}$, and of Sign with a random oracle, using the

assumption that fail does not occur (i.e. the hash value for the signature was not yet fixed).

Overall, this implies that $\Pr[\mathcal{B}(\lambda,n)=1 \mid \neg\text{fail}] = \Pr[\text{Exp}^{\text{anon}-b}_{\Pi,\mathcal{A}}(\lambda,n)=1 \mid \neg\text{fail}]$ for $b \in \{0,1\}$. And in particular,

$$\Pr[\text{Exp}^{\text{anon}-0}_{\Pi,\mathcal{A}}(\lambda,n)=1 \mid \neg\text{fail}] = \Pr[\text{Exp}^{\text{anon}-1}_{\Pi,\mathcal{A}}(\lambda,n)=1 \mid \neg\text{fail}] \tag{4.4}$$

We now analyze the probability of event fail. Recall that fail is the event that for the challenge signature σ, $\exists y : \ ((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})), y) \in Q'$, where Q' is the record of random oracle queries when $\mathcal{A}$ outputs its challenge message m. By definition of $\mathcal{B}$ and our signature scheme Π, respectively, it holds that $(\mathcal{G},\mathcal{U},\mathcal{W},S,B)$ as part of $\Psi_1(\sigma, acc_{V^*})$ is uniformly distributed in $\mathbb{G}^5$. Because $|Q'| \leq q(\lambda)$, the probability that $((m^*, \Psi_1(\sigma, acc_{V^*})), y) \in Q'$ is at most $q(\lambda)/p^5$. Hence,

$$\Pr[\text{fail}] \leq q(\lambda)/p^5 \tag{4.5}$$

For the sake of readability, for $b \in \{0,1\}$, let $\text{Exp}_b = 1$ denote the event that $\text{Exp}^{\text{anon}-b}_{\Pi,\mathcal{A}}(\lambda,n) = 1$. Finally, this implies that

$$\begin{aligned}
&|\Pr[\text{Exp}_0=1] - \Pr[\text{Exp}_1=1]| \\
=&|\Pr[\text{Exp}_0=1 \mid \neg\text{fail}]\cdot\Pr[\neg\text{fail}] + \Pr[\text{Exp}_0=1 \mid \text{fail}]\cdot\Pr[\text{fail}] \\
&- \Pr[\text{Exp}_1=1 \mid \neg\text{fail}]\cdot\Pr[\neg\text{fail}] - \Pr[\text{Exp}_1=1 \mid \text{fail}]\cdot\Pr[\text{fail}]| \\
\overset{(4.4)}{=}&|\Pr[\text{Exp}_0=1 \mid \text{fail}]\cdot\Pr[\text{fail}] - \Pr[\text{Exp}_1=1 \mid \text{fail}]\cdot\Pr[\text{fail}]| \\
=&|\Pr[\text{Exp}_0=1 \mid \text{fail}] - \Pr[\text{Exp}_1=1 \mid \text{fail}]|\cdot\Pr[\text{fail}] \\
\overset{(4.5)}{\leq}& \Pr[\text{fail}] \leq q(\lambda)/p^5
\end{aligned}$$

Because q is a polynomial and $p \geq 2^\lambda$, $q(\lambda)/p^5$ is negligible in λ. Consequently, $\mathcal{A}$ has negligible advantage in the anonymity experiment against Construction 4.5. □

4.2.4 Unforgeability

For unforgeability, we will split the proof into multiple parts. The rough idea is that we apply the forking lemma on a successful ring signature forger to receive two signatures with different hash challenges. From these, we can compute a witness of the underlying protocol (Construction 4.2), which will allow us to break one of several security assumptions.

More specifically, from an attacker $\mathcal{A}$ against the unforgeability game, we construct a simulator algorithm $\mathcal{B}$ that will later be the algorithm that we use the forking lemma on. $\mathcal{B}$ simulates the game for $\mathcal{A}$, replacing signature queries with simulated signatures like for anonymity (Lemma 4.11) and simulating the random oracle. From $\mathcal{B}$, we then construct an extractor algorithm $\mathcal{E}$ that uses forking lemma rewinding techniques on $\mathcal{B}$ to retrieve a witness for the underlying protocol. Finally, we set up multiple reductions

that run $\mathcal{E}$ to retrieve a witness and compute solutions for their respective games.

These reductions correspond to ways of (potentially) breaking the ring signature scheme, namely:

- Computing an accumulator witness for an identity $i \notin V$ allows signing a message for V using $sk[i]$ (by definition of Sign) even though i is not part of the ring.
- Forging an identity signature $\sigma = g^{1/(\gamma^i+\delta)}$ (alongside u^{γ^i}) for any user i is equivalent to computing their secret key $sk[i]$ and hence would allow forging a ring signature.
- Computing the discrete logarithm of $\tilde{h}$ to base g would disable the underlying protocol's guarantee that the prover knows a valid signature (cf. Section 4.1.1, $mult = r_1 \dots r_3$).
- Forging a signature $\sigma = g^{1/(m+\delta)}$ for some $m \notin \{\gamma^i \mid i \in [n]\}$ would allow creating a ring signature by computing a witness for the syntactically invalid accumulator value m, which is not covered through accumulator security.

For more details and formal definitions, we refer to the proof of Lemma 4.14.

We proceed with the three steps (constructing the simulator $\mathcal{B}$, constructing the extractor $\mathcal{E}$, then applying the reductions) as noted above.

The simulator $\mathcal{B}$

First, we construct $\mathcal{B}$. Besides replacing queried signatures with simulated signatures, $\mathcal{B}$ will also establish syntax useful for rewinding and later proofs.

Let $\mathcal{A}$ be a probabilistic polynomial-time attacker against our ring signature Π's unforgeability game $\mathrm{Exp}^{\mathrm{sigforge}}_{\Pi,\cdot}$ (Definition 2.10). Let q be a polynomial such that $q(\lambda) - 1$ is an upper bound on the combined number of signature and hash queries that $\mathcal{A}$ does for security parameter λ.

$\mathcal{B}$ on input $\lambda, (pk, U, (sk[i])_{i\in U})$, and a list of hash oracle values $H = (H_1, \dots, H_{q(\lambda)}) \in \mathbb{Z}_p^{q(\lambda)}$ behaves as follows:

1. $\mathcal{B}$ chooses $((\mathcal{G}^{(k)}, \mathcal{U}^{(k)}, \mathcal{W}^{(k)}, S^{(k)}, B^{(k)}), (s_1^{(k)}, \dots, s_7^{(k)}))_{k=1}^{q(\lambda)} \xleftarrow{R} (\mathbb{G}^5 \times \mathbb{Z}_p^7)^{q(\lambda)}$. It sets $k = 1$ initially.
2. $\mathcal{B}$ simulates $\mathcal{A}$ with random bits $\omega_{\mathcal{A}} \xleftarrow{R} \{0,1\}^{poly(\lambda)}$.
3. $\mathcal{A}$ is given pk, U, oracle access to $\mathrm{Sign}_{sk[\cdot]}(\cdot,\cdot)$, oracle access to $\mathcal{H}$ and oracle access to $i \mapsto sk[i]$.
4. $\mathcal{B}$ answers hash queries $\mathcal{H}(x)$ by keeping an initially empty set $Q \subseteq \{0,1\}^* \times \mathbb{Z}_p$ and an index j (initially, $j = 1$). If $\exists y : (x, y) \in Q$, it answers the query with y, otherwise it answers with H_j, adds (x, H_j) to Q, and increments j.

5. $\mathcal{B}$ answers signature queries for $\text{Sign}_{sk[\cdot]}(m, V)$ in the following way: $\mathcal{B}$ sets $\mathcal{C} := H_j$, then increments j, and sets $((\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (s_1, \ldots, s_7)) = ((\mathcal{G}^{(k)}, \mathcal{U}^{(k)}, \mathcal{W}^{(k)}, S^{(k)}, B^{(k)}), (s_1^{(k)}, \ldots, s_7^{(k)}))$, then increments k.
   $\mathcal{B}$ then sets $\sigma = (\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, s_1, \ldots, s_7, \mathcal{C})$. If $\exists y : ((m, acc_V, \Psi_1(\sigma, acc_V)), y) \in Q$, then $\mathcal{B}$ outputs fail and aborts. Otherwise it adds $((m, acc_V, \Psi_1(\sigma, acc_V)), \mathcal{C})$ to Q. Then $\mathcal{B}$ hands the signature σ to $\mathcal{A}$.

6. $\mathcal{B}$ answers secret key queries $sk[i]$ with the corresponding key from its input.

7. Eventually, $\mathcal{A}$ outputs a message m^*, a set $V^* \subseteq U$ and a signature σ^*. If $\neg\exists y : ((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})), y) \in Q$, then $\mathcal{B}$ issues a virtual query for the hash value $\mathcal{H}((m^*, acc_{V^*}, \Psi_1(\sigma, acc_{V^*})))$.

8. If $\text{Verify}_{pk}(m^*, V^*, \sigma^*) = 1$ and $\mathcal{A}$ did not query $sk[i]$ for any $i \in V^*$, nor did it query $\text{Sign}_{sk[\cdot]}(m^*, V^*)$, then $\mathcal{B}$ outputs (m^*, V^*, σ^*). Otherwise, $\mathcal{B}$ outputs fail.

Note that $\mathcal{B}$ takes all secret keys as input, so a priori, there is no need to create signatures in this way instead of relying on the Sign algorithm. However, one of the reductions later will take advantage of this by replacing one of the $sk[i]$ by a dummy key, and exploit that signature queries can still be correctly answered. Furthermore, note that $\mathcal{B}$ takes random hash oracle values as input (which will simplify rewinding later), and chooses all random values for signature queries in the beginning to simplify proofs. Of course, this is completely equivalent to choosing fresh hash values or signature values, respectively, on demand. Lastly, in Step 7, $\mathcal{B}$ issues a virtual hash query if the hash value for $\mathcal{A}$'s output message is not yet determined (this is necessary anyway to evaluate Verify in Step 8). Again, this simplifies proofs and covers attackers that do not query the hash value of their forgery. Our choice of q accounts for that extra query.

The following lemma bounds the probability for $\mathcal{B}$ to output a forgery (i.e. to not output fail).

Lemma 4.12. *For all $n' \in \mathbb{N}$, it holds that if* $\Pr[\text{Exp}^{\text{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n') = 1] = \epsilon(\lambda)$*, then*

$$\Pr[B(\lambda, pk, U, (sk[i])_{i\in U}, H) \neq \text{fail}] \geq \epsilon(\lambda) - q(\lambda)^2/2^{5\lambda}$$

where $U = [n']$ and the probability is over the random bits $\omega_{\mathcal{B}}$ of $\mathcal{B}$, $H \xleftarrow{R} \mathbb{Z}_p^{q(\lambda)}$, and $pk, (sk[i])_i$ distributed as induced by KeyGen.

Proof. Let fail be the event that $\mathcal{B}$ outputs fail. Let fail_5 be the event that $\mathcal{B}$ outputs fail in Step 5, and fail_8 the same for Step 8.

Our probability space is based on $pk, (sk[i])_{i\in U}, H$, and $\omega_{\mathcal{B}}$. All other random variables are derived. In particular, $\text{Exp}^{\text{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n') = 1$ is the event that the sigforge experiment outputs 1 for the run where the experiment answers queries using H and $((\mathcal{G}^{(k)}, \mathcal{U}^{(k)}, \mathcal{W}^{(k)}, S^{(k)}, B^{(k)}), (s_1^{(k)}, \ldots, s_7^{(k)}))_{k=1}^{q(\lambda)}$, and $\mathcal{A}$ uses random bits $\omega_{\mathcal{A}}$ (as derived from $\omega_{\mathcal{B}}$). Note that this follows the same distribution as in the experiment's original definition.

We first analyze $\Pr[\text{fail}_5]$, which occurs if the hash value for a simulated signature is already fixed. Consider one signature query and let Q be the set of fixed hash values at that point. Because $\mathcal{G}, \mathcal{U}, \mathcal{W}, S, B$ are chosen uniformly at random from $\mathbb{G}$, $\Pr[\exists y : ((m, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B, T_1, \ldots, T_5), y) \in Q] \leq |Q|/|\mathbb{G}|^5 \leq q(\lambda)/p^5$ using the union bound. And hence, the probability that this happens in one of the at most $q(\lambda)$ signature queries is

$$\Pr[\text{fail}_5] \leq q(\lambda)^2/p^5$$

We now consider the event fail_8. If fail_5 does not occur, then $\mathcal{B}$'s simulation of the sigforge game is consistent and in that case, $\mathcal{B}$ outputs $fail$ if and only if the experiment outputs 0, i.e. $\text{Exp}^{\text{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n') = 1 \wedge \neg\text{fail}_5 \Leftrightarrow \neg\text{fail}_8 \wedge \neg\text{fail}_5$.
It follows that

$$\begin{aligned}
\Pr[\neg\text{fail}] &= \Pr[\neg\text{fail}_8 \wedge \neg\text{fail}_5] \\
&= \Pr[\text{Exp}^{\text{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n') = 1 \wedge \neg\text{fail}_5] \\
&\geq \Pr[\text{Exp}^{\text{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n') = 1] - \Pr[\text{fail}_5] \\
&= \epsilon(\lambda) - \Pr[\text{fail}_5] \\
&\geq \epsilon(\lambda) - q(\lambda)^2/p^5 \\
&\geq \epsilon(\lambda) - q(\lambda)^2/2^{5\lambda}
\end{aligned}$$

as required. □

The extractor $\mathcal{E}$

From the algorithm $\mathcal{B}$ we will now construct $\mathcal{E}$, which will run $\mathcal{B}$ and after $\mathcal{B}$ outputs a forgery, rewind it to the involved hash query, executing it from that point on with different random hash values. In our case, rewinding happens in the form of executing $\mathcal{B}$ again with the same input, same random bits, and same hash values up until some hash index. If $\mathcal{E}$ is able to obtain two forgeries from $\mathcal{B}$, it computes a witness of the underlying protocol or, under special circumstances, two sets with the same accumulator value.

Let $\mathcal{B}, q$ be as described above. For a message m, a set $V \subseteq U$, and a valid signature σ, we say that σ was created at hash index i if i is the index of H that $\mathcal{B}$ used when adding $((m, acc_V, \Psi_1(\sigma, acc_V)), H_i)$ to Q. Note that for the output of $\mathcal{B}$, such an index always exists (if it does not already when $\mathcal{A}$ outputs its forgery, $\mathcal{B}$ issues a virtual hash query).

$\mathcal{E}$ on input λ, $(pk, U, (sk[i])_{i \in U})$ behaves as follows:

1. Choose $H = (H_1, \ldots, H_{q(\lambda)}) \xleftarrow{R} \mathbb{Z}_p^{q(\lambda)}$ and $\omega_{\mathcal{B}} \xleftarrow{R} \{0,1\}^{poly(\lambda)}$.
2. $\mathcal{E}$ runs $\mathcal{B}$ on input $(\lambda, pk, U, (sk[i])_{i \in U}, H)$ with randomness $\omega_{\mathcal{B}}$. If $\mathcal{B}$ fails, $\mathcal{E}$ outputs fail and aborts. Otherwise $\mathcal{B}$ outputs (m, V, σ).

3. Let $i \in [q(\lambda)]$ be the hash index where σ was created.
4. $\mathcal{E}$ chooses new $H'_i, \ldots, H'_{q(\lambda)} \xleftarrow{R} \mathbb{Z}_p$ and sets $H' := (H_1, \ldots, H_{i-1}, H'_i, \ldots, H'_{q(\lambda)})$.
5. $\mathcal{E}$ runs $\mathcal{B}$ again with the same randomness $\omega_{\mathcal{B}}$ and input $(\lambda, pk, U, (sk[i])_{i\in U}, \underline{H'})$. If $\mathcal{B}$ fails, $\mathcal{E}$ returns fail and aborts. Otherwise $\mathcal{B}$ outputs (m', V', σ').
6. Let $i' \in [q(\lambda)]$ be the hash index where σ' was created.
7. If $i' \neq i$ or $\Psi_2(\sigma) = \Psi_2(\sigma')$, $\mathcal{E}$ outputs fail and aborts.
8. If index i was used to answer a signature query $\mathrm{Sign}_{sk[\cdot]}(m, \hat{V})$, $\mathcal{E}$ outputs $(V, \hat{V})$.
9. Otherwise, if $V \neq V'$, $\mathcal{E}$ outputs (V, V').
10. Otherwise, $\mathcal{E}$ computes $((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, \ldots, r_4, open, mult, tmp))$ as in Lemma 4.3 with transcripts $(\Psi_1(\sigma), \Psi_2(\sigma), \Psi_3(\sigma))$ and $(\Psi_1(\sigma'), \Psi_2(\sigma'), \Psi_3(\sigma'))$. $\mathcal{E}$ outputs $(V, ((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, \ldots, r_4, open, mult, tmp)))$.

Lemma 4.13. *Let R be the NP-relation of the protocol Σ in Construction 4.2. Let $\mathcal{A}$ be a probabilistic polynomial-time algorithm with $\Pr[\mathrm{Exp}^{\mathrm{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n') = 1] = \epsilon(\lambda)$. Then $\mathcal{E}$ runs in polynomial time. $\mathcal{E}$ either succeeds or outputs* fail. *If $\mathcal{E}$ succeeds, it either outputs*

- *a set V and an element $((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, r_2, r_3, r_4, open, mult, tmp)) \in R$*
- *or sets $V', V'' \subseteq U$ with $acc_{V'} = acc_{V''}$*

The probability for $\mathcal{E}$ to succeed is

$$\Pr[\mathcal{E}(\lambda, (pk, U, (sk[i])_{i\in U})) \neq \text{fail}] \geq \mu(\lambda) \left(\frac{\mu(\lambda)}{q(\lambda)} - \frac{1}{2^\lambda} \right)$$

where $\mu(\lambda) = \epsilon(\lambda) - q(\lambda)^2/p^5$ and the probability space is defined over $(pk, U, (sk[i])_{i\in U}) \leftarrow \mathrm{KeyGen}(1^\lambda, 1^{n'})$.

Proof. Let $\mathcal{B}$ be the algorithm constructed from $\mathcal{A}$ in Lemma 4.12. Let q be a polynomial such that $q(\lambda) - 1$ is an upper bound on the combined number of signature and hash queries that $\mathcal{A}$ does for security parameter λ.

By Lemma 4.12, $\mathcal{B}$ has success probability at least $\mu(\lambda) := \epsilon(\lambda) - q(\lambda)^2/p^5$.

To match the syntax of the forking lemma (Lemma 2.7), assume that $\mathcal{B}$ outputs $(0, \epsilon)$ instead of fail and otherwise $(i, (m, V, \sigma))$ such that σ was created at hash index i. Note that the first part of $\mathcal{E}$ (Steps 1-7) behaves exactly like the forking lemma's $F_{\mathcal{B}}$. $\mathcal{E}$ outputs fail if and only if $F_{\mathcal{B}}$ outputs fail (for the same choice of hash values and random bits $\omega_{\mathcal{B}}$ in both algorithms). Applying the forking lemma gives us

$$\Pr[\mathcal{E}(\ldots) \neq \text{fail}] = \Pr[F_{\mathcal{B}}(\ldots) \neq \text{fail}] \leq \mu(\lambda) \left(\frac{\mu(\lambda)}{q(\lambda)} - \frac{1}{p} \right) \leq \mu(\lambda) \left(\frac{\mu(\lambda)}{q(\lambda)} - \frac{1}{2^\lambda} \right)$$

as required.

If $\mathcal{E}$ does not abort prematurely (i.e. fail), there are three cases to consider for $\mathcal{E}$'s output, corresponding to Step 8, Step 9, and Step 10, respectively.

Suppose that index i was used to answer a signature query $\text{Sign}_{sk[\cdot]}(m, \hat{V})$. Let $\hat{\sigma}$ be the resulting signature of that query. Then in the first run, $\mathcal{A}$ output the forgery (m, V, σ) and $\mathcal{B}$ checked that $\mathcal{A}$ has never queried $\text{Sign}_{sk[\cdot]}(m, V)$. Hence $V \neq \hat{V}$. Furthermore, since both σ and $\hat{\sigma}$ were created at hash index i, $(m, acc_V, \Psi_1(\sigma, acc_V)) = (m, acc_{\hat{V}}, \Psi_1(\hat{\sigma}, acc_{\hat{V}}))$ and in particular, $acc_V = acc_{\hat{V}}$. Overall, in this case, $\mathcal{E}$ outputs $(V, \hat{V})$ with $V \neq \hat{V}$ and $acc_V = acc_{\hat{V}}$, as required.

If index i was not used to answer any signature query, then it was used to answer a hash query. Since both σ and σ' were created at hash index i, $(m, acc_V, \Psi_1(\sigma, acc_V)) = (m', acc_{V'}, \Psi_1(\sigma', acc_{V'}))$.

For this, there are two cases. If $\mathcal{E}$ outputs (V, V') in Step 9 because $V \neq V'$, it also holds that $acc_V = acc_{V'}$ as required.

For $V = V'$, consider the following argument: $(\Psi_1(\sigma), \Psi_2(\sigma), \Psi_3(\sigma))$ and $(\Psi_1(\sigma'), \Psi_2(\sigma'), \Psi_3(\sigma'))$ are two accepting transcripts of Construction 4.2 because σ, σ' are valid signatures (cf. Observation 4.9). It holds that $\Psi_1(\sigma) = \Psi_1(\sigma')$ (since σ, σ' were created at the same hash index i), and that $\Psi_2(\sigma) \neq \Psi_2(\sigma')$ (otherwise $\mathcal{E}$ would have aborted). Consequently, Lemma 4.3 can be applied and $\mathcal{E}$ correctly computes $((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, \ldots, r_4, open, mult, tmp)) \in R$ as required. □

$\mathcal{E}$ outputting $V' \neq V''$ with $acc_{V'} = acc_{V''}$ corresponds to the attacker $\mathcal{A}$ inside $\mathcal{B}$ finding an accumulator collision. Such attackers exploit the fact that a signature for V' is also valid for V''. Indeed, this weakness exists on purpose: we will later use the ring signature scheme in a context where the signer only knows acc_V but not the set V when creating a signature (cf. Observation 4.7). Attackers should not be able to find such sets V', V'' efficiently since it would immediately result in an accumulator forgery.

For the other type of output, the interpretation is less simple. There are several ways the attacker may have violated our security assumptions. We refer to the following proof for details.

Reductions

Finally, we are ready to use $\mathcal{E}$ to prove unforgeability of our ring signature scheme.

Lemma 4.14. *Construction 4.5 has* unforgeable signatures *under the n-DHE assumption[5] (Definition 2.15) and the n-HSDHE assumption (Definition 2.17) in the random oracle model.*

Proof. Let $\mathcal{A}$ be an arbitrary probabilistic polynomial-time algorithm, $n' \in \mathbb{N}$, and $n = n'(n'+1)/2$. Let $\mathcal{E}$ be the algorithm constructed as above. Lemma 4.13 implies

[5] Note that $n = n'(n'+1)/2$ where n' is the number of users of the ring signature scheme.

that if $\mathcal{A}$ has non-negligible probability to win the sigforge experiment, then $\mathcal{E}$ has non-negligible chance to output (V, x) with $x \in R$ as in Lemma 4.4 or two different sets V', V'' with the same accumulator. We will show that our assumptions imply that $\mathcal{E}$ must have negligible probability to output such values. This contradicts that $\mathcal{A}$ has non-negligible probability against unforgeability of our scheme.

For this, we will employ five reductions, each corresponding to a type of forgery output by $\mathcal{E}$.

- Type 1: $\mathcal{E}$ outputs (V', V'') with $V' \neq V''$ and $acc_{V'} = acc_{V''}$. Such a forger can break the accumulator scheme.

For the other types, let $((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, r_2, r_3, r_4, open, mult, tmp)) \in R$ be a valid element of the relation, output by $\mathcal{E}$ together with a set $V \subseteq U$. For the derandomized value of the blinded identity $\mathcal{G}$, we write $g^* := \mathcal{G} \cdot \tilde{h}^{-r_1}$. We define the remaining forgery types as follows (with informal description).

- Type 2: $mult \neq r_1 \cdot r_3$ (and not Type 1). Such a forger can compute the discrete logarithm of $\tilde{h}$ to base g.

- Type 3: $g^* \notin \{g_i \mid i \in [n]\}$ (and not Type 1 or 2), i.e. the identity that was used for signing is not one of the set up accumulator identities. Such a forger can compute an n-HSDHE tuple.

- Type 4: $g^* = g_i$ for some $i \in [n]$ but $i \notin V$ (and not Type 1 or 2), i.e. the identity is not accumulated in acc_V. Such a forger can break the accumulator scheme, computing a witness for $i \notin V$.

- Type 5: $g^* = g_i$ for some $i \in [n]$ and $i \in V$ (and not Type 1 or 2), i.e. the identity is present in V, so $\mathcal{A}$ inside $\mathcal{E}$ did not query for $sk[i]$ and was still able to produce a signature. Such a forger can break weak unforgeability of Construction 2.19.

For $i \in \{1, \ldots, 5\}$, let $\mathcal{E}_i$ be the algorithm that behaves like $\mathcal{E}$ but only outputs Type-i forgeries (it discards all others, outputting "fail" instead). Since every valid witness is exactly one of these types, it suffices to show that each $\mathcal{E}_i$ has negligible success probability. This holds because $\Pr[\mathcal{E}(\ldots) \neq \text{fail}] = \sum_{i=1}^{5} \Pr[\mathcal{E}(\ldots) \neq \text{fail} \wedge \mathcal{E} \text{ outputs a Type } i \text{ forgery}] = \sum_{i=1}^{5} \Pr[\mathcal{E}_i(\ldots) \neq \text{fail}]$ and hence if $\mathcal{E}$ has non-negligible success probability, then at least one of the $\mathcal{E}_i$ must have non-negligible success probability.

In the remainder of the proof, we will examine each $\mathcal{E}_i$ and upper-bound their success probability.

Type 1 forger

Consider the Type 1 forger $\mathcal{E}_1$. In case of success it outputs (V', V'') with $V' \neq V''$ and $acc_{V'} = acc_{V''}$. We construct an algorithm $\mathcal{A}'$ against the accumulator forging game (Definition 3.2). $\mathcal{A}'$ receives input $U = [n]$ and $(g^{\gamma^i})_{i=0, i\neq n+1}^{2n}, z$ (as well as a description of

$(\mathbb{G}, \mathbb{G}_T, e, p))$ and is expected to output (V, acc_V, i, wit) such that $e(g_i, acc_V)/e(g, wit) = z$ and $i \notin V$.

Setup

$\mathcal{A}'$ is given the public key of the accumulator $((\mathbb{G}, \mathbb{G}_T, e, p), (g^{\gamma^i})_{i=0; i\neq n+1}^{2n}, z)$ and needs to complete the ring signature scheme setup. The main challenge here is generating valid signatures $\sigma_i = h^{1/(\delta+\gamma^i)}$ for $i \in [n']$ since $\mathcal{A}'$ cannot efficiently compute any γ^i.[6] It also needs to generate u^{γ^i} such that u is chosen uniformly from $\mathbb{G} \setminus \{1\}$.

$\mathcal{A}'$ sets up the ring signature scheme as follows: It chooses a random signature scheme secret key $\delta \xleftarrow{R} \mathbb{Z}_p$. For determining the signature base h, consider the polynomial $f(x) = \prod_{i=1}^{n'}(x^i + \delta)$ (a similar trick was used in [BB04] but here, the signature scheme's secret δ is known but we are not given the n' messages γ^i). Because $\deg(f) = n'(n'+1)/2 = n$ and we are given g^{γ^i} for $i \in [n]$, we can compute $g^{f(\gamma)}$ by writing f in its canonical form $f(x) = \sum_{i=0}^{n} \alpha_i x^i$ and computing $\prod_{i=0}^{n}(g^{\gamma^i})^{\alpha_i} = g^{f(\gamma)}$. $\mathcal{A}'$ then sets $h := (g^{f(\gamma)})^{\mathcal{R}'}$ for some $\mathcal{R}' \xleftarrow{R} \mathbb{Z}_p^*$. With this setup, $\mathcal{A}'$ can compute signatures $\sigma_i = h^{1/(\delta+\gamma^i)}$ for $i \in [n']$ through polynomials $f_i(x) := f(x)/(x^i + \delta)$: $\sigma_i = (g^{f_i(\gamma)})^{\mathcal{R}'}$. Here, $g^{f_i(\gamma)}$ can be computed with the same technique as $g^{f(\gamma)}$ above. If $h = 1$, $\mathcal{A}$ outputs fail and aborts. Otherwise $f(\gamma) \neq 0$ and so $\sigma_i = (g^{f_i(\gamma)})^{\mathcal{R}'} = (g^{f(\gamma)/(\gamma^i+\delta)})^{\mathcal{R}'} = h^{1/(\gamma^i+\delta)}$ as required. To complete the secret key values, $\mathcal{A}'$ chooses $\mathcal{R} \xleftarrow{R} \mathbb{Z}_p^*$ and computes $u = g^{\mathcal{R}}$ and $u_i := g_i^{\mathcal{R}}$ (so $g_i^{\mathcal{R}} = (g^{\mathcal{R}})^{\gamma^i} = u^{\gamma^i}$ as required) for $i \in [n']$.

Finally, it chooses $\tilde{h} \xleftarrow{R} \mathbb{G}\setminus\{1\}$ and sets $pk = ((\mathbb{G}, \mathbb{G}_T, e, p), (g^{\gamma^i})_{i=0; i\neq n+1}^{2n}, z, u, g^{\delta}, h, \tilde{h})$ and $sk[i] = (i, g_i, u_i, \sigma_i, pk)$ for $i \in [n']$.

Computing a solution

$\mathcal{A}'$ runs $\mathcal{E}_1(\lambda, (pk, U, (sk[i])_{i\in U}))$. If $\mathcal{E}_1$ is successful, it outputs (V', V'') with $V' \neq V''$ and $acc_{V'} = acc_{V''}$. Without loss of generality, we assume that $V' \setminus V'' \neq \emptyset$ (if not the case, this can be achieved by renaming V, V''). $\mathcal{A}'$ finds $i \in V' \setminus V''$ and outputs $(V'', acc_{V'}, i, wit_{V',i})$, which is a valid accumulator forgery since $i \notin V''$ but $\text{Verify}(acc_{V''}, i, wit_{V',i}) = \text{Verify}(acc_{V'}, i, wit_{V',i}) = 1$ using correctness of Construction 3.5.

If $\mathcal{E}_1$ is unsuccessful, $\mathcal{A}'$ outputs fail and aborts.

Probability analysis

If $h = 1$, i.e. $f(\gamma) = 0$ (in the setup), $\mathcal{A}'$ aborts (as $h = 1$ does not happen in the actual scheme). Let bad be the event that $f(\gamma) = 0$. $f(\gamma) = 0 \Leftrightarrow \delta \in \{-\gamma^i \mid i \in [n']\}$ and because δ is chosen independently of γ, $\Pr[\text{bad}] \leq n'/p \leq n'/2^{\lambda}$.

If bad does not occur, then $pk, (sk[i])_i$ are distributed as expected. Whenever $\mathcal{E}_1$ is successful, $\mathcal{A}'$ outputs an accumulator forgery.

[6] Note that this complication is not surprising. It arises through the need to have a clear relation between the signatures σ_i and their messages (γ^i) in order to enable a proof protocol (cf. Section 4.1.2). We will discuss this issue in more detail later.

It follows that

$$\begin{aligned}
&\Pr[\mathcal{E}_1(\dots) \neq \text{fail}] \\
&\leq \Pr[\mathcal{E}_1(\dots) \neq \text{fail} \wedge \neg\text{bad}] + \Pr[\text{bad}] \\
&= \Pr[\mathcal{A}'(\dots) \neq \text{fail}] + \Pr[\text{bad}] \\
&\leq \Pr[\mathcal{A}'(\dots) \neq \text{fail}] + n'/2^\lambda \\
&\leq \mu(\lambda)
\end{aligned}$$

for some negligible function μ, using that $n'/2^\lambda$ is negligible and the n-DHE assumption for $\Pr[\mathcal{A}'(\dots) \neq \text{fail}]$ being negligible (accumulator security, Theorem 3.7).

Type 2 forger

Consider the Type 2 forger $\mathcal{E}_2$. In case of success it outputs only protocol witnesses where $mult \neq r_1 \cdot r_3$. We construct an algorithm $\mathcal{A}'$ against the discrete logarithm game. $\mathcal{A}'$ receives input $g \xleftarrow{R} \mathbb{G}$ and $\tilde{h} \xleftarrow{R} \mathbb{G}$ (as well as a description of $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$) and is expected to output $l \in \mathbb{Z}_p$ such that $g^l = \tilde{h}$.

Setup

$\mathcal{A}'$ sets up the ring signature scheme by choosing secrets $\gamma, \delta \xleftarrow{R} \mathbb{Z}_p$ and setting
$pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), (g^{\gamma^i})_{i=0;i\neq n+1}^{2n}, z, u, g^\delta, h, \tilde{h})$ for $z = e(g,g)^{\gamma^{n+1}}$ and $u, h \xleftarrow{R} \mathbb{Z}_p$. Furthermore, $sk[i] = (i, g_i, u_i, \sigma_i, pk)$ can be computed in a straight-forward manner (knowing γ and δ).

Computing a solution

$\mathcal{A}'$ runs $\mathcal{E}_2(\lambda, (pk, U, (sk[i])_{i\in U}))$. If $\mathcal{E}$ is successful, it outputs a witness
$((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, r_2, r_3, r_4, open, mult, tmp)) \in R$ with $mult \neq r_1 \cdot r_3$. In this case, Lemma 4.4 states that from such a witness, we can efficiently compute the discrete logarithm $l \in \mathbb{Z}_p$ of $\tilde{h} = g^l$. $\mathcal{A}'$ outputs l. If $\mathcal{B}$ is unsuccessful, $\mathcal{A}'$ outputs fail and aborts.

Probability analysis

$pk, (sk[i])_i$ are distributed as expected. Whenever $\mathcal{E}_2$ is successful, $\mathcal{A}'$ outputs the discrete logarithm. It follows that

$$\Pr[\mathcal{E}_2(\dots) \neq \text{fail}] = \Pr[\mathcal{A}'(\dots) \neq \text{fail}] \leq \mu(\lambda)$$

for some negligible function μ, using the discrete logarithm assumption for $\mathbb{G}$ (implied by n-DHE).

Type 3 forger

Consider the Type 3 forger $\mathcal{E}_3$. In case of success it outputs only protocol witnesses where $g^* \notin \{g_i \mid i \in [n]\}$ (and $mult = r_1 r_3$).

We construct an algorithm $\mathcal{A}'$ against the n-HSDHE game (Definition 2.17). $\mathcal{A}'$ receives input $(g, g^\delta, u, (g^{1/(\delta+\gamma^i)}, g_i = g^{\gamma^i}, u_i = u^{\gamma^i})_{i=1}^n, (g_i = g^{\gamma^i})_{i=n+2}^{2n})$ (as well as a description of $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$) and is expected to output a (new) tuple $(g^{1/(\delta+c)}, g^c, u^c)$ (where $c \in \mathbb{Z}_p \setminus \{\gamma^i \mid i \in [n]\}$).

Setup

$\mathcal{A}'$ sets up the ring signature scheme as follows: It sets $z := e(g_1, g_n) = e(g, g)^{\gamma^{n+1}}$, chooses a random $\mathcal{R} \xleftarrow{R} \mathbb{Z}_p^*$ and sets $h = g^\mathcal{R}$. It also chooses $\tilde{h} \xleftarrow{R} \mathbb{G}$. $\mathcal{A}'$ then sets $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), (g^{\gamma^i})_{i=0; i\neq n+1}^{2n}, z, u, g^\delta, h, \tilde{h})$. Furthermore, $\sigma_i := (g^{1/(\delta+\gamma^i)})^\mathcal{R} = h^{1/(\delta+\gamma^i)}$ (using the $g^{1/(\delta+\gamma^i)}$ values from its input) and with that $sk[i] = (i, g_i, u_i, \sigma_i, pk)$ for $i \in [n']$.

Computing a solution

$\mathcal{A}'$ runs $\mathcal{E}_3(\lambda, (pk, U, (sk[i])_{i\in U}))$. If $\mathcal{E}_3$ is successful, it outputs a witness $((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, r_2, r_3, r_4, open, mult, tmp)) \in R$ such that the derandomized $g^* \notin \{g_i \mid i \in [n]\}$ (and $mult = r_1 \cdot r_3$). In this case, Lemma 4.4 states that from such a witness, we can efficiently compute a signature σ^* with $e(\sigma^*, g^\delta \cdot g^*) = e(h, g)$ and u^* with $\exists x \in \mathbb{Z}_p : \; g^* = g^x = u^x = u^*$. Hence $\sigma^* = h^{1/(\delta+x)}$ and consequently $((\sigma^*)^{1/\mathcal{R}} = g^{1/(\delta+x)}, g^x, u^x)$ is a valid new n-HSDHE tuple with $x \notin \{\gamma^i \mid i \in [n]\}$. If $\mathcal{E}_3$ is unsuccessful, $\mathcal{A}'$ outputs fail and aborts.

Probability analysis

$pk, (sk[i])_i$ are distributed as expected. Whenever $\mathcal{E}_3$ is successful, $\mathcal{A}'$ outputs an n-HSDHE tuple. It follows that

$$\Pr[\mathcal{E}_3(\dots) \neq \text{fail}] = \Pr[\mathcal{A}'(\dots) \neq \text{fail}] \leq \mu(\lambda)$$

for some negligible μ, using the n-HSDHE assumption.

Type 4 forger

Consider the Type 4 forger $\mathcal{E}_4$. In case of success it outputs only a set $V \subseteq U$ and a witness where $g^* = g_i$ for some $i \in [n]$ but $i \notin V$ (and $mult = r_1 r_3$).

We construct an algorithm $\mathcal{A}'$ against the accumulator forging game (Definition 3.2). $\mathcal{A}'$ receives input $U = [n]$ and $(g^{\gamma^i})_{i=0,i\neq n+1}^{2n}, z$ (as well as a description of $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$) and is expected to output (V, acc_V, i, wit) such that $e(g_i, acc_V)/e(g, wit) = z$.

Setup

As for the Type 1 forger.

Computing a solution

$\mathcal{A}'$ runs $\mathcal{E}_4(\lambda, (pk, U, (sk[i])_{i\in U}))$. If $\mathcal{E}_4$ is successful, it outputs a witness $((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, r_2, r_3, r_4, open, mult, tmp)) \in R$ and a set $V \subseteq U$ such that the derandomized $g^* = g_i$ for some $i \in [n]$ but $i \notin V$ (and $mult = r_1 r_3$). In

this case, Lemma 4.4 states that from such a witness, we can efficiently compute an accumulator witness *wit* with $e(g^*, acc_V)/e(g, wit) = z$, i.e. $e(g_i, acc_V)/e(g, wit) = z$. If $\mathcal{E}_4$ is successful, $\mathcal{A}'$ can output (V, acc_V, i, wit) as a valid accumulator forgery. If $\mathcal{E}_4$ is unsuccessful, $\mathcal{A}'$ outputs fail and aborts.

Probability analysis

Same as for the Type 1 forger: if bad (cf. Type 1 forger) does not occur, then $pk, (sk[i])_i$ are distributed as expected. Whenever $\mathcal{E}_4$ is successful, $\mathcal{A}'$ outputs an accumulator forgery.

It follows that

$$\begin{aligned}
&\Pr[\mathcal{E}_4(\dots) \neq \text{fail}] \\
&\leq \Pr[\mathcal{E}_4(\dots) \neq \text{fail} \wedge \neg\text{bad}] + \Pr[\text{bad}] \\
&\leq \Pr[\mathcal{A}'(\dots) \neq \text{fail}] + n'/2^\lambda \\
&\leq \mu(\lambda)
\end{aligned}$$

for some negligible function μ, using that $n'/2^\lambda$ is negligible and the n-DHE assumption for accumulator security (Theorem 3.7).

Type 5 forger

Consider the Type 5 forger $\mathcal{E}_5$. In case of success it outputs only a set V and a protocol witness where $g^* = g_i$ for some $i \in [n]$ and $i \in V$ (and $mult = r_1 r_3$).

We construct an algorithm $\mathcal{A}'$ against the weakly secure signature scheme (Construction 2.19, Definition 2.20). $\mathcal{A}'$ receives a description of $(\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p)$ as input and is expected to specify a list of signature queries $m_1, \dots, m_{n'}$ before receiving the public key (h, h^δ) and signatures $\sigma_i = h^{1/(\delta+m_i)}$. Then $\mathcal{A}'$ must output a message $m^* \notin \{m_1, \dots, m_{n'}\}$ and a signature $\sigma = h^{1/(\delta+m^*)}$.

Setup

$\mathcal{A}'$ first chooses $\gamma \xleftarrow{R} \mathbb{Z}_p^*$ and a random index $i^* \xleftarrow{R} U = [n']$, and hands $(\gamma^i)_{i \in U \setminus \{i^*\}}$ to the experiment to receive signatures $(\sigma_i)_{i \in U \setminus \{i^*\}}$ and the signature scheme's public key (h, h^δ).

It then sets up the ring signature scheme as follows: $\mathcal{A}'$ sets $g := h^\mathcal{R}$ for $\mathcal{R} \xleftarrow{R} \mathbb{Z}_p^*$ (hence for the public key, g^δ can be computed as $(h^\delta)^\mathcal{R}$). It then chooses $u, \tilde{h} \xleftarrow{R} \mathbb{G}$, computes g_i, u_i using γ and sets $z = e(g, g)^{\gamma^{n+1}}$. Finally, $\mathcal{A}'$ sets $pk = ((\mathbb{G}, \mathbb{G}_\mathrm{T}, e, p), (g^{\gamma^i})_{i=0; i \neq n+1}^{2n}, z, u, g^\delta, h, \tilde{h})$ and $sk[i] = (i, g_i, u_i, \sigma_i, pk)$ for $i \in [n'] \setminus \{i^*\}$.

Computing a solution

$\mathcal{A}'$ runs $\mathcal{E}_5(\lambda, (pk, U, (sk[i]')_{i \in U}))$ with $sk[i]' = sk[i]$ for $i \neq i^*$ and $sk[i^*]' := \epsilon$ in lieu of a valid secret key (which contains the signature that $\mathcal{A}'$ does not have and wants to forge). If $\mathcal{E}_5$ is successful, it outputs $((pk, acc_V, \mathcal{G}, \mathcal{U}, \mathcal{W}, S, B), (r_1, r_2, r_3, r_4, open, mult, tmp)) \in R$ and a set $V \subseteq U$ such that the derandomized $g^* = g_i$ for some $i \in [n]$ and $i \in V$

(and $mult = r_1 r_3$). In this case, Lemma 4.4 states that from such a witness, we can efficiently compute an a signature σ^* with $e(\sigma^*, g^\delta \cdot g_i) = e(h, g)$. If $i = i^*$, $\mathcal{A}'$ outputs the signature forgery σ^* on $m^* = \gamma^i$, which it has not queried and which is valid because $e(\sigma^*, g^\delta \cdot g^{\gamma^i}) = e(h, g) \Rightarrow e(\sigma^*, h^\delta \cdot h^{\gamma^i})^{\mathcal{R}} = e(h, h)^{\mathcal{R}}$ and hence $e(\sigma^*, h^\delta \cdot h^{\gamma^i}) = e(h, h)$ as required. If $i \neq i^*$, $\mathcal{A}'$ outputs fail.

Probability analysis

Consider the probability space defined over $(pk, U, (sk[i])_{i \in U}) \leftarrow \mathrm{KeyGen}(1^\lambda, 1^{n'})$, $i^* \xleftarrow{R} [n']$, and the random bits ω for $\mathcal{E}_5$. All variables are chosen with their respective distributions by $\mathcal{A}'$ or the weak unforgeability game with the special case of $sk[i^*] = (i^*, g_{i^*}, u_{i^*}, \sigma_{i^*}, pk)$, which $\mathcal{A}'$ does not compute explicitly but that is uniquely determined through the choice of i^*, g, g_1, u, g^δ in pk. So as a random variable, $sk[i^*]$ is well-defined, following the distribution induced by KeyGen.

For fixed $(pk, U, (sk[i])_{i \in U})$, and random bits ω, the output of $\mathcal{E}_5(pk, U, (sk[i])_{i \in U})$ is uniquely determined. Assume that $\mathcal{E}_5$ does not output fail. Then it outputs a witness with $g^* = g_k$ for some $k \in [n]$ and $k \in V$. By definition of $\mathcal{E}$ (Lemma 4.13) and $\mathcal{B}$ (Lemma 4.12), the algorithm $\mathcal{A}$ does not query for $sk[k]$ in either run that $\mathcal{E}$ issues: if it had queried for $sk[k]$, it would not have won its (simulated) ring signature forging game since $k \in V$ – then $\mathcal{B}$ would have exited with output fail. Since the $sk[\cdot]$ array is only ever used by $\mathcal{B}$ (and $\mathcal{E}$) to answer $\mathcal{A}$'s key queries, if we run $\mathcal{E}_5(pk, U, (sk[i]')_{i \in U})$ with $sk[i]' = sk[i]$ for $i \neq k$ and $sk[k]' = \epsilon$ and the same random bits ω, the result will be exactly the same (the value of $sk[k]$ is never used so it can be left out). Let good be the event that $i^* = k$ (where we set $k = 1$ if $\mathcal{E}_5$ outputs fail). As i^* is chosen independently of all other variables, $\Pr[\mathrm{good}] = 1/n'$.

Whenever $\mathcal{E}_5$ with full keys is successful and good occurs, $\mathcal{E}_5$ without $sk[k]$ behaves exactly the same and then $\mathcal{A}'$ is able to output a signature forgery. Formally, it holds that $\mathcal{E}_5(pk, U, (sk[i])_{i \in U}) \neq \mathrm{fail} \wedge \mathrm{good} \Leftrightarrow \mathcal{E}_5(pk, U, \underline{(sk[i]')_{i \in U}}) \neq \mathrm{fail} \wedge \mathrm{good} \Leftrightarrow \mathcal{A}'(\dots) \neq \mathrm{fail}$. This implies

$$\begin{aligned}
&\Pr[\mathcal{A}'(\dots) \neq \mathrm{fail}] \\
&= \Pr[\mathcal{E}_5(pk, U, (sk[i])_{i \in U}) \neq \mathrm{fail} \wedge \mathrm{good}] \\
&= \Pr[\mathcal{E}_5(pk, U, (sk[i])_{i \in U}) \neq \mathrm{fail}] \cdot \Pr[\mathrm{good}] \\
&= \Pr[\mathcal{E}_5(pk, U, (sk[i])_{i \in U}) \neq \mathrm{fail}]/n'
\end{aligned}$$

By the $(n'+1)$-SDH assumption, which is implied by n-HSDHE, the attacked signature scheme (Construction 2.19) is weakly secure. It follows that $\Pr[\mathcal{A}'(\dots) \neq \mathrm{fail}] \leq \mu(\lambda)$ for some negligible μ and so $\Pr[\mathcal{E}_5(\dots) \neq \mathrm{fail}] \leq \mu(\lambda)$.

Conclusion

Overall, we have shown that $\mathcal{E}_1, \dots, \mathcal{E}_5$ only have negligible probability for success. Using our arguments from the beginning of the proof, this implies that $\mathcal{A}$ must have negligible probability of success against the unforgeability experiment. □

Recall that we set up the accumulator for $n = n'(n'+1)/2$ for just n' users of the ring signature scheme. The need to do this arises exactly from type 1 and type 4 forgers, where in the reduction, one needs to generate signatures $h^{1/(\gamma^i+\delta)}$ to complete the setup of the ring signature scheme. For this, the n'-DHE assumption (underlying the accumulator) does not supply sufficiently many g^{γ^i} values but n-DHE does.

This is a consequence of our choice for the signature scheme signing the accumulator identities (cf. Section 4.1.1). Since we sign exactly the γ^i values from the accumulator, which we do not know in the reduction to accumulator security, we need to resort to a special technique requiring $n' \cdot (n'+1)/2$ values of g^{γ^i} to create our n' signatures (cf. type 1 forgers above). Furthermore, the n-HSDHE assumption (cf. type 3 forgers) was created in [CKS09] specifically to make the chosen signature scheme work: the attacker against n-HSDHE is given the accumulator values and exactly the signatures on γ^i as required. Note that while syntactically similar, the relationship between n-DHE and n-HSDHE is unknown [CKS09]. In particular, we could not have simply reduced type 1 forgers or type 4 forgers to n-HSDHE, where the signatures are conveniently supplied to the reduction algorithm (since the value of $g^{\gamma^{n+1}}$ from breaking n-DHE does not help create a tuple $(g^{1/(\delta+c)}, g^c, u^c)$ for n-HSDHE).

Overall, both the specially-tailored n-HSDHE assumption and setting up the accumulator for $n = n' \cdot (n'+1)/2$ values could have potentially been avoided by choosing a different signature scheme for accumulator identities. Both of these are basically workarounds for the fact that the users proving knowledge of a valid signature do not have access to the signed message γ^i. The easiest solution here would be a signature scheme whose message space is $\mathbb{G}$ (as discussed in Section 4.1.1). Then the type 1 and 4 reductions could have set up the signature scheme independently of the accumulator scheme and just sign the g_i values (which are part of the accumulator's public key) using the standard signing mechanism of that signature scheme. Type 3 forgeries could have been reduced to unforgeability of the signature scheme, since with message space $\mathbb{G}$, it is reasonable to expect a protocol witness to contain the signed message g_i (and the signature σ_i). However, we were unable to find such a signature scheme. While multiple schemes with message space $\mathbb{Z}_p$ allow verifying a signature on a message m using only g^m, they usually rely on a forger supplying m to break their underlying assumption. For example, this seems to be the case for Construction 2.18 (for type 3 forgers [Oka06], m seems to be needed) and Construction 2.19 (simply because m is part of SDH tuples).

This concludes the security proofs for our ring signature construction. Our ring signature scheme can be used as a building block for other contexts. For example, we will extend an arbitrary group signature scheme with this ring signature scheme to make it support revocation with the accumulator. Note that there is no overhead in this scheme associated with it being a secure ring signature scheme: it is a straight application of the Fiat-Shamir heuristic on with no additional ring-signature-specific constructs. It also has the nice property that it is possible to sign and verify messages using only an accumulator value acc_V (cf. Observation 4.7).

Performance-wise, signatures are comparatively large in size, containing 5 elements of $\mathbb{G}$ and 8 numbers in $\mathbb{Z}_p$. Of these, $\mathcal{G}, \mathcal{W}, S \in \mathbb{G}$ and $s_1, s_2, s_3 \in \mathbb{Z}_p$ are unavoidable as

they represent an accumulator identity, an accumulator witness, and the signature on the accumulator identity. $\mathcal{U}, B \in \mathbb{G}$ and $s_4, s_5, s_6, s_7 \in \mathbb{Z}_p$ are specific to the signature scheme for signing γ^i and its protocol to demonstrate possession of a signature: $\mathcal{U}, s_4$ are used for the n-HSDHE workaround of users not knowing γ^i, and $\mathcal{B}, s_5, s_6, s_7$ are used to handle the product $r_1 \cdot r_3$ in the exponent needed when verifying the signature using blinded values $\mathcal{G}, S$.

In the remainder of this thesis, we will use the constructed ring signature scheme as a building block, noting that there might be a more efficient version of this scheme. Such an improvement can likely be achieved by replacing the signature scheme for accumulator identities with a new construction, as discussed above.

5 Revocation for group signatures

In this chapter, we examine the application of accumulators for revocation in group signature schemes. First, we define the revocation semantics we can achieve with accumulators (Section 5.1). Afterwards, we present a generic construction of group signature schemes with accumulator revocation that augments an arbitrary group signature scheme with accumulator revocation (Section 5.2).

5.1 Defining group signatures with accumulator revocation

It is not possible to augment group signatures with accumulator revocation without changing our security expectations (Definitions 2.12, 2.13, 2.14). The following argument illustrates this:

Suppose we have an accumulator scheme where accumulators can be created or updated with public information (such as Construction 3.5). In this case, an attacker is able to compute acc_V and $acc_{V'}$ for any two sets $V, V' \subseteq U$. Assume that a signature can be checked for revocation against any (current) accumulated set. Any signature that is valid w.r.t. acc_V but not w.r.t. $acc_{V'}$ was signed by someone in $V \setminus V'$. That way, attackers would get a powerful oracle for disabling anonymity.

Hence, we *cannot* realize the usual revocation semantics with the Camenisch et al. accumulator: signatures' validity cannot change for different accumulators. A valid signature always stays valid. This can also be positively formulated: revoked users' anonymity is not overturned retroactively (i.e. their old signatures cannot be traced to them). In literature, this is often called *backward-unlinkability* [NF06]. The usual group signature revocation mechanism is not backward-unlinkable: for each revoked user, a special revocation token is published that links messages (old and new) to that user.

Consequently, signatures cannot be revoked retroactively. However, what we can revoke using accumulators are *signing rights*. In this setting, an *epoch* defines an interval of time where the set of valid users does not change. A signature is created with respect to the current epoch's accumulator and creating such a signature without being included in the accumulator should be infeasible. So once a user's signing rights are revoked, he cannot sign messages that claim to be from a recent epoch.

The revocation of *signing rights* as opposed to signature revocation has certain advantages in some applications. For example, if a contract is signed in a company (by a member of the group of executives) and the signer leaves the company, the contract should still be considered valid (but the departed employee should not be able to sign any more contracts after leaving). Using classical group signature revocation techniques in this context would mean that after leaving, all signatures become invalid and the

signer's anonymity on all signed messages is revoked (retroactively) as well. While this is acceptable for contexts where revocation only occurs in case of misuse, this is an undesirable property in many other cases. In these cases, the *revocation of signing rights* semantics as introduced here are more appropriate.

We start by defining the syntax of group signatures with accumulator revocation. Afterwards, we define security properties.

5.1.1 Syntax definition

Because the notion of revoking signing rights requires a new model, we introduce a formal definition of group signatures with accumulator revocation. This definition is based on the standard definition of group signatures given in Section 2.6, but is modified and extended for the reasons detailed above. First, we give the formal definition, then we discuss how we would expect such a scheme to be used in practice.

Definition 5.1 (Group signature scheme with accumulator revocation)**.** A group signature scheme with accumulator revocation consists of the following polynomial-time algorithms:

- $\text{KeyGen}(1^\lambda, 1^n)$ is a probabilistic algorithm that generates a set U of identities (with $|U| = n$), a public key gpk, the group manager's secret key $gmsk$, and member secret keys $(sk[i])_{i \in U}$. It outputs $(U, gpk, gmsk, (sk[i])_{i \in U})$.
- $\text{EpochCreate}_{gmsk}(V, \tau)$ is a probabilistic algorithm that, given a set $V \subseteq U$ and a timestamp $\tau \in \{0,1\}^*$, outputs epoch information $\mathfrak{E}$.
- $\text{ExtractTimestamp}(\mathfrak{E})$ is a deterministic algorithm that returns a timestamp τ.
- $\text{EpochVerify}_{gpk}(V, \mathfrak{E})$ is a deterministic algorithm that returns 0 or 1.
- $\text{WitCreate}_{gpk}(V, i)$ is a deterministic algorithm that creates a witness $wit_{V,i}$.
- $\text{Sign}_{sk[i]}(m, \mathfrak{E}, wit_{V,i})$ is a probabilistic algorithm that outputs a signature σ.
- $\text{Verify}_{gpk}(m, \mathfrak{E}, \sigma)$ is a deterministic algorithm that returns 0 or 1.
- $\text{Open}_{gmsk}(m, \mathfrak{E}, \sigma)$ is a deterministic algorithm that returns an identity $i \in U$ or the failure symbol $\perp$.

A group signature scheme with accumulator revocation is *correct* if for all $\lambda, n \in \mathbb{N}, (U, gpk, gmsk, (sk[i])_i) \in [\text{KeyGen}(1^\lambda, 1^n)], V \subseteq U, \mathfrak{E} \in [\text{EpochCreate}_{gmsk}(V, \tau)], i \in V, \tau \in \{0,1\}^*$; and all messages m,

$$\Pr[\text{Verify}_{gpk}(m, \mathfrak{E}, \text{Sign}_{sk[i]}(m, \mathfrak{E}, \text{WitCreate}_{gpk}(V, i))) = 1] = 1$$

$$\Pr[\text{Open}_{gmsk}(m, \mathfrak{E}, \text{Sign}_{sk[i]}(m, \mathfrak{E}, \text{WitCreate}_{gpk}(V, i))) = i] = 1$$

$$\text{EpochVerify}_{gpk}(V, \mathfrak{E}) = 1$$

$$\text{ExtractTimestamp}(\text{EpochCreate}_{gmsk}(V, \tau)) = \tau$$

According to this definition, the group manager implements revocation by regularly publishing epoch information $\mathfrak{E}$, which one can imagine to contain an accumulator value acc_V and a timestamp τ together with a signature on (acc_V, τ) to ensure that only the group manager is able to create new epochs (using $gmsk$). Additionally, the current set V of valid users is published alongside $\mathfrak{E}$ by the group manager, which enables creation of witnesses. Through EpochVerify, it can be publicly verified that V is indeed represented by $\mathfrak{E}$.

Functionally, KeyGen sets up the system for a fixed set U of users. EpochCreate allows the group manager to create new epoch information. The group manager embeds the supplied timestamp τ in the epoch information. τ can subsequently be retrieved through ExtractTimestamp from the epoch information. EpochVerify is used to link epoch information $\mathfrak{E}$ to a set V of accumulated values. WitCreate creates a witness for inclusion in the set V of accumulated values. Lastly, next to the usual parameters, Sign, Verify, and Open additionally take epoch information $\mathfrak{E}$ as input. Sign also requires a witness, as computed by WitCreate, to create the signature.

We now describe how such a scheme could be applied in practice (this already contains some hints about security expectations but mainly serves to motivate the operations described above). When a user i wants to sign a message m for the current epoch, he follows these steps:

- Retrieve the set V of valid users and current epoch information $\mathfrak{E}$ from some (untrusted) repository.
- Check that $\text{EpochVerify}(V, \mathfrak{E}) \stackrel{!}{=} 1$, which establishes that V and $\mathfrak{E}$ are genuine. If $i \notin V$ (i.e. the user is revoked), abort.
- Create the witness $wit_{V,i}$ through WitCreate.
- Use $\text{Sign}_{sk[i]}(m, \mathfrak{E}, wit_{V,i})$ to obtain a signature σ. The signer stays anonymous among the group of potential signers V.
- Send σ and $\mathfrak{E}$ alongside m to verifiers.

Note that the first three steps only need to be done once per epoch (e.g., once per day). Furthermore, the WitCreate computation may be offloaded to some (untrusted) third-party service (and a specific scheme may offer an efficient way to check correctness of a witness). If the signer trusts the epoch information repository in Step 1 and the witness computed by a third party, there is no need for him to know the specific group V in which he is currently signing (the same way a signer does not explicitly need to know the group in normal group signatures).

When a verifier receives a message m with a signature σ and epoch information $\mathfrak{E}$, the following actions are sensible:

- Use $\text{Verify}_{gpk}(m, \mathfrak{E}, \sigma)$ to check that $\mathfrak{E}$ is valid epoch information and that σ is a valid signature for message m in epoch $\mathfrak{E}$.
- Retrieve the timestamp τ, that the group manager embedded for $\mathfrak{E}$, using the ExtractTimestamp($\mathfrak{E}$) operation and check whether this timestamp is reasonable for the message.
- (Optionally) retrieve the set V of possible signers in the epoch $\mathfrak{E}$ from some public (untrusted) repository. Check validity of V through $\text{EpochVerify}_{gpk}(V, \mathfrak{E})$.

Note that while the timestamp τ (second step) included in epoch information will not contribute anything significant to formal security, it is a crucial component in practice. If a user was ever included in any epoch $\mathfrak{E}$, he can always create valid signatures with respect to $\mathfrak{E}$. Our formal security definitions merely require that he cannot create signatures for epochs $\mathfrak{E}'$ where he was revoked. It is the application's task to decide whether or not the supplied epoch information is acceptably recent – using the timestamp τ retrieved through ExtractTimestamp($\mathfrak{E}$) as an indicator. For example, emails should be signed with epoch information from the same day they were sent (or received); contracts should be signed with the epoch according to the document's date.

Typical verifiers will not need to do the optional third step of retrieving and checking the set V of potential signers. Indeed, if this step is omitted, a group signature scheme with accumulator revocation has the convenient property of *offline-verifiability*, i.e. signatures can be checked with only the (short) epoch information $\mathfrak{E}$ sent alongside the message and signature. However, the capability to view V adds transparency because verifiers get an insight who exactly may have produced some valid signature (rather than only being guaranteed the vague notion of "*someone who was not revoked*"). Note that in contrast to ring signatures, verifiers do not *need* to know the specific group that signed a message to check validity. This is also an advantage over revocation lists: there, verifiers need a list of revocation tokens for revoked users; in group signature schemes with accumulator revocation the information needed to verify a signature is (typically) independent of the number of revoked users.

Finally, the group manager in group signature schemes with accumulator revocation has the following responsibilities:

- Maintain a set $V \subseteq U$ of valid (non-revoked) users.
- Periodically publish epoch information $\mathfrak{E}$ for the current timestamp τ created through $\text{EpochCreate}_{gmsk}(V, \tau)$.
- If necessary (e.g., in case of legal dispute), use $\text{Open}_{gmsk}(m, \mathfrak{E}, \sigma)$ to trace a signature to its signer.

Note that while epoch-based schemes are often based on fixed time slots (i.e. an epoch is defined to be valid for some a-priori fixed interval of time, for example daily), applications could implement instant revocation mechanics using incremental IDs as timestamps and maintain an additional updateable table that maps a timestamp ID to the time interval

where it was valid. This is a good approach in cases where a delay in revocation (i.e. waiting for the current epoch to expire) cannot be accepted but it adds communication complexity for querying that table.

Of course, revocation for (normal) group signatures can be trivially implemented by invalidating an old group's public key and setting up a completely new group. In our scheme with accumulator revocation, we also need to distribute new information to every group member: the new epoch $\mathfrak{E}$ and either V or witnesses $wit_{V,i}$. The key difference is that the $\mathfrak{E}$ and V are public information and hence can be stored in public repositories, so there is no need for elaborate private key transmission mechanisms and the setup algorithm (which needs to be completely trusted to not leak secret keys) only has to be run once. In this context it is also possible that some untrusted third party distributes $\mathfrak{E}$ and $wit_{V,i}$ to any (unauthenticated) requester.

5.1.2 Security definitions

Our security requirements are expressed in three parts:

- **Full Anonymity**: It should be infeasible to distinguish signatures created by two *non-revoked* users.
- **Full Traceability**: It should be infeasible for any set of corrupted users to create a valid signature that is traced to some uncorrupted user by Open (or cannot be traced at all).
- **Unforgeability**: Given any set of revoked users' secret keys, it should be infeasible to create a valid signature.

Note that in contrast to normal group signatures, in this variant *fully-traceability* does not imply *unforgeability*. This is because full traceability does not consider revocation (which Definition 2.13 also does not account for), but unforgeability does. We chose not to include any revocation-related requirements in *full traceability*: Suppose some forged signature by a revoked user can still be correctly traced to the forger; then it would be unnatural to assume traceability broken. By this reasoning, revocation does not play an integral part in *full traceability*. All revocation-related requirements are instead covered by *unforgeability*, which roughly states that no user can create a signature that he should not be able to create.

We start by defining *full anonymity*. The most important difference to full anonymity in regular group signatures (Definition 2.12) is that for any epoch, it is trivial to distinguish accumulated and non-accumulated users' signatures (non-accumulated users should not be able to create a valid signature). Hence, the following definition of full anonymity requires only that it is hard to distinguish two users that both have signing rights in an epoch [1].

[1] One could argue that this is also the case for group signatures with revocation tokens. However, full anonymity as per Definition 2.12 does not model this

Definition 5.2 (Full Anonymity)**.** A group signature scheme with accumulator revocation Π has *full anonymity* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{anon}}_{\Pi,\mathcal{A}}(\lambda, n) := |\Pr[\mathrm{Exp}^{\mathrm{anon}-1}_{\Pi,\mathcal{A}}(\lambda, n) = 1] - \Pr[\mathrm{Exp}^{\mathrm{anon}-0}_{\Pi,\mathcal{A}}(\lambda, n) = 1]| \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiments $\mathrm{Exp}^{\mathrm{anon}-b}_{\Pi,\mathcal{A}}(\lambda, n)$ for $b \in \{0, 1\}$ work as follow:

1. $(U, gpk, gmsk, (sk[i])_{i\in U}) \leftarrow \mathrm{KeyGen}(1^\lambda, 1^n)$.

2. $\mathcal{A}$ is given $U, gpk, (sk[i])_{i\in U}$, and oracle access to $\mathrm{EpochCreate}_{gmsk}(\cdot,\cdot)$ and to $\mathrm{Open}_{gmsk}(\cdot,\cdot,\cdot)$.

3. Then $\mathcal{A}$ outputs a set $V^* \subseteq U$, epoch information $\mathfrak{E}^*$, two identities $(i_0, i_1) \in (V^*)^2$ and a message m^* (if $(i_0, i_1) \notin (V^*)^2$, the experiment aborts and outputs 0).

4. $\mathcal{A}$ is given a signature $\sigma^* \leftarrow \mathrm{Sign}_{sk[i_b]}(m^*, \mathfrak{E}^*, \mathrm{WitCreate}_{gpk}(V^*, i_b))$ created using $sk[i_b]$.

5. $\mathcal{A}$ continues to have oracle access to $\mathrm{Open}_{gmsk}(\cdot,\cdot,\cdot)$ and $\mathrm{EpochCreate}_{gmsk}(\cdot,\cdot)$.

6. In the end, $\mathcal{A}$ outputs a bit b'.

7. If $\mathrm{EpochVerify}_{gpk}(V^*, \mathfrak{E}^*) = 0$ or $\mathcal{A}$ has queried $\mathrm{Open}_{gmsk}(m^*, \mathfrak{E}^*, \sigma^*)$ after receiving the signature σ^*, the experiment outputs 0, otherwise it outputs b'.

Similar to normal group signatures, full anonymity states the following: The only way to reliably get any distinguishing information about two *non-revoked* signers is to consult Open for that signature. This even holds when all secret keys except the group manager's $gmsk$ (which can always be used to open signatures) are compromised.

In this definition, $\mathcal{A}$ may adaptively choose the set $V^* \subseteq U$ and the epoch it wants to attack. It is restricted to choose $i_0, i_1 \in V^*$ (because revoked users have the distinguishing feature of not being able to create valid signatures). For the chosen epoch information $\mathfrak{E}^*$, we require that $\mathrm{EpochVerify}_{gpk}(V^*, \mathfrak{E}^*) = 1$, i.e. $\mathfrak{E}^*$ is valid epoch information for V^*.

The following observation establishes that it is not an unreasonable restriction to have the attacker fix a single epoch for the two signatures to distinguish in Step 3 (as opposed to distinguishing, say, two signatures from two different epochs).

Observation 5.3. Definition 5.2 implies unlinkability of signatures (from different epochs).

Unlinkability intuitively means that no attacker, given a list of signatures (for potentially different messages, different epochs) can reliably point out two signatures on that list that were created by the same signer; in particular, some form of anonymity is preserved over different epochs. We omit the (non-trivial) formal definition and refer to [BMW03] for a discussion on unlinkability. We informally argue that our observation holds: if there is an attacker $\mathcal{B}$ that can reliably link two signatures by the same signer, an attacker $\mathcal{A}$ against full anonymity may use this as an oracle: to check whether σ^* corresponds to i_0 or i_1, the attacker may query a signature σ' for i_0 (on another epoch) and let $\mathcal{B}$ try to link σ^* and σ'. If $\mathcal{B}$ is reasonably good at linking and it succeeds at linking σ^*, σ', then i_0 was probably the signer of σ^*.

The following definition of *full traceability* is based on Definition 2.13 and is largely unmodified except for syntactical changes.

Definition 5.4 (Full traceability)**.** A group signature scheme with accumulator revocation Π has *full traceability* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{trace}}_{\Pi,\mathcal{A}}(\lambda, n) := \Pr[\mathrm{Exp}^{\mathrm{trace}}_{\Pi,\mathcal{A}}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{trace}}_{\Pi,\mathcal{A}}(\lambda, n)$ is defined as follows:

1. $(U, gpk, gmsk, (sk[i])_{i=1}) \leftarrow \mathrm{KeyGen}(1^\lambda, 1^n)$.
2. $\mathcal{A}$ is given $U, gmsk, gpk$, oracle access to $\mathrm{Sign}_{sk[\cdot]}(\cdot,\cdot,\cdot)$, and oracle access to $i \mapsto sk[i]$.
3. Eventually, $\mathcal{A}$ outputs a message m^*, epoch info $\mathfrak{E}^*$, and a signature σ^*.
4. If $\mathrm{Verify}_{gpk}(m^*, \mathfrak{E}^*, \sigma^*) = 1$ and $\mathrm{Open}_{gmsk}(m^*, \mathfrak{E}^*, \sigma^*) = \perp$, the experiment outputs 1.
5. If $\mathrm{Verify}_{gpk}(m^*, \mathfrak{E}^*, \sigma^*) = 1$ and $\mathrm{Open}_{gmsk}(m^*, \mathfrak{E}^*, \sigma^*) = i \in U$, and $\mathcal{A}$ has neither queried $sk[i]$ nor $\mathrm{Sign}_{sk[i]}(m^*, \mathfrak{E}^*, \cdot)$, then the experiment outputs 1.
6. Otherwise it outputs 0.

Full traceability states that an attacker, even given $gmsk$ and some adaptively chosen set of corrupted users' secret keys cannot create a signature that Open traces to an uncorrupted user or cannot trace to any user.

$\mathcal{A}$ has access to an oracle supplying user keys (which models corruption) and to a signing oracle to create signatures of (uncorrupted) users (which models that attackers may observe signatures from other users).

Note that supplying $\mathcal{A}$ with $gmsk$ in this experiment implies that even the group manager cannot create signatures on behalf of the users. In particular, it must be

infeasible to derive user keys from $gmsk$. As a consequence, in a scheme with full traceability the group manager cannot expand the group dynamically (i.e. user keys need to be computed by some external setup algorithm in the beginning and distributed without involvement of the group manager). This is also an restriction in normal group signature with full traceability (Definition 2.13) and we will not consider this problem further.

Note that this definition does not preclude creating signatures for revoked users. Full traceability only guarantees that *if* a revoked user creates a signature, it can be traced to his identity. However, since verifying users do not generally have access to $gmsk$ to trace the signature, this is not sufficient to discover revocation. In the following, *Unforgeability* will guarantee that revoked users are unable to create signatures:

Definition 5.5 (Unforgeability)**.** A group signature scheme with accumulator revocation Π has *unforgeable signatures* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n) := \Pr[\mathrm{Exp}^{\mathrm{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n)$ is defined as follows:

1. $(U, gpk, gmsk, (sk[i])_{i=1}^{n}) \leftarrow \mathrm{KeyGen}(1^\lambda, 1^n)$.
2. $\mathcal{A}$ is given $U, gpk, gmsk$ and oracle access to $\mathrm{Sign}_{sk[\cdot]}(\cdot,\cdot,\cdot)$, and $i \mapsto sk[i]$.
3. Eventually, $\mathcal{A}$ outputs a message m^*, a set $V^* \subseteq U$, epoch info $\mathfrak{E}^*$ and a signature σ^*.
4. If $\{i \in U \mid \mathcal{A} \text{ queried } sk[i]\} \cap V^* \neq \emptyset$ or $\mathrm{EpochVerify}_{gpk}(V^*, \mathfrak{E}^*) = 0$, the experiment aborts and outputs 0.
5. If $\mathrm{Verify}_{gpk}(m^*, \mathfrak{E}^*, \sigma^*) = 1$ and $\mathcal{A}$ has not queried $\mathrm{Sign}_{sk[\cdot]}(m^*, \mathfrak{E}^*, \cdot)$, the experiment outputs 1.
6. Otherwise, it outputs 0.

For unforgeability, $\mathcal{A}$ is given oracle access to Sign and (in contrast to [BMW03]) $gmsk$. It may adaptively corrupt users by querying the oracle for $i \mapsto sk[i]$.

$\mathcal{A}$ succeeds if it manages to create a signature on any message m^* for an epoch where all corrupted users are revoked (cf. Step 4) without querying the Sign oracle for m^* on an uncorrupted user (cf. Step 5). In other words, any set of colluding revoked users should be unable to create valid signatures, even if given $gmsk$. This also implies unforgeability in the traditional sense: without any user keys, it is infeasible to create a valid signature (cf. Definition 2.14). This is covered through attackers that do not corrupt any users (i.e. do not query the $sk[\cdot]$ oracle).

Lastly, we state our security requirement that valid epoch information can only be created by the group manager:

Definition 5.6 (Unforgeability of epoch information)**.** A group signature scheme with accumulator revocation Π has *unforgeable epochs* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{forge}}_{\Pi,\mathcal{A}}(\lambda, n) := \Pr[\mathrm{Exp}^{\mathrm{forge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{forge}}_{\Pi,\mathcal{A}}(\lambda, n)$ is defined as follows:

1. $(U, gpk, gmsk, (sk[i])_{i=1}^{n}) \leftarrow \mathrm{KeyGen}(1^{\lambda}, 1^{n})$.
2. $\mathcal{A}$ is given $U, gpk, (sk[i])_{i=1}^{n}$, oracle access to $\mathrm{Open}_{gmsk}(\cdot,\cdot,\cdot)$, and oracle access to $\mathrm{EpochCreate}_{gmsk}(\cdot,\cdot)$.
3. Eventually, $\mathcal{A}$ outputs epoch information $\mathfrak{E}^*$, a set $V^* \subseteq U$, a message m^* and a signature σ^*.
4. If $\mathcal{A}$ has queried $\mathrm{EpochCreate}_{gmsk}(V^*, \mathrm{ExtractTimestamp}(\mathfrak{E}^*))$, the experiment aborts and outputs 0.
5. If $\mathrm{EpochVerify}_{gpk}(V^*, \mathfrak{E}^*) = 1$ or $\mathrm{Verify}_{gpk}(m^*, \mathfrak{E}^*, \sigma^*) = 1$, the experiment outputs 1.
6. Otherwise, it outputs 0.

Unforgeability of epoch information states that without $gmsk$, it is infeasible to create any epoch information $\mathfrak{E}^*$ for any timestamp τ that is accepted by EpochVerify or Verify. This is not covered by any of the other security definitions (where generally, the attacker is given the means to create epoch information). However, unforgeability of epoch information is essential in order to achieve actual unforgeability in practice: if epoch information could be forged, revoked users may be able to create epoch information $\mathfrak{E}'$ where they are not revoked and create a signature with respect to $\mathfrak{E}'$, circumventing their actual revocation status. While unforgeability guarantees that for any epoch information, it is infeasible to create a signature for revoked users, *unforgeability of epoch information* guarantees that the epoch information supplied alongside a message is genuine. Together, they imply that when a user receives $(m, \mathfrak{E}, \sigma)$ (a message, epoch information and a signature), and $\mathrm{Verify}_{gpk}(m, \mathfrak{E}, \sigma) = 1$, he can trust that the message was signed by a user who the group manager did not consider revoked at timestamp $\tau = \mathrm{ExtractTimestamp}(\mathfrak{E})$.

In summary, the definitions above imply the following informal observations:

- The group manager needs to be trusted *only* for anonymity, as he can reliably uncover identities (full traceability), and for his revocation decisions, as he publishes epoch information. The group manager cannot sign messages on behalf of any user (unforgeability) and he cannot blame a message on a user who did not sign it (full traceability)[2].

- If a signature is valid according to $\text{Verify}_{gpk}(m, \mathfrak{E}, \sigma)$, the verifier can be sure that:
  - The group manager created $\mathfrak{E}$ and $\tau = \text{ExtractTimestamp}(\mathfrak{E})$ is the epoch's genuine timestamp (unforgeability of epoch information).
  - The signer chose $\mathfrak{E}$ to sign this message (unforgeability).
  - The signer's signing rights were not revoked in $\mathfrak{E}$ (unforgeability).
  - If some (untrusted) party can supply a set V where $\text{EpochVerify}_{gpk}(V, \mathfrak{E}) = 1$, then some user $i \in V$ signed m (unforgeability).

- When a signer publishes a valid signature in an epoch $\mathfrak{E}$ where the set of valid users is V (i.e. $\text{EpochVerify}_{gpk}(V, \mathfrak{E}) = 1$), his signature is indistinguishable from signatures produced by other users in V, unless the group manager chooses to reveal identities. This remains true even if the signer is revoked in another epoch (full anonymity).

- When a signer publishes a valid signature for an epoch $\mathfrak{E}$ and another signature for epoch $\mathfrak{E}'$, nobody except the group manager can tell that these signatures were created by the same signer, unless in both epochs all other users are revoked (full anonymity).

5.2 A generic construction of group signature schemes with accumulator revocation

In this Section, we will construct a group signature scheme with accumulator revocation from any normal group signature scheme (Section 2.6). This serves as a proof of concept that such systems exist, and allow us to get more insights as to how to construct them.

The idea of our construction is to start with any group signature scheme Π_G and construct a group signature scheme with accumulator revocation (Definition 5.1) by adding to signatures of Π_G a signature of our ring signature scheme Π_R (Construction 4.5) that certifies revocation status.

We start with the formal definition of our construction, then we proceed to the security proofs.

[2] Of course, if the group manager is treated as a black box, he can claim any user as the signer. Full traceability guarantees that even the group manager cannot prepare a signature to look like it originated from some user he does not know the key for. In practice, this requires some oversight when executing Open.

5.2.1 Construction

Construction 5.7. Let $\Pi_G = (\text{KeyGen}^{(G)}, \text{Sign}^{(G)}, \text{Verify}^{(G)}, \text{Open}^{(G)})$ be a group signature scheme and $\Pi_R = (\text{KeyGen}^{(R)}, \text{Sign}^{(R)}, \text{Verify}^{(R)})$ the ring signature scheme in Construction 4.5. Let $\Pi_S = (\text{KeyGen}^{(S)}, \text{Sign}^{(S)}, \text{Verify}^{(S)})$ be a signature scheme with message space $\{0,1\}^*$ and secure against chosen-message attacks.

We construct a group signature scheme with accumulator revocation Π as follows:

- $\text{KeyGen}(1^\lambda, 1^n)$ runs
  - $\text{KeyGen}^{(G)}(1^\lambda, 1^n)$ to obtain a set U with $|U| = n$, the scheme's public key gpk_G, its group manager secret $gmsk_G$ and its secret keys $sk_G[i]$ for $i \in U$. Without loss of generality, we assume that $U = [n]$ [3].
  - $\text{KeyGen}^{(R)}(1^\lambda, 1^n)$ to obtain the ring signature's public key pk_R and secret keys $(sk_R[i])_{i\in[n]}$.
  - $\text{KeyGen}^{(S)}(1^\lambda)$ to obtain the signature scheme's public key pk_S and secret key sk_S.

  It sets $gpk = (gpk_G, pk_R, pk_S)$, $gmsk = (gmsk_G, sk_S, gpk)$, and $sk[i] = (sk_G[i], sk_R[i], gpk)$ for $i \in U = [n]$. It outputs $(U, gpk, gmsk, (sk[i])_{i\in U})$.

- $\text{EpochCreate}_{gmsk}(V, \tau)$ computes the accumulator $acc_V = \prod_{j\in V} g_{n+1-j}$ using pk_R and runs $\text{Sign}^{(S)}_{sk_S}((acc_V, \tau))$ to obtain a signature θ. It outputs epoch information $\mathfrak{E} = (acc_V, \tau, \theta)$.

- $\text{ExtractTimestamp}((acc, \tau, \theta))$ outputs τ.

- $\text{EpochVerify}_{gpk}(V, (acc, \tau, \theta))$ returns 1 if and only if $\text{Verify}^{(S)}_{pk_S}((acc, \tau), \theta) = 1$ and $acc = acc_V$ (computed using pk_R).

- $\text{WitCreate}_{gpk}(V, i)$ outputs $wit_{V,i} = \prod_{j\in V\setminus\{i\}} g_{n+1-j+i}$ computed with values from pk_R.

- $\text{Sign}_{sk[i]}(m, \mathfrak{E}, wit)$ parses $\mathfrak{E}$ as (acc, τ, θ), sets $m' = (m, \mathfrak{E})$ and creates $\sigma_R = \text{Sign}^{(R)}_{sk_R[i]}(m', acc, wit)$ (cf. Observation 4.7) and $\sigma_G = \text{Sign}^{(G)}_{sk_G[i]}(m')$. The signature is $\sigma = (\sigma_R, \sigma_G)$.

- $\text{Verify}_{gpk}(m, \mathfrak{E}, (\sigma_R, \sigma_G))$ parses $\mathfrak{E}$ as (acc, τ, θ), sets $m' = (m, \mathfrak{E})$ and returns 1 if $\text{Verify}^{(S)}_{pk_S}((acc, \tau), \theta) = 1$, $\text{Verify}^{(R)}_{pk_R}(m', acc, \sigma) = 1$, and $\text{Verify}^{(G)}_{gpk_G}(m') = 1$.

- $\text{Open}_{gmsk}(m, \mathfrak{E}, (\sigma_R, \sigma_G))$ sets $m' = (m, \mathfrak{E})$. It returns what $\text{Open}^{(G)}_{gmsk_G}(m', \sigma_G)$ outputs.

[3] For arbitrary U, one can employ some fixed one-to-one mapping between the users of the group signature scheme U and the users of the ring signature scheme $[n]$.

Note that a signature of this scheme consists of two signatures: σ_G from Π_G and σ_R from Π_R. σ_G is essentially used to provide Open functionality. σ_R provides the revocation mechanism, ensuring that signers had proper signing rights when the signature was created. Both σ_G and σ_R are signatures on the message $m' = (m, \mathfrak{E})$. This corresponds to the requirement, put forth by our security definitions, that signers should not later be able to claim a different epoch for their signature (cf. Definition 5.5).

Note that in this particular scheme, one can efficiently update witnesses (directly through WitUpdate of the accumulator construction (Construction 3.5) using the values in pk_R. Furthermore, one can efficiently check whether or not witnesses are correctly computed (for example, by a third party) by using the Verify operation of the accumulator construction.

In the remainder of this chapter, we will prove this construction secure.

Theorem 5.8. *Construction 5.7 is a* correct *group signature scheme with accumulator revocation (Definition 5.1) that fulfills* full anonymity *(Definition 5.2),* full traceability *(Definition 5.4),* unforgeability *(Definition 5.5) and* unforgeability of epoch information *(Definition 5.6) if* Π_G *is a secure group signature scheme and* Π_S *is a CPA-secure signature scheme.*

Proof. We subdivide the proofs into the following Lemmas: Lemma 5.9 (correctness), Lemma 5.10 (anonymity), Lemma 5.11 (traceability), Lemma 5.14 (unforgeability), and Lemma 5.15 (unforgeability of epoch information). The theorem results from these lemmas. □

5.2.2 Correctness

Correctness of the scheme is mostly derived from the fact that signatures simply consist of two signatures of the correct schemes Π_G, Π_R.

Lemma 5.9. *Construction 5.7 is* correct *(Definition 5.1)*

Proof. Let $\lambda, n \in \mathbb{N}, (U, gpk, gmsk, (sk[i])_i) \in [\text{KeyGen}(1^\lambda, 1^n)], V \subseteq U, i \in V, \tau \in \{0,1\}^*, \mathfrak{E} = (acc, \tau, \theta) \in [\text{EpochCreate}_{gmsk}(V, \tau)]$. Let m be a message, $m' := (m, \mathfrak{E})$, and $\sigma = (\sigma_R, \sigma_G) \in [\text{Sign}_{sk[i]}(m, \mathfrak{E}, wit_{V,i})]$

Correctness of Verify

By construction,

$$\begin{aligned}&\text{Verify}_{gpk}(m, \mathfrak{E}, \sigma) = 1\\ &\Leftrightarrow \text{Verify}^{(S)}_{pk_S}((acc, \tau), \theta) = 1 \wedge \text{Verify}^{(R)}_{pk_R}(m', acc, \sigma) = 1 \wedge \text{Verify}^{(G)}_{gpk_G}(m') = 1\end{aligned}$$

Since by definition of EpochCreate, θ is a signature created with $\text{Sign}^{(S)}_{sk_S}((acc, \tau))$, correctness of Π_S yields $\text{Verify}^{(S)}_{pk_S}((acc, \tau), \theta) = 1$. By definition of Sign, σ_R is a signature

created with $\text{Sign}^{(R)}_{sk_R[i]}(m', acc, wit)$, and so $acc = acc_V, wit = wit_{V,i}$ and correctness of Π_R (with Observation 4.7) yields $\text{Verify}^{(R)}_{pk_R}(m', acc, \sigma) = 1$. Finally, by definition of Sign, σ_G is a signature created with $\text{Sign}^{(G)}_{sk_G[i]}(m')$, hence correctness of Π_G yields $\text{Verify}^{(G)}_{gpk_G}(m') = 1$.

Correctness of Open

$\text{Open}_{gmsk}(m, \mathfrak{E}, \sigma) = \text{Open}^{(G)}_{gmsk_G}(m', \sigma_G) = i$ using that $\sigma_G \in [\text{Sign}^{(G)}_{sk_G[i]}(m')]$ and correctness of Π_G.

Correctness of EpochVerify

$\text{EpochVerify}_{gpk}(V, \mathfrak{E}) = 1 \Leftrightarrow \text{Verify}^{(S)}_{pk_S}((acc, \tau), \theta) = 1 \wedge acc = acc_V$. It holds that $\text{Verify}^{(S)}_{pk_S}((acc, \tau), \theta) = 1$ because Π_S is correct and $\theta \in \text{Sign}^{(S)}_{sk_S}((acc_V, \tau))$. $acc = acc_V$ holds by definition of EpochCreate.

Correctness of ExtractTimestamp

Follows immediately from the definition of ExtractTimestamp.

Conclusion

Thus we have shown the four necessary relations as required by Definition 5.1. □

5.2.3 Anonymity

We will now prove anonymity of the scheme. Every signature of our scheme consists of two signatures σ_G, σ_R and either of them (or their combination) could potentially leak information about the signer. However, using a hybrid argument, we will be able to show that anonymity follows from anonymity of Π_G and Π_R (according to their respective anonymity definitions).

Lemma 5.10. *Construction 5.7 fulfills* full anonymity *(Definition 5.2)*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time algorithm against full anonymity of Π. Let $\lambda, n \in \mathbb{N}$.

Hybrid argument

We define a probability space over (1) the random choice of $gpk, gmsk$ and $(sk[i])_{i \in U}$ distributed as in $\text{KeyGen}(1^\lambda, 1^n)$, (2) the random bits of $\mathcal{A}$, (3) the random bits used for EpochCreate queries, and (4) the random bits used to create the challenge signature parts σ^*_R and σ^*_G.

In this probability space, for $b_R, b_G \in \{0, 1\}$, let H_{b_R, b_G} be a random variable that takes on the value of $\mathcal{A}$'s outputs when it is run against a full anonymity experiment of Π with the following modification: the challenge signature parts are computed as

$\sigma_G^* \leftarrow \mathrm{Sign}^{(G)}_{sk_G[i_{b_G}]}((m^*, \mathfrak{E}^*))$ and $\sigma_R^* \leftarrow \mathrm{Sign}^{(R)}_{sk_R[i_{b_R}]}((m^*, \mathfrak{E}^*), acc^*, wit_{V^*, i_{b_R}})$, i.e. σ_G is created for i_{b_G} and σ_R for i_{b_R}. Note that in the original experiment $\mathrm{Exp}^{\mathrm{anon}-b}$, we have $b_G = b_R = b$.

Consequently,

$$\begin{aligned}
&\mathrm{Adv}^{\mathrm{anon}}_{\Pi,\mathcal{A}}(\lambda, n) \\
=&|\Pr[\mathrm{Exp}^{\mathrm{anon}-0}_{\Pi,\mathcal{A}}(\lambda, n) = 1] - \Pr[\mathrm{Exp}^{\mathrm{anon}-1}_{\Pi,\mathcal{A}}(\lambda, n) = 1]| \\
=&|\Pr[H_{0,0} = 1] - \Pr[H_{1,1} = 1]| \\
=&|\Pr[H_{0,0} = 1] - \Pr[H_{1,0} = 1] + \Pr[H_{1,0} = 1] - \Pr[H_{1,1} = 1]| \\
\leq&|\Pr[H_{0,0} = 1] - \Pr[H_{1,0} = 1]| + |\Pr[H_{1,0} = 1] - \Pr[H_{1,1} = 1]|
\end{aligned}$$

In the following, we will use $\mathcal{A}$ to construct an algorithm $\mathcal{A}_R$ against *anonymity against early full key exposure* of Π_R (Definition 2.9), and another algorithm $\mathcal{A}_G$ against *full anonymity* of Π_G (Definition 2.12). By construction, it will hold that $|\Pr[H_{0,0}] - \Pr[H_{1,0}]| = \mathrm{Adv}^{\mathrm{anon}}_{\Pi_R,\mathcal{A}_R}(\lambda, n)$ and $|\Pr[H_{1,0}] - \Pr[H_{1,1}]| = \mathrm{Adv}^{\mathrm{anon}}_{\Pi_G,\mathcal{A}_G}(\lambda, n)$. Both of these are negligible in λ because Π_R and Π_G are secure. This will conclude the proof.

Constructing A_R

$\mathcal{A}_R$ against *anonymity against early full key exposure* of Π_R (Definition 2.9) works as follows: it receives pk_R, randomness ω, and U from its experiment anon-b. It runs $(pk_R, (sk_R[i])_{i\in[n]}) \leftarrow \mathrm{KeyGen}^{(R)}(1^\lambda, 1^n)$ with randomness ω. $\mathcal{A}_R$ then completes the setup of Π by running $(pk_S, sk_S) \leftarrow \mathrm{KeyGen}^{(S)}(1^\lambda)$ and $(gpk_G, U, gmsk_G, (sk_G[i])_{i\in U}) \leftarrow \mathrm{KeyGen}^{(G)}(1^\lambda, 1^n)$. A_R hands $U, gpk = (gpk_G, pk_R, pk_S)$ and $sk[i] = (sk_G[i], sk_R[i], gpk)$ for $i \in U$ to $\mathcal{A}$.

When $\mathcal{A}$ queries Open or EpochCreate, $\mathcal{A}_R$ computes the answer using $gmsk = (gmsk_G, sk_S, gpk)$.

Eventually, $\mathcal{A}$ outputs $V^* \subseteq U, \mathfrak{E}^* = (acc^*, \tau^*, \theta^*), (i_0, i_1) \in (V^*)^2, m^*$. A_R outputs $(i_0, i_1), V^*$, and $m' = (m^*, \mathfrak{E}^*)$ to receive a signature $\sigma_R^* \leftarrow \mathrm{Sign}^{(R)}_{sk_R[i_b]}(m', V^*)$ for i_b. It computes $\sigma_G^* \leftarrow \mathrm{Sign}^{(G)}_{sk_G[i_0]}(m')$ for identity i_0 and hands $\sigma^* = (\sigma_R^*, \sigma_G^*)$ to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a bit b'. If $\mathrm{EpochVerify}_{gpk}(V^*, \mathfrak{E}^*) = 0$ or $\mathcal{A}$ has queried $\mathrm{Open}_{gmsk}(m^*, \mathfrak{E}^*, \sigma^*)$, $\mathcal{A}_R$ outputs 0. Otherwise, $\mathcal{A}_R$ outputs b'.

By construction, when $\mathcal{A}_R$ is run against Π_R's anon-0, it perfectly simulates the experiment underlying $H_{0,0}$. When $\mathcal{A}_R$ is run against anon-1, it perfectly simulates the experiment underlying $H_{1,0}$. Formally, using that the random variables underlying our probability space are computed by $\mathcal{A}_R$ or the anon-b experiment with the correct distributions,

$$H_{1,0} = \mathrm{Exp}^{\mathrm{anon}-1}_{\Pi_R,\mathcal{A}_R}(\lambda, n)$$

and

$$H_{0,0} = \mathrm{Exp}^{\mathrm{anon}-0}_{\Pi_R,\mathcal{A}_R}(\lambda, n)$$

which implies that $|\Pr[H_{0,0} = 1] - \Pr[H_{1,0} = 1]| \leq \mu_R(\lambda)$ for some negligible μ_R, using that Π_R is secure.

Constructing A_G

$\mathcal{A}_G$ against *full anonymity* of Π_G (Definition 2.12) works as follows: it receives U, gpk_G, $(sk_G[i])_{i \in U}$ from its experiment anon-b. It completes the setup of our scheme Π by running $(pk_S, sk_S) \leftarrow \text{KeyGen}^{(S)}(1^\lambda)$ and $(pk_R, (sk_R[i])_{i \in [n]}) \leftarrow \text{KeyGen}^{(R)}(1^\lambda, 1^n)$ itself. $\mathcal{A}_G$ hands $U, gpk = (gpk_G, pk_R, pk_S)$, and $sk[i] = (sk_G[i], sk_R[i], gpk)$ for $i \in U$ to $\mathcal{A}$.

If $\mathcal{A}$ queries $\text{Open}_{gmsk}(m, \mathfrak{E}, (\sigma_R, \sigma_G))$, $\mathcal{A}_G$ queries $\text{Open}^{(G)}_{gmsk_G}((m, \mathfrak{E}), \sigma_G)$ and returns the result to $\mathcal{A}$. If $\mathcal{A}$ queries $\text{EpochCreate}_{gmsk}(V, \tau)$, $\mathcal{A}_G$ answers using sk_S and pk_R.

Eventually, $\mathcal{A}$ outputs $V^* \subseteq U, \mathfrak{E}^* = (acc^*, \tau^*, \theta^*), (i_0, i_1) \in (V^*)^2, m^*$. $\mathcal{A}_G$ outputs (i_0, i_1) and $m' = (m^*, \mathfrak{E}^*)$ to receive a signature $\sigma^*_G \leftarrow \text{Sign}^{(G)}_{sk_G[i_b]}(m')$ for i_b. It computes $\sigma^*_R \leftarrow \text{Sign}^{(R)}_{sk_R[i_1]}((m', acc^*), wit_{V^*, i_1})$ for identity i_1 and hands $\sigma^* = (\sigma^*_R, \sigma^*_G)$ to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a bit b'. If $\text{EpochVerify}_{gpk}(V^*, \mathfrak{E}^*) = 0$, $\mathcal{A}_G$ outputs 0, otherwise $\mathcal{A}_G$ outputs b'. Note that if $\mathcal{A}$ did not query $\text{Open}_{gmsk}(m^*, \mathfrak{E}^*, \sigma^*)$ after receiving the challenge signature σ^*, then $\mathcal{A}_G$ did not query $\text{Open}_{gmsk_G}((m^*, \mathfrak{E}^*), \sigma^*_G)$ after receiving its signature σ^*_G.

By construction, when $\mathcal{A}_G$ is run against Π_G's anon-0, it perfectly simulates the experiment underlying $H_{1,0}$. When $\mathcal{A}_R$ is run against anon-1, it perfectly simulates the experiment underlying $H_{1,1}$. Formally, using that the random variables underlying our probability space are computed by $\mathcal{A}_G$ or the anon-b experiment with the correct distributions,

$$H_{1,0} = \text{Exp}^{\text{anon}-1}_{\Pi_G, \mathcal{A}_G}(\lambda, n)$$

and

$$H_{1,1} = \text{Exp}^{\text{anon}-0}_{\Pi_G, \mathcal{A}_G}(\lambda, n)$$

which implies that $|\Pr[H_{1,0} = 1] - \Pr[H_{1,1} = 1]| \leq \mu_G(\lambda)$ for some negligible μ_G, using that Π_G is secure.

Conclusion

Overall,

$$\begin{aligned}
&\text{Adv}^{\text{anon}}_{\Pi, \mathcal{A}}(\lambda, n) \\
\leq &|\Pr[H_{0,0} = 1] - \Pr[H_{1,0} = 1]| + |\Pr[H_{1,0} = 1] - \Pr[H_{1,1} = 1]| \\
\leq &\mu_R(\lambda) + \mu_G(\lambda)
\end{aligned}$$

for all $n, \lambda \in \mathbb{N}$, where μ_R and μ_G are negligible (and hence, their sum is negligible), as required. □

5.2.4 Traceability

We will now prove full traceability of our scheme. Since the traceability definition for group signatures with accumulator revocation does not consider revocation, it suffices to reduce full traceability of our scheme to full traceability of Π_G.

Lemma 5.11. *Construction 5.7 fulfills* full traceability *(Definition 5.4)*

Proof. Let $\mathcal{A}$ a probabilistic polynomial-time algorithm against full traceability of Π. We construct an algorithm $\mathcal{A}_G$ against full traceability of the group signature scheme Π_G (Definition 2.13). $\mathcal{A}_G$ receives $U, gmsk_G, gpk_G$ from its full traceability game and completes the setup by running $(pk_S, sk_S) \leftarrow \text{KeyGen}^{(S)}(1^\lambda)$ and $(pk_R, (sk_R[i])_{i\in[n]}) \leftarrow \text{KeyGen}^{(R)}(1^\lambda, 1^n)$ itself. $\mathcal{A}_G$ hands $U, gpk = (gpk_G, pk_R, pk_S)$, and $gmsk = (gmsk_G, sk_S, gpk)$ to $\mathcal{A}$.

$\mathcal{A}_G$ answers signature queries $\text{Sign}_{sk[i]}(m, \mathfrak{E}, wit)$ of $\mathcal{A}$ with (σ_R, σ_G) by querying σ_G from its signing oracle and computing σ_R itself. It answers secret key queries $sk[i]$ with $(sk_G[i], sk_R[i], gpk)$ by querying $sk_G[i]$ from its game and using $sk_R[i]$ from the setup phase.

If $\mathcal{A}$ outputs a forgery $(m^*, \mathfrak{E}^*, (\sigma_R^*, , \sigma_G^*))$ accepted by its experiment, $\mathcal{A}_G$ outputs σ_G^* as a signature forgery for $m' = (m^*, \mathfrak{E}^*)$. Note that if $\text{Open}^{(G)}_{gmsk_G}(m', \sigma_G^*) = \text{Open}_{gmsk}(m^*, \mathfrak{E}^*, \sigma^*) \stackrel{!}{=} i \in U$, $\mathcal{A}_G$ neither queried $\text{Sign}_{sk_G[i]}(m')$, nor $sk_G[i]$ (as the forgery of $\mathcal{A}$ would not have been accepting otherwise). $\mathcal{A}_G$'s simulation for $\mathcal{A}$ is perfect and $\mathcal{A}_G$ wins its game if and only if $\mathcal{A}$ wins its game.

It follows that

$$\Pr[\text{Exp}^{\text{trace}}_{\Pi,\mathcal{A}}(\lambda, n)] = \Pr[\text{Exp}^{\text{trace}}_{\Pi_G,\mathcal{A}_G}(\lambda, n) = 1] \leq \mu(\lambda)$$

for some negligible function μ and for all $\lambda \in \mathbb{N}$ by the assumption that Π_G fulfills full traceability. □

5.2.5 Unforgeability

Now, we will prove unforgeability of our scheme. To forge a signature, one must forge both the group signature part σ_G and the ring signature part σ_R. We reduce unforgeability of our scheme to the security of Π_R, since it implements the accumulator-driven revocation mechanism.

However, there is a small complication: the unforgeability game of Π_R is gives an attacker oracle access to $\text{Sign}^{(R)}$ demanding the set V of signer identities. In contrast, the Sign operation of our group signature scheme with accumulator revocation takes $\mathfrak{E}, wit$ as input instead of V. Since it is infeasible to recreate a set V from $\mathfrak{E}$, we cannot use the $\text{Sign}^{(R)}$ oracle in this form in our reduction. Note that syntactically, this is not a problem since the first step that $\text{Sign}^{(R)}$ takes is computing acc_V and $wit_{V,i}$ from V and it never uses V again (cf. Observation 4.7). However, it is not immediately clear that using the scheme in this way is still secure.

For this reason, we introduce the following modified version of unforgeability for Π_R that is closer to the unforgeability experiment of our construction. In particular, it supports oracle access to $\text{Sign}^{(R)}$ with parameters acc, wit instead of V.

Definition 5.12 (Unforgeability w.r.t. insider corruption for accumulator signing)**.** Π_R has *unforgeable signatures for accumulator signing* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}_R$, there exists a negligible function μ such that

$$\text{Adv}^{\text{sigforge}'}_{\Pi_R,\mathcal{A}_R}(\lambda, n) := \Pr[\text{Exp}^{\text{sigforge}'}_{\Pi_R,\mathcal{A}_R}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\text{Exp}^{\text{sigforge}'}_{\Pi_R,\mathcal{A}_R}(\lambda, n)$ is defined as follows:

$(pk, U, (sk[i])_{i\in U}) \leftarrow \text{KeyGen}(1^\lambda, 1^n)$.

$\mathcal{A}_R$ is given pk, U, oracle access to $\text{Sign}^{(R)}_{sk[\cdot]}(\cdot,\cdot,\cdot)$ (where the arguments are a message, an accumulator value, and a witness, cf. Observation 4.7), and oracle access to $i \mapsto sk[i]$.

Eventually, $\mathcal{A}_R$ outputs a message m^*, a set $V^* \subseteq U$ and a signature σ^*.

If $\text{Verify}^{(R)}_{pk}(m^*, V^*, \sigma^*) = 1$ and $\mathcal{A}_R$ has neither queried $\text{Sign}^{(R)}_{sk[\cdot]}(m^*, acc_{V^*}, \cdot)$ nor $sk[i]$ for any $i \in V^*$, the experiment outputs 1. Otherwise it outputs 0.

Unfortunately, this variant of unforgeability neither directly implies nor is directly implied by the original one. On the one hand, $\mathcal{A}_R$ gains some power by querying $\text{Sign}^{(R)}$ even for accumulator values where it does not know a corresponding set; on the other hand, $\mathcal{A}_R$ is restricted to output signatures where it never queried $\text{Sign}^{(R)}$ for the specified message and acc_{V^*} as opposed to V^* in the original. This rules out attackers from the original game that find two sets $V \neq V'$ with $acc_V = acc_{V'}$, query $\sigma \leftarrow \text{Sign}^{(R)}_{sk[i]}(m, V)$ and output the valid forgery m, V', σ, which they did not query. However, the proof of the original unforgeability experiment (Section 4.2.4) can be adapted for the modified unforgeability experiment as well.

Lemma 5.13. Π_R *has* unforgeable signatures for accumulator signing *(Definition 5.12).*

Proof. We sketch the proof by pointing out the necessary changes to the proof of Lemma 4.12 and Lemma 4.13. For this proof, Sign denotes the operation as defined on Π_R.

For Lemma 4.12, we construct a simulator algorithm $\mathcal{B}$ that answers signature queries by simulating transcripts of the underlying protocol. Obviously, $\mathcal{B}$ can handle queries in the format $\text{Sign}_{sk[i]}(m', acc, wit)$ without problem as in the original (Step 5 of $\mathcal{B}$), V is merely used to compute acc_V. In Step 8, $\mathcal{B}$ should output fail if $\text{Sign}_{sk[\cdot]}(m^*, acc_{V^*}, \cdot)$

was queried as opposed to if $\text{Sign}_{sk[\cdot]}(m^*, V^*)$ was queried. The proof works completely analogously. With the latter change, we can still argue that $\text{Exp}^{\text{sigforge}'}_{\Pi_R,\mathcal{A}_R}(\lambda, n') = 1 \wedge \neg\text{fail}_5 \Leftrightarrow \neg\text{fail}_8 \wedge \neg\text{fail}_5$.

In Lemma 4.13, we construct an algorithm $\mathcal{E}$ that uses rewinding on $\mathcal{B}$ to output either an element of the NP-relation of the underlying Σ protocol or sets V', V'' with the same accumulator $acc_{V'} = acc_{V''}$. The construction itself can stay the same way. However, the condition in Step 8 (that the special hash index i was used to answer a query $\text{Sign}_{sk[\cdot]}(m, \hat{V})$) cannot be fulfilled anymore[4]. The analogous situation would be that i was used for a signature query $\text{Sign}_{sk[\cdot]}(m, acc_{\hat{V}}, \cdot)$. But in this case, $\mathcal{B}$ would have output fail with our changes above (as $\mathcal{A}_R$ is not allowed to query that and still claim it as a forgery). So in the modified experiment, i is *always* used to answer an explicit hash query, so Step 9 or Step 10 apply and output the expected values. The probability analysis stays exactly the same and results in the same bound relative to the success probability $\epsilon(\lambda)$ of $\mathcal{A}_R$ in $\text{Exp}^{(\text{sigforge}')}_{\Pi_R,\mathcal{A}_R}(\lambda, n')$.

Finally, the proof of Lemma 4.14 can be applied on the modified $\mathcal{E}$ and $\mathcal{B}$ as before, implying that Π_R has *unforgeable signatures for accumulator signing* as required. □

Note that this result is interesting independently of our group signatures scheme with accumulator revocation. It implies that our ring signature scheme is still secure if one redefines the $\text{Sign}^{(R)}$ and $\text{Verify}^{(R)}$ operations to omit V and take acc_V and $wit_{V,i}$ as input to the algorithms as noted in Observation 4.7. For example, this enables third parties to compute acc_V for some ring V, and distribute witnesses $wit_{V,i}$ to signers. Still, signatures are difficult to forge.

Finally, we remark that the problem of using the $\text{Sign}^{(R)}$ operation without knowing the set V (as defined in our construction above) does not come up in any other security proof of this section. For correctness, this is not a problem as syntactically, signing and verifying with acc_V (and $wit_{V,i}$) instead of V is simple as noted in Observation 4.7. In the anonymity experiment (Definition 5.2), the attacker is given all users' secret keys, hence there is no need for access to the signing oracle. For the challenge signature, the attacker against our construction is required to output the challenge set V^*, which can be handed to the ring signature's anonymity game to use for the $\text{Sign}^{(R)}$ algorithm. For the traceability proof, the ring signature scheme's security was not involved.

Using the adapted security experiment, a reduction can now consult the $\text{Sign}^{(R)}$ oracle of Π_R to create the σ_R portion of our scheme's signatures. This allows us to prove unforgeability of Π through a straight reduction to the modified notion of unforgeability (Lemma 5.13) for Π_R.

Lemma 5.14. *Construction 5.7 fulfills* unforgeability *(Definition 5.5)*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time algorithm against unforgeability of Π.

[4]This step corresponded to attackers that exploit accumulator collisions as explained right before this lemma.

We construct an algorithm A_R against unforgeability with accumulator signing (Definition 5.12) of Π_R.

$\mathcal{A}_R$ receives pk_R, U from its unforgeability experiment. It completes the setup with $(pk_S, sk_S) \leftarrow \text{KeyGen}^{(S)}(1^\lambda)$ and $(gpk_G, U, gmsk_G, (sk_G[i])_{i\in U}) \leftarrow \text{KeyGen}^{(G)}(1^\lambda, 1^n)$. A_R hands $U, gpk = (gpk_G, pk_R, pk_S)$ and $sk[i] = (sk_G[i], sk_R[i], gpk)$ for $i \in U$ to $\mathcal{A}$.

If $\mathcal{A}$ queries $\text{Sign}_{sk[i]}(m, \mathfrak{E}, wit)$, $\mathcal{A}_R$ parses $\mathfrak{E}$ as (acc, τ, θ) and queries its oracle for the signature $\sigma_R \leftarrow \text{Sign}^{(R)}_{sk_R[i]}((m, \mathfrak{E}), acc, wit)$. It computes $\sigma_G \leftarrow \text{Sign}^{(G)}_{sk_G[i]}((m, \mathfrak{E}))$ itself and hands the signature (σ_R, σ_G) to $\mathcal{A}$. If $\mathcal{A}$ queries $sk[i]$, $\mathcal{A}_R$ queries $sk_R[i]$ and hands $sk[i] = (sk_G[i], sk_R[i], gpk)$ to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs $m^*, V^*, \mathfrak{E}^* = (acc^*, \tau^*, \theta^*)$ and $\sigma^* = (\sigma^*_R, \sigma^*_G)$. If $\mathcal{A}$'s forgery is accepting, $\mathcal{A}_R$ outputs σ^*_R as a forgery for message $(m^*, \mathfrak{E}^*)$ on set V^*. Note that $\mathcal{A}_R$ has not queried $\text{Sign}^{(R)}_{sk_R[\cdot]}((m^*, \mathfrak{E}^*), acc_{V^*}, \cdot)$ (because $\mathcal{A}$ has not queried $\text{Sign}_{sk[\cdot]}(m^*, \mathfrak{E}^*, \cdot)$) nor $sk_R[i]$ for any $i \in V^*$ (because $\mathcal{A}$ did not query $sk[i]$ for any $i \in V^*$).

$\mathcal{A}_R$'s simulation for $\mathcal{A}$ is perfect and $\mathcal{A}_R$ wins its game if and only if $\mathcal{A}$ wins its game. It follows that

$$\Pr[\text{Exp}^{\text{sigforge}}_{\Pi,\mathcal{A}}(\lambda, n)] = \Pr[\text{Exp}^{\text{sigforge}'}_{\Pi_R,\mathcal{A}_R}(\lambda, n) = 1] \leq \mu(\lambda)$$

for some negligible function μ and for all $\lambda \in \mathbb{N}$ using Lemma 5.13. □

5.2.6 Unforgeability of epoch information

Finally, we prove unforgeability of epoch information, i.e. it should be infeasible for anyone but the group manager to produce epoch information $\mathfrak{E}$ that is accepted by EpochVerify or Verify. Note that this mostly follows from the security of the scheme Π_S used to create the signature θ as part of $\mathfrak{E}$ that is checked in both EpochVerify and Verify. However, note that for $V \neq V'$ with $acc_V = acc_{V'}$, the attacker against unforgeability of epoch information may query $\text{EpochCreate}_{gmsk}(V, \tau)$ and claim the resulting epoch information $\mathfrak{E}$ as an epoch forgery for V', τ. Of course, as observed before, accumulator security implies that it is infeasible to find such an accumulator collision.

In the proof, we first construct a forger against Π_S that is successful whenever the attacker $\mathcal{A}$ does not exploit the accumulator collision as described above. Then, we bound the probability for $\mathcal{A}$ to find an accumulator collision by reducing that event to the security of the underlying accumulator construction (Construction 3.5).

Lemma 5.15. *Construction 5.7 fulfills* unforgeability of epoch information *(Definition 5.6)*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time algorithm against unforgeability of epoch information in Π. We first construct an algorithm A_S against unforgeability of Π_S. Let $n \in \mathbb{N}$.

Forging a signature

$\mathcal{A}_S$ receives pk_S from its unforgeability experiment. It completes the setup by computing the rest of the keys itself: $(gpk_G, U, gmsk_G, (sk_G[i])_{i\in U}) \leftarrow \text{KeyGen}^{(G)}(1^\lambda, 1^n)$ and $(pk_R, (sk_R[i])_{i\in[n]}) \leftarrow \text{KeyGen}^{(R)}(1^\lambda, 1^n)$. $\mathcal{A}_S$ hands $U, gpk = (gpk_G, pk_R, pk_S)$, and $sk[i] = (sk_G[i], sk_R[i], gpk)$ for $i \in U$ to $\mathcal{A}$.

If $\mathcal{A}$ queries Open_{gmsk}, $\mathcal{A}_S$ answers using $gmsk_G$. If $\mathcal{A}$ queries $\text{EpochCreate}_{gmsk}(V, \tau)$, $\mathcal{A}_S$ computes $acc_V = \prod_{j\in V} g_{n+1-j}$ using pk_R and queries $\text{Sign}^{(S)}_{sk_S}((acc_V, \tau))$ to obtain a signature θ. It hands $\mathfrak{E} = (acc_V, \tau, \theta)$ to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs $\mathfrak{E}^* = (acc^*, \tau^*, \theta^*)$, V^*, m^*, and σ^*. If $\mathcal{A}$ has previously queried $\text{EpochCreate}_{gmsk}(V', \tau^*)$ for any $V' \subseteq U$ with $acc_{V'} = acc_{V^*}$, $\mathcal{A}_S$ outputs $\perp$ and aborts. Otherwise, $\mathcal{A}_S$ has never queried $\text{Sign}^{(S)}_{sk_S}((acc_{V^*}, \tau^*))$ and outputs θ^* as a forgery for message (acc^*, τ^*).

Let accforge be the event that $\mathcal{A}$ queried $\text{EpochCreate}_{gmsk}(V', \tau^*)$ for some $V' \subseteq U$ with $V' \neq V^*$ but $acc_{V'} = acc_{V^*}$. $\mathcal{A}_S$'s simulation for $\mathcal{A}$ is perfect and $\mathcal{A}_S$ wins its game if and only if $\mathcal{A}$ wins its game and accforge does not occur. It follows that

$$\Pr[\text{Exp}^{\text{epochforge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1 \wedge \neg\text{accforge}] = \Pr[\text{Exp}^{\text{sigforge}}_{\Pi_S,\mathcal{A}_S}(\lambda) = 1] \leq \mu_S(\lambda)$$

for some negligible μ_S using that Π_S is CPA-secure.

Accumulator collision

It remains to show that Pr[accforge] is negligible. We construct an attacker $\mathcal{A}_{acc}$ against the accumulator scheme's (Construction 3.5) security (Definition 3.2). Let $n^* := n(n+1)/2$ be the size of the accumulator universe $U^* = [n^*]$ as set up by $\text{KeyGen}^{(R)}(1^\lambda, 1^n)$. Note that $|U| = n$ is the maximum group size of our scheme, whereas n^* is the size that the accumulator is set up for (cf. Definition 4.5).

$\mathcal{A}_{acc}$ is given $pk_{acc} = ((\mathbb{G}, \mathbb{G}_\text{T}, e, p), (g^{\gamma^i})^{2n^*}_{i=0;i\neq n^*+1}, z)$. It completes the setup of Π_R the same way as the Type 1 forger in Lemma 4.14 (which fails with probability at most $n^*/|\mathbb{G}|$ – we call this event bad), yielding pk_R and $sk_R[i]$ for $i \in U$. $\mathcal{A}_{acc}$ then completes the setup of Π by computing $(pk_S, sk_S) \leftarrow \text{KeyGen}^{(S)}(1^\lambda)$ and $(gpk_G, U, gmsk_G, (sk_G[i])_{i\in U}) \leftarrow \text{KeyGen}^{(G)}(1^\lambda, 1^n)$. $\mathcal{A}_{acc}$ hands $U = [n], gpk = (gpk_G, pk_R, pk_S)$ and $sk[i] = (sk_G[i], sk_R[i], gpk)$ for $i \in U$ to $\mathcal{A}$.

If $\mathcal{A}$ queries Open_{gmsk} or EpochCreate$gmsk$, $\mathcal{A}_S$ answers using $gmsk_G$ or sk_S (and gpk).

Eventually, $\mathcal{A}$ outputs V^* and some other values. If accforge occurs (i.e. $\mathcal{A}$ queried $\text{EpochCreate}_{gmsk}(V', \tau^*)$ for some $V' \subseteq U$ with $V' \neq V^*$ but $acc_{V'} = acc_{V^*}$), then $\mathcal{A}_{acc}$ finds $i \in V' \backslash V^*$ (or $i \in V^* \backslash V'$) and outputs $(V^*, acc_{V'}, i, wit_{V',i})$ (or $(V', acc_{V'}, i, wit_{V^*,i})$) as an accumulator forgery (again like the Type 1 forger in Lemma 4.14).

This implies that

$$\begin{aligned}
&\Pr[\text{accforge}] \\
\leq &\Pr[\text{accforge} \wedge \neg\text{bad}] + \Pr[\text{bad}] \\
\leq &\Pr[\text{Exp}^{\text{witforge}}_{\mathcal{A}_{acc},\Pi_{acc}}(\lambda, n^*) = 1] + \Pr[\text{bad}] \\
\leq &\mu_{acc}(\lambda)
\end{aligned}$$

for some negligible μ_{acc}, using that $\Pr[\text{bad}]$ is negligible in λ and the accumulator is secure.

Conclusion

Putting everything together,

$$\begin{aligned}
&\Pr[\text{Exp}^{\text{epochforge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1] \\
\leq &\Pr[\text{Exp}^{\text{epochforge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1 \wedge \neg\text{accforge}] + \Pr[\text{accforge}] \\
\leq &\mu_S(\lambda) + \mu_{acc}(\lambda)
\end{aligned}$$

where μ_S and μ_{acc} are negligible. Consequently, $\Pr[\text{Exp}^{\text{epochforge}}_{\Pi,\mathcal{A}}(\lambda, n) = 1]$ is negligible in λ as required. □

Note that in the reduction to accumulator security, the accumulator was set up for $n^* = n(n+1)/2$ values as discussed at the end of Section 4.2.4 for the ring signature scheme. Since the accumulator is secure for any number of possible values, this is not a problem.

This concludes the security proofs for our group signature scheme with accumulator revocation.

5.2.7 Performance

We remark that while it may seem wasteful to create a scheme where signatures consist of two individual signatures, the ring signature portion actually contains only values that seem to be necessary to prove the user's revocation status. Indeed, neglecting the easy-to-fulfill role of Π_S for signing epoch information, one may view a group signature scheme with accumulator revocation as consisting of these two parts: some part that ensures that the group manager can *open* signatures (Π_G in the generic construction) and some part that manages revocation of signing rights (Π_R in our construction). Considering the standard methods to achieve these (adding an encrypted identity for Open to signatures, and using a signature of knowledge for accumulator inclusion, respectively), the two parts do not seem to have significant overlap.

This already gives one approach to create a potentially more efficient scheme: add an Open mechanism to our ring signature scheme Π_R. For example, this could be done by adding a linear encryption of g_i to signatures and extending the underlying protocol to prove that indeed g_i was encrypted [BBS04].

6 Anonymous credential systems

In this chapter, we explain and define anonymous credential systems. Then, we discuss revocation of credentials, and extend our definition to include revocation.

In anonymous credential systems [Lys02], there are two types of entities: users and organizations. Users are known to organizations only under a *pseudonym*. An organization may issue *credentials*, which certify a set of *attributes* to a user. If user U has a credential from organization O_1, he can choose to *show* it to another organization O_2, which knows U under a different pseudonym. When showing a credential, the user can choose exactly what information about his attributes is revealed to O_2. O_2 does not learn anything about the identity of U other than what U chooses to reveal about his attributes. This even holds if O_1 and O_2 cooperate: they know U under different pseudonyms which they cannot link.

Such systems implement the idea of *data minimization*: organizations should only learn the bare minimum information about the user needed to offer their service. A simple example application of such systems are government-issued ID cards. Card holders are routinely asked to show their ID card, for example to certify that they have reached a certain age. Right now, doing this means that the card holder unconditionally reveals all information on his card to the verifier, including, for example, name, address, exact birth date, etc., as they are physically printed onto the card. Using credential systems, the user can show his ID card credential from O_1 (the government) to some other organization O_2, revealing only, for example, the fact that his birth date attribute value passes a certain threshold. If the user subsequently reveals his address to some third organization O_3, then O_2 and O_3 cannot tell that they communicated with the same user. In particular, even if the two organizations collaborate, O_2 will not learn the user's address.

Another example can be found in the area of paid (flat rate) services. Right now, users registering for a paid service need to reveal their name, address, and credit card number to enable billing. In particular, the paid service knows about the user's identity and can link the user's behavior on their service to the user's data on cooperating services. With credential systems, paid services can be realized such that the user is not forced to enable organizations creating extensive profiles about him. Let P be an organization that handles payments. Our user U establishes a pseudonym with P and reveals all information necessary to handle billing (e.g., by showing their government ID card credential as before, which even guarantees P that the received data is correct). U pays some amount of money to P, specifying that it should be used to pay some service S. In return, P issues a credential to U. U then registers at the paid service S by establishing a pseudonym. He shows the credential issued by P to S, which activates his account (S remembers the payment status for the user's pseudonym). Through this

mechanism, S is guaranteed that only users who have paid can register for their service. S does not learn the user's billing information. P does not learn anything about U's behavior with S (other than that he paid them). And if U registers with some other (paid) service S', none of the organizations P, S, S' can link any of his pseudonyms to any other.

In conclusion, anonymous credential systems enable users to preserve their anonymity on a variety of services, and organizations are guaranteed that users, despite being anonymous, still fulfill some basic requirements for gaining access to a resource.

In the remainder of this chapter, we will give a formal experiment-based definition of anonymous credential systems (Section 6.1), and then define and discuss revocation in that context (Section 6.2).

6.1 Definition of anonymous credential systems

Anonymous credential systems are typically defined over an "ideal-world" model, where the anonymous credential system instantiates some ideal functionality [Lys02]. Proofs of correctness and security then boil down to showing that the credential system's behavior is computationally indistinguishable from the ideal functionality. In contrast, we will present an experiment-based definition of anonymous credential systems. One consideration here is that our more traditional way of defining security might be less prone to misunderstandings than its ideal-world counterpart. While we do not necessarily claim equivalence to the definition of [Lys02], our definition covers the desirable informal properties of such systems, as we will discuss later.

6.1.1 Syntax

We start with the syntax definition. In an anonymous credential system, we need operations and protocols to generate user and organization keys, to establish a pseudonym for a user at an organization, and to issue and show credentials. We start by formally defining the syntax of (anonymous) credential systems. Afterwards, we explain how the defined operations can be used in practice.

Definition 6.1. An *anonymous credential system* consists of the following polynomial-time algorithms:

- $\mathrm{KeyGen}(1^\lambda)$ is a probabilistic algorithm that generates public parameters pp and a description of the attribute universe A and the predicate universe $\Phi \subseteq 2^A$.
- $\mathrm{OInit}_{pp}(1^\lambda)$ is a probabilistic algorithm that generates a public key opk and a secret key osk.
- $\mathrm{UInit}_{pp}(1^\lambda)$ is a probabilistic algorithm that generates a user secret usk.

- UFormNym(usk, opk), OFormNym(osk) are two probabilistic interactive algorithms that both output a pseudonym N and UFormNym additionally outputs a corresponding secret r.

- UIssue(N, r, S), OIssue(osk, N, S) are two probabilistic interactive algorithms where in the end, UIssue outputs a credential *cred* for attributes $S \subseteq A$ or the failure symbol $\perp$.

- UShow($N', r', N, r, cred, \phi$), OShow(osk, N, opk', ϕ) are two probabilistic interactive algorithms that both output either 0 or 1 for a formula $\phi \in \Phi$.

We say that an anonymous credential system is *correct* if for all $\lambda \in \mathbb{N}$, all $(pp, A, \Phi) \in [\text{KeyGen}(1^\lambda)]$, $usk \in [\text{UInit}_{pp}(1^\lambda)]$, $(opk, osk), (opk', osk') \in [\text{OInit}_{pp}(1^\lambda)]$, and all $S \subseteq A$ and $\phi \in \Phi$ with $S \in \phi$

$$\begin{aligned}
&\Pr[(\cdot, (N, r), N) \leftarrow (\text{UFormNym}(usk, opk) \overset{o}{\leftrightarrow} \text{OFormNym}(osk)),\\
&\quad (\cdot, (N', r'), N') \leftarrow (\text{UFormNym}(usk, opk') \overset{o}{\leftrightarrow} \text{OFormNym}(osk')),\\
&\quad (\cdot, cred, \cdot) \leftarrow (\text{UIssue}(N', r', S) \overset{o}{\leftrightarrow} \text{OIssue}(osk', N', S)),\\
&\quad (\cdot, b_{\text{UShow}}, b_{\text{OShow}}) \leftarrow (\text{UShow}(N', r', N, r, cred, \phi) \overset{o}{\leftrightarrow} \text{OShow}(osk, N, opk', \phi)) :\\
&\quad o_{\text{UShow}} = o_{\text{OShow}} = 1] = 1
\end{aligned}$$

In such a scheme, UFormNym,UIssue,UShow are executed by users, and their respective counterparts these algorithms interact with, namely OFormNym,OIssue,OShow, are run by organizations. Consider the following example run. First, the system is set up by running KeyGen(1^λ). Two organizations generate their key pairs (opk, osk) and (opk', osk') using OInit. A user generates his secret key usk using UInit. The user registers at the organization with public key opk' by establishing a pseudonym N' using UFormNym(usk, opk') (while OFormNym(osk') is run on the organization's end). The user also remembers a secret r' for his pseudonym. The organization can issue the user (which it knows as N') a credential *cred* for some set S of attributes by running OIssue(osk', N', S) (while UIssue is run on the user's end). Then the user may establish another pseudonym N at another organization and then run UShow($N', r', N, r, cred, \phi$) (while OShow(osk, N, opk', ϕ) is run on the organization's end) to prove possession of a credential *cred*, whose attributes S fulfill ϕ (i.e. $S \in \phi$), issued by the organization with public key opk' (and that N, N' belong to the same user secret usk). Correctness of the scheme exactly says that in this scenario (provided $S \in \phi$), UShow and OShow both output 1 with probability 1.

For illustration, note that such a system can be constructed as follows [Lys02]. A pseudonym N is a commitment on a user secret usk (and r its open value). A credential *cred* is an organization's signature on usk (signed knowing only the commitment N to usk) and on S. The protocol for showing a credential is a zero-knowledge proof of

knowledge of a hidden signature on hidden values usk, S such that $S \in \phi$, and that both pseudonyms N, N' and the signature contain the same user secret usk.

In the following, we define security for anonymous credential systems.

6.1.2 Anonymity

We now start defining security of anonymous credential systems. First, we formalize that honest users should stay anonymous against collaborating organizations. It should hold that

- Establishing a pseudonym does not leak any information about the user's identity.
- Two different pseudonyms of the same user should not be linkable to one another.
- Showing a credential should only reveal exactly what the user chooses to, i.e. only the user's pseudonym with the verifier organization, the identity of the issuer organization, and that its attributes fulfill the formula ϕ.

We will first give a rough overview of the experiment, then give the formal definition, and then discuss details.

In the experiment, we set up two honest users. The adversary $\mathcal{A}$ can set up any number of corrupted users and corrupted organizations itself by running UInit and OInit, respectively. $\mathcal{A}$ then sets up the situation it wants to attack by commanding the two honest users to run some sequence of UFormNym, UIssue, UShow with $\mathcal{A}$'s organizations. Eventually, $\mathcal{A}$ points out two credentials that the honest users have received. The experiment chooses one of the two credentials, which $\mathcal{A}$ can again use in its requests for UFormNym, UIssue, UShow in order to try to distinguish the chosen credential from the other one. Intuitively, it should be infeasible for $\mathcal{A}$ to distinguish which of the two credentials was chosen unless it explicitly asks to be shown the credential for an attribute formula ϕ that is fulfilled by only exactly one of the two credentials. Special care needs to be taken that $\mathcal{A}$ is not trivially given distinguishing information about the credential. For example, if the two credentials belong to two different users, showing the credential to an organization using a pseudonym of the credential holder will succeed, doing the same with a pseudonym of the other user should fail, since credentials are non-transferable.

Definition 6.2. A anonymous credential system Π has *anonymity* if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{anon}}_{\Pi,\mathcal{A}}(\lambda) := |\Pr[\mathrm{Exp}^{\mathrm{anon}-1}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathrm{Exp}^{\mathrm{anon}-0}_{\Pi,\mathcal{A}}(\lambda) = 1]| \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiments $\mathrm{Exp}^{\mathrm{anon}-b}_{\Pi,\mathcal{A}}(\lambda)$ for $b \in \{0,1\}$ work as follow:

1. $(pp, A, \Phi) \leftarrow \mathrm{KeyGen}(1^\lambda)$, $usk_0, usk_1 \xleftarrow{R} [\mathrm{UInit}_{pp}(1^\lambda)]$.
2. $\mathcal{A}$ is given pp, A, Φ.

3. $\mathcal{A}$ can request to interact[1] with UFormNym(usk_j, opk) by specifying $j \in \{0,1\}$ and opk. Let (N, r) be the output of UFormNym. The experiment stores $(j_\ell, opk_\ell, N_\ell, r_\ell) := (j, opk, N, r)$ on an ℓ-indexed list $\mathcal{N}$.

4. $\mathcal{A}$ can request to interact with UIssue(N_ℓ, r_ℓ, S) by specifying an index ℓ on the list of pseudonyms $\mathcal{N}$. It may choose $S \subseteq A$ freely. Let $cred$ be the output of UIssue. The experiment stores $(cred_{\ell,k}, S_{\ell,k}) := (cred, S)$ on a k-indexed list $\mathcal{C}_\ell$.

5. $\mathcal{A}$ can request to interact with UShow($N_{\ell'}, r_{\ell'}, N_\ell, r_\ell, cred_{\ell',k}, \phi$) by specifying $\phi \in \Phi$, two indices ℓ, ℓ' on $\mathcal{N}$, and an index k on $\mathcal{C}_{\ell'}$.

6. Eventually $\mathcal{A}$ outputs two indices ℓ_0, ℓ_1 on $\mathcal{N}$ and two indices k_0, k_1 on $\mathcal{C}_{\ell_0}, \mathcal{C}_{\ell_1}$, respectively. The experiment sets $cred^* := cred_{\ell_b,k_b}$, $S^* := S_{\ell_b,k_b}$, $usk_\star := usk_b$, $opk^* := opk_{\ell_b}$, and $(N^*, r^*) := (N_{\ell_b}, r_{\ell_b})$.

7. If $\perp \in \{cred_{\ell_0,k_0}, cred_{\ell_1,k_1}\}$ (i.e. one of the credentials is invalid) or $opk_{\ell_0} \neq opk_{\ell_1}$ (i.e. the credentials were created by different organizations), the experiment outputs 0 and aborts.

8. If $j_{\ell_0} = j_{\ell_1}$, let $\square = j_{\ell_0}$, otherwise, let $\square = \star$. The experiment adds $(j_\ell, opk_\ell, N_\ell, r_\ell) := (\square, opk^*, N^*, r^*)$ to $\mathcal{N}$ for a fresh index $\ell =: \ell^*$, and it adds $(cred_{\ell,1}, S_{\ell,1}) := (cred^*, S^*)$ to $\mathcal{C}_\ell$.

9. $\mathcal{A}$ may continue to request interactions with UFormNym, UIssue, and UShow as before.

10. In addition, $\mathcal{A}$ may query UFormNym with $j = \star$.

11. In the end, $\mathcal{A}$ outputs a bit b'.

12. If $\mathcal{A}$ has requested an interaction with UShow for ϕ, ℓ, ℓ', k where $j_{\ell'} \neq j_\ell$ and $\star \in \{j_\ell, j_{\ell'}\}$, then the experiment outputs 0.

13. Otherwise, if $\mathcal{A}$ has requested an interaction with UShow for ϕ, ℓ, ℓ', k where $\ell' = \ell^*$ and $k = 1$ and either $S_{\ell_0,k_0} \in \phi$ and $S_{\ell_1,k_1} \notin \phi$, or $S_{\ell_0,k_0} \notin \phi$ and $S_{\ell_1,k_1} \in \phi$, then the experiment outputs 0.

14. Otherwise it outputs b'.

Note that in the first phase (before $\mathcal{A}$ outputs its challenge credential indices), $\mathcal{A}$ has full information about the relation between users and pseudonyms, and what credentials and attributes were issued to which pseudonyms. It does not have full information about

[1] Interaction with an algorithm here means that the algorithm sends, receives, and processes protocol messages to and from $\mathcal{A}$. In particular, $\mathcal{A}$ cannot rewind the algorithm to any previous state. Furthermore, while an interaction is in progress, $\mathcal{A}$ cannot start another one (a similar restriction is made in [Lys02]).

$cred^*$ (it only knows that it is one of two possible credentials). Afterwards, $\mathcal{A}$ may again fully control the system but should not be able to link $cred^*$ to what it knows, except through trivial means.

The corresponding real-world scenario is as follows: suppose collaborating organizations gathered all possible information about two honest users, their pseudonyms and credentials (e.g., by inferring from request timing). Suppose the two users each have a credential from organization O. At some point, the organizations lose track and cannot distinguish between two users through side channels anymore. Then the user needs to explicitly choose to give the organizations distinguishing information, for example by showing his credential for a formula ϕ specific enough to distinguish his attributes from the other user's.

Note that for its challenge credentials, $\mathcal{A}$ may adaptively choose any two credentials that it has previously issued to the honest users. The two credentials that $\mathcal{A}$ points out can be either from the same user or from different users. The first case (same user) covers that organizations cannot learn anything about a credential's attributes other than what is revealed by the user through ϕ. The second case (different users) covers that $\mathrm{UFormNym}, \mathrm{UShow}, \mathrm{UIssue}$ do not leak any information about the user's identity and that pseudonyms are unlinkable.

After $\mathcal{A}$ points out the two credentials, the experiment stores the challenge credential's pseudonym with the issuer organization as a new entry $(j_{\ell^*}, opk_{\ell^*}, N_{\ell^*}, r_{\ell^*})$ on $\mathcal{N}$ at index ℓ^* (we will explain the role of $j_{\ell^*} = \star$ later). Hence, $\mathcal{A}$ can afterwards interact with this pseudonym using the same mechanisms as in the first phase. Furthermore, it stores the credential $cred^*$ on $\mathcal{C}_{\ell^*}$.

Note the following ways that $\mathcal{A}$ can trivially distinguish the two credentials $cred_{\ell_0,k_0}$ and $cred_{\ell_1,k_1}$.

- If exactly one of the credentials is invalid (e.g., because $\mathcal{A}$ refused to do a correct run of OIssue when issuing it), $\mathcal{A}$ can trivially distinguish the invalid credential from the valid one in the show protocol.

- If the credentials were issued by different organizations $opk_{\ell_0} \neq opk_{\ell_1}$, running the show protocol will succeed for the correct of the two organization keys. It should fail for the other, which allows $\mathcal{A}$ to distinguish the two credentials.

- If exactly one of the credentials' attributes fulfills some formula ϕ, running the show protocol with ϕ distinguishes the credentials intentionally.

- If the credentials belong to two different users $j_{\ell_0} \neq j_{\ell_1}$, then $\mathcal{A}$ can distinguish the credentials by running the show protocol with a pseudonym for which it knows the corresponding user. By design, the protocol will succeed if the pseudonym belongs to the user the credential was issued to. It should fail if the pseudonym belongs to the other user.

Note that if $\mathcal{A}$ attempts to exploit one of the first three trivial distinguishers, the experiment will abort and output 0 in Step 7 or Step 13.

To prevent last one, we use the following mechanism: if $j_{\ell_0} \neq j_{\ell_1}$, the experiment sets the user index j_{ℓ^*} for pseudonym N^* to the special symbol $\star$ in Step 8. Intuitively, this represents the fact that $\mathcal{A}$ does not know which of the two honest users the pseudonym belongs to (as it depends on b). To prevent $\mathcal{A}$ from using the trivial distinguishing mechanism described above, the experiment aborts in Step 12 if $\mathcal{A}$ attempts to run the show protocol for a pseudonym index $\ell' = \star$ (the credential owner) and pseudonym index $\ell \neq \star$ (the user's pseudonym at the verifier organization), or vice versa. If $\mathcal{A}$ were to run this, it could immediately judge that the user behind pseudonym ℓ' is the same as the one behind ℓ. Since $\mathcal{A}$ knows the user behind ℓ if $\ell \neq \star$, this would allow $\mathcal{A}$ to trivially distinguish the two credentials. Note that $\mathcal{A}$ may show the credential to other pseudonyms (that it does not know) by querying UFormNym with $j = \star$ (Step 10) beforehand, which allows it to create new pseudonyms (and other credentials) for the unknown user (which are also marked with $j_\ell = \star$).

Note that all of these restrictions are reasonable. Users need to be aware that when showing their credential, they reveal the credential issuer, their pseudonym with the verifier, and information about the attributes according to ϕ.

In the construction mentioned above, the *hiding* property of the commitment scheme implies that $\mathcal{A}$ cannot gain any information about the user from his pseudonym N. Furthermore, the show protocol being *zero-knowledge* corresponds to $\mathcal{A}$ not gaining any information from its UShow queries on the challenge credential. Finally, in the issue protocol, the issuer is just given a commitment to create a signature, so the organization does not learn anything else about the user through this (again because of the commitment scheme's *hiding* property).

6.1.3 Soundness

We now define *soundness*, which guarantees that dishonest users cannot claim possession of a credential or attributes that they did not receive.

More specifically, organizations expect that if $\mathrm{OShow}(osk, N, opk', \phi)$ outputs 1, then the user should indeed have been issued a credential from the organization with public key opk', and with attributes S fulfilling ϕ. Furthermore, the user behind the pseudonym N should be the same as the user who was issued the credential.

The latter property guarantees that users do not simply hand their credentials to other users. This still leaves the option that users hand their credential *and* their secret key(s) to another person. As noted in [Lys02], this should be discouraged through other means, e.g., by embedding a valuable secret in the user's key.

In the experiment, we set up two honest organizations and an arbitrary number of honest users. Again, $\mathcal{A}$ can set up any number of corrupted users (and organizations) itself. $\mathcal{A}$ controls the system, allowing it to issue commands to the honest organizations and honest users to interact with its corrupted entities. Furthermore, $\mathcal{A}$ may request transcripts of interactions between any honest user and any honest organization. Eventually, $\mathcal{A}$ outputs a challenge formula ϕ^* and a challenge pseudonym N^*. Then $\mathcal{A}$ participates in the showing protocol with the first honest organization under pseudonym N^*, trying

to convince it that it has a valid credential from the second honest organization. If $\mathcal{A}$ never requested such a credential but the show protocol accepts, $\mathcal{A}$ wins the game.

Definition 6.3 (Soundness)**.** An anonymous credential system Π is *sound* if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{soundness}}_{\Pi,\mathcal{A}}(\lambda) := \Pr[\mathrm{Exp}^{\mathrm{soundness}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{soundness}}_{\Pi,\mathcal{A}}(\lambda)$ is defined as follows:

1. The experiment sets up the system $(pp, A, \Phi) \leftarrow \mathrm{KeyGen}(1^\lambda)$ and two honest organizations $(opk_0^*, osk_0^*), (opk_1^*, osk_1^*) \leftarrow \mathrm{OInit}_{pp}(1^\lambda)$. The experiment maintains a list $\mathcal{U}$ of honest user keys usk_j. Whenever an index j is accessed for the first time, the experiment runs $usk_j \leftarrow \mathrm{UInit}_{pp}(1^\lambda)$.

2. $\mathcal{A}$ is given pp and opk_0^*, opk_1^*.

3. For $t \in \{0,1\}$, $\mathcal{A}$ may request interaction with $\mathrm{OFormNym}(osk_t^*)$, $\mathrm{OIssue}(osk_t^*, \cdot, \cdot)$, and $\mathrm{OShow}(osk_t^*, \cdot, \cdot, \cdot)$.

4. $\mathcal{A}$ may request interaction with $\mathrm{UFormNym}(usk_j, opk)$ by specifying opk and a user index j. Let (N, r) be the output of UFormNym. The experiment stores $(j_\ell, opk_\ell, N_\ell, r_\ell) := (j, opk, N, r)$ on an ℓ-indexed list $\mathcal{N}$.

5. $\mathcal{A}$ may request interaction with $\mathrm{UIssue}(N_\ell, r_\ell, S)$ by specifying an index ℓ on $\mathcal{N}$ and any $S \subseteq A$. Let *cred* be the output of UIssue. The experiment stores $cred_{\ell,k} :=$ *cred* on a k-indexed list $\mathcal{C}_\ell$. $\mathcal{A}$ is given *cred*.

6. $\mathcal{A}$ may request interaction with $\mathrm{UShow}(N_{\ell'}, r_{\ell'}, N_\ell, r_\ell, cred_{\ell',k}, \phi)$ by specifying $\phi \in \Phi$, two indices ℓ, ℓ' on $\mathcal{N}$, and an index k on $\mathcal{C}_{\ell'}$.

7. $\mathcal{A}$ may request a transcript for $\mathrm{UFormNym}(usk_j, opk_t^*) \leftrightarrow \mathrm{OFormNym}(osk_t^*)$ by specifying a user index j and $t \in \{0,1\}$. Let (N, r) be the output of UFormNym. The experiment stores $(j_\ell, opk_\ell, N_\ell, r_\ell) := (j, opk_t^*, N, r)$ on a $\mathcal{N}$.

8. $\mathcal{A}$ may request a transcript for $\mathrm{UIssue}(N_\ell, r_\ell, S) \leftrightarrow \mathrm{OIssue}(osk_t^*, N_\ell, S)$ by specifying an index ℓ on $\mathcal{N}$, $t \in \{0,1\}$, and $S \subseteq A$. Let *cred* be the output of UIssue. The experiment stores $cred_{\ell,k} :=$ *cred* on $\mathcal{C}_\ell$. $\mathcal{A}$ is given *cred*.

9. $\mathcal{A}$ may request a transcript for $\mathrm{UShow}(N_{\ell'}, r_{\ell'}, N_\ell, r_\ell, cred_{\ell',k}, \phi) \leftrightarrow \mathrm{OShow}(osk_t^*, N_\ell, opk_{t'}^*, \phi)$ by specifying $t, t' \in \{0,1\}$, ϕ and two indices ℓ, ℓ' on $\mathcal{N}$ and an index k on $\mathcal{C}_{\ell'}$.

10. Eventually, $\mathcal{A}$ outputs a formula $\phi^* \in \Phi$ and a pseudonym N^*.

11. The experiment runs $z \leftarrow \mathrm{OShow}(osk_0^*, N^*, opk_1^*, \phi^*)$ to interact with $\mathcal{A}$.

12. If $z = 1$ and $\mathcal{A}$ never requested interaction with OIssue$(osk_1^*, \cdot, S)$ in Step 3 where $S \in \phi^*$, the experiment outputs 1.

13. Otherwise it outputs 0.

In the corresponding real-world scenario, a group of dishonest users tries to convince an honest organization that they were issued a credential (with certain attributes) by another honest organization which none of them was issued directly.

Note that after any honest run of UIssue, $\mathcal{A}$ is given the resulting credential *cred*. This corresponds to the requirement that credentials are tied to their owners (and hence, should be useless without revealing the corresponding pseudonym secret r). Revealing r should be discouraged, for example, by embedding a valuable user secret in r [Lys02] (in case of the construction mentioned above, r allows computing the user secret, which in turn should be made undesirable for users).

Among others, our definition covers

- It is infeasible to make OShow accept using a credential with attributes $S \notin \phi$.
- A user cannot combine attributes from multiple credentials to fulfill a more specific formula ϕ than either of the original credentials.
- The transcripts of honest parties do not reveal any information that could be used to impersonate an honest user or to forge a credential.
- Credentials are bound to a specific user and cannot be shown without the user's secrets.

In the construction mentioned above, *unforgeability* of the signature scheme implies that users cannot create credentials that they did not request. The show protocol being a *proof of knowledge* implies that it is essentially necessary to have such a credential in order to reliably pass the protocol. Furthermore, the show protocol needs to ensure that the signed user identity is the same as in the user's pseudonym, which corresponds to (among others) the *binding* property of the commitment scheme.

6.2 Anonymous credential systems with revocation

In this section, we will explore revocation for anonymous credential systems. We identify three general types of revocation: (1) revocation of users, (2) revocation of pseudonyms, and (3) revocation of credentials.

Revoking users is the most broad application of revocation. It requires a central authority that maintains and dictates the revocation status of users. In order to make meaningful revocation decisions, the central authority would have to be able to uncover identities from pseudonyms.

Revoking pseudonyms can be done by each organization individually and without any special capabilities. Revoking a pseudonym should invalidate all credentials issued to that pseudonym.

However, in many cases, organizations just need to revoke a single credential, for example, because the user's attributes changed. In these cases, it would be disproportionate to revoke the user's pseudonym or even the whole user across all organizations. For this reason, we will focus on revocation of credentials, which can also be used to realize the semantics of pseudonym revocation. Note that revocation of users may still have its applications (possibly in addition to revocation of credentials) because it allows revoking a user across multiple services, which cannot be easily replicated through credential revocation without a dedicated entity that can link pseudonyms.

We model revocation of credentials as follows. An organization embeds a (unique) identifier i into every issued credential. Periodically or on change, it broadcasts new epoch information $\mathfrak{E}$ that contains information about the set of valid identifiers. Before showing a credential *cred*, the user computes a witness *wit* for the credential identifier being valid (this needs to be done only once per epoch $\mathfrak{E}$). Then the show protocol additionally proves that the credential identifier is unrevoked in the current epoch.

One can instantiate this idea with accumulators: $\mathfrak{E}$ is an accumulator value acc_V of the set V of valid identifiers i. The witness *wit* is simply the accumulator witness, and the proof protocol ensures knowledge of *wit* that fulfills the accumulator verification equation. Alternatively, $\mathfrak{E}$ could be a list of valid/invalid identifiers, *wit* is unused (no updates required), and one proves inclusion/exclusion on $\mathfrak{E}$ (cf. revocation lists like for group signatures [BS04]). However, note that our anonymity definition will require that users of revoked credentials retain their anonymity (since revocation of credentials is a normal process that does not imply user misbehavior). Hence the revocation list must not reveal any information that can be used to identify previous credential show transcripts.

In the following, we give definitions of anonymous credential systems with revocation (syntax, anonymity, and soundness).

6.2.1 Syntax

Notable changes to the corresponding definitions without revocation (Section 6.1) are underlined.

Definition 6.4. An *anonymous credential system* with revocation consists of the following polynomial-time algorithms:

- $\text{KeyGen}(1^\lambda)$ is a probabilistic algorithm that generates public parameters pp and a description of the attribute universe A and the predicate universe $\Phi \subseteq 2^A$.
- $\text{OInit}_{pp}(1^\lambda, 1^n)$ is a probabilistic algorithm that generates a set U ($|U| = n$) of credential identifiers, a public key opk and a secret key osk.

- $\underline{\text{EpochCreate}(osk, V)}$ is a probabilistic algorithm that outputs epoch information $\mathfrak{E}$.
- $\underline{\text{EpochVerify}(opk, V, \mathfrak{E})}$ is a deterministic algorithm that returns 0 or 1.
- $\underline{\text{WitCreate}(opk, V, i)}$ is a deterministic algorithm that creates a witness $wit = wit_{V,i}$ for an identifier $i \in V$.
- $\text{UInit}_{pp}(1^\lambda)$ is a probabilistic algorithm that generates a user secret usk.
- $\text{UFormNym}(usk, opk), \text{OFormNym}(osk)$ are two probabilistic interactive algorithms that both output a pseudonym N and UFormNym additionally outputs a corresponding secret r.
- $\text{UIssue}(N, r, \underline{i}, S), \text{OIssue}(osk, N, \underline{i}, S)$ are two probabilistic interactive algorithms where in the end, UIssue outputs a credential $cred$ for identifier i and attributes $S \subseteq A$, or the failure symbol $\perp$.
- $\text{UShow}(N', r', N, r, cred, \phi, \underline{\mathfrak{E}}, \underline{wit}), \text{OShow}(osk, N, opk', \phi, \underline{\mathfrak{E}})$ are two probabilistic interactive algorithms that both output either 0 or 1 for a formula $\phi \in \Phi$.

We say that an anonymous credential system is *correct* if for all $\lambda, n \in \mathbb{N}$, all $(pp, A, \Phi) \in [\text{KeyGen}(1^\lambda)]$, $usk \in [\text{UInit}_{pp}(1^\lambda)]$, $(U, opk, osk), (U', opk', osk') \in [\text{OInit}_{pp}(1^\lambda, 1^n)]$, all $\underline{V \subseteq U}$, $\underline{i \in V}$, $\underline{\mathfrak{E} \in \text{EpochCreate} \in [\text{EpochCreate}(osk', V)]}$, and all $S \subseteq A$ and $\phi \in \Phi$ with $S \in \phi$

$$
\begin{aligned}
&\Pr[(\cdot, (N, r), N) \leftarrow (\text{UFormNym}(usk, opk) \overset{o}{\leftrightarrow} \text{OFormNym}(osk)),\\
&\quad (\cdot, (N', r'), N') \leftarrow (\text{UFormNym}(usk, opk') \overset{o}{\leftrightarrow} \text{OFormNym}(osk')),\\
&\quad (\cdot, cred, \cdot) \leftarrow (\text{UIssue}(N', r', \underline{i}, S) \overset{o}{\leftrightarrow} \text{OIssue}(osk', N', \underline{i}, S)),\\
&\quad (\cdot, b_{\text{UShow}}, b_{\text{OShow}}) \leftarrow (\text{UShow}(N', r', N, r, cred, \phi, \underline{\mathfrak{E}}, \underline{\text{WitCreate}(opk', V, i)})\\
&\qquad\qquad \overset{o}{\leftrightarrow} \text{OShow}(osk, N, opk', \phi, \underline{\mathfrak{E}})) :\\
&\quad o_{\text{UShow}} = o_{\text{OShow}} = 1] = 1
\end{aligned}
$$

and

$$\text{EpochVerify}(opk', V, \mathfrak{E}) = 1$$

Note that we added three new operations: EpochCreate for an organization to create new epochs $\mathfrak{E}$, WitCreate for users to compute their credential identifiers' witnesses, and EpochVerify for users to check that $\mathfrak{E}$ indeed corresponds to some claimed set V.

Note that revocation opens up new possibilities for organizations to circumvent anonymity. For example, using an epoch for $V = \{i\}$ allows the verifier organization to test whether the credential shown in a protocol has identifier i (which allows the issuer and verifier

organization to link the user's pseudonyms). When the user is given the set V to create his witness, he may judge whether or not he wants to let an organization narrow down his credential identifier to V by running the show protocol.

The setup for organizations has an additional parameter 1^n, corresponding to an upper bound on the number of credential identifiers (for example to set up the underlying accumulator construction).

6.2.2 Anonymity

As indicated earlier, credential revocation generally restricts user anonymity. Organizations now gain information on whether the identifier of the shown credential is one of the valid values of the current epoch. Hence, we need to adapt the anonymity experiment. We add an additional restriction that the distinguisher $\mathcal{A}$ may not trivially distinguish the two credentials by simply creating an epoch where only exactly one of the credential identifiers is revoked.

Definition 6.5. A anonymous credential system with revocation Π has *anonymity* if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{anon}}_{\Pi,\mathcal{A}}(\lambda) := |\Pr[\mathrm{Exp}^{\mathrm{anon}-1}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathrm{Exp}^{\mathrm{anon}-0}_{\Pi,\mathcal{A}}(\lambda) = 1]| \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiments $\mathrm{Exp}^{\mathrm{anon}-b}_{\Pi,\mathcal{A}}(\lambda)$ for $b \in \{0,1\}$ work as follow:

- $(pp, A, \Phi) \leftarrow \mathrm{KeyGen}(1^\lambda)$, $usk_0, usk_1 \xleftarrow{R} [\mathrm{UInit}_{pp}(1^\lambda)]$.
- $\mathcal{A}$ is given pp, A, Φ.
- $\mathcal{A}$ can request to interact with $\mathrm{UFormNym}(usk_j, opk)$ by specifying $j \in \{0,1\}$ and opk. Let (N, r) be the output of UFormNym. The experiment stores $(j_\ell, opk_\ell, N_\ell, r_\ell) := (j, opk, N, r)$ on an ℓ-indexed list $\mathcal{N}$.
- $\mathcal{A}$ can request to interact with $\mathrm{UIssue}(N_\ell, r_\ell, i, S)$ by specifying an index ℓ on the list of pseudonyms $\mathcal{N}$. It may choose $S \subseteq A$ and $\underline{i}$ freely. Let $cred$ be the output of UIssue. The experiment stores $(cred_{\ell,k}, S_{\ell,k}, \underline{i_{\ell,k}}) := (cred, S, \underline{i})$ on a k-indexed list $\mathcal{C}_\ell$.
- $\mathcal{A}$ can request to interact with $\mathrm{UShow}(N_{\ell'}, r_{\ell'}, N_\ell, r_\ell, cred_{\ell',k}, \phi, \mathfrak{E}, \mathrm{WitCreate}(opk_{\ell'}, V, i_{\ell',k}))$ by specifying $\phi \in \Phi$, $\underline{V, \mathfrak{E}}$, two indices ℓ, ℓ' on $\mathcal{N}$, and an index k on $\mathcal{C}_{\ell'}$.
- Eventually $\mathcal{A}$ outputs two indices ℓ_0, ℓ_1 on $\mathcal{N}$ and two indices k_0, k_1 on $\mathcal{C}_{\ell_0}, \mathcal{C}_{\ell_1}$, respectively. The experiment sets $cred^* := cred_{\ell_b,k_b}$, $S^* := S_{\ell_b,k_b}$, $\underline{i^* := i_{\ell,k}}$ and $usk_\star := usk_b$, $(N^*, r^*) := (N_{\ell_b}, r_{\ell_b})$, $opk^* := opk_{\ell_b}$.
- If $\perp \in \{cred_{\ell_0,k_0}, cred_{\ell_1,k_1}\}$ (i.e. one of the credentials is invalid) or $opk_{\ell_0} \neq opk_{\ell_1}$ (i.e. the credentials were created by different organizations), the experiment outputs 0 and aborts.

- If $j_{\ell_0} = j_{\ell_1}$, let $\square = j_{\ell_0}$, otherwise, let $\square = \star$. The experiment adds $(j_\ell, opk_\ell, N_\ell, r_\ell) := (\square, opk^*, N^*, r^*)$ to $\mathcal{N}$ for a fresh index $\ell =: \ell^*$, and $(cred_{\ell,1}, S_{\ell,1}, \underline{i_{\ell,1}}) := (cred^*, S^*, \underline{i^*})$ to $\mathcal{C}_\ell$.
- $\mathcal{A}$ may continue to request interactions with UFormNym, UIssue, and UShow as before.
- In addition, $\mathcal{A}$ may query UFormNym with $j = \star$.
- In the end, $\mathcal{A}$ outputs a bit b'.
- If $\mathcal{A}$ has requested an interaction with UShow for $\phi, \ell, \ell', k, V, \mathfrak{E}$ where $j_{\ell'} \neq j_\ell$ and $\star \in \{j_\ell, j_{\ell'}\}$, then the experiment outputs 0.
- Otherwise, if $\mathcal{A}$ has requested an interaction with UShow for $\phi, \ell, \ell', k, V, \mathfrak{E}$ where $\ell' = \ell^*$ and $k = 1$ and either $S_{\ell_0,k_0} \in \phi$ and $S_{\ell_1,k_1} \notin \phi$, or $S_{\ell_0,k_0} \notin \phi$ and $S_{\ell_1,k_1} \in \phi$, then the experiment outputs 0.
- Otherwise, if $\mathcal{A}$ has requested an interaction with UShow for $\phi, \ell, \ell', k, V, \mathfrak{E}$ where $\ell' = \ell^*$ and $k = 1$ and either $\underline{i_{\ell_0,k_0} \in V}$ and $\underline{i_{\ell_1,k_1} \notin V}$, or $\underline{i_{\ell_0,k_0} \notin V}$ and $\underline{i_{\ell_1,k_1} \in V}$, then the experiment outputs 0.
- Otherwise it outputs b'.

Since $\mathcal{A}$ controls all organizations, it can create any necessary epoch information itself.

Note that when $\mathcal{A}$ interacts with UShow for the challenge credential, it needs to specify the set V in addition to epoch information $\mathfrak{E}$. This is because V is needed to compute the witness for the challenge credential's identifier i^* and in order for the experiment to judge whether or not i^* is included in the epoch. The indirect link between V and $\mathfrak{E}$ is through WitCreate – if $\mathfrak{E}$ and V do not fit together, WitCreate should not be able to compute a valid accumulator witness, so the protocol should fail regardless of revocation status.

Note that according to this definitions, revoked users retain their anonymity, i.e. after a user is revoked, their old transcripts still cannot be traced to them. This is a necessary condition that arises from the fact that organizations, which are potentially interested in disabling anonymity, are the ones dictating revocation.

Overall, the changes to the anonymity experiment are unsurprising. The focus is on the additional restriction that $\mathcal{A}$ may not request being shown the challenge credential for epoch information that trivially distinguishes the two credentials.

6.2.3 Soundness

We now consider the soundness definition (i.e. users should not be able to show a credential they were not issued). Here, users gain a new way to break the scheme,

namely by circumventing the revocation mechanism and successfully showing a revoked credential.

Definition 6.6 (Soundness)**.** An anonymous credential system with revocation Π is *sound* if for all $n \in \mathbb{N}$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\mathrm{Adv}^{\mathrm{soundness}}_{\Pi,\mathcal{A}}(\lambda, n) := \Pr[\mathrm{Exp}^{\mathrm{soundness}}_{\Pi,\mathcal{A}}(\lambda, n) = 1] \leq \mu(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the experiment $\mathrm{Exp}^{\mathrm{soundness}}_{\Pi,\mathcal{A}}(\lambda, n)$ is defined as follows:

1. The experiment sets up the system $(pp, A, \Phi) \leftarrow \mathrm{KeyGen}(1^\lambda)$ and two honest organizations $(U_0^*, opk_0^*, osk_0^*), (U_1^*, opk_1^*, osk_1^*) \leftarrow \mathrm{OInit}_{pp}(1^\lambda, \underline{1^n})$. The experiment maintains a list $\mathcal{U}$ of honest user keys usk_j. Whenever an index j is accessed for the first time, the experiment runs $usk_j \leftarrow \mathrm{UInit}_{pp}(1^\lambda)$.

2. $\mathcal{A}$ is given pp and $U_0, opk_0^*, U_1, opk_1^*$ and oracle access to $\underline{\mathrm{EpochCreate}(osk_0^*, \cdot)}$ and $\underline{\mathrm{EpochCreate}(osk_1^*, \cdot)}$.

3. For $t \in \{0, 1\}$, $\mathcal{A}$ may request interaction with $\mathrm{OFormNym}(osk_t^*)$, $\mathrm{OIssue}(osk_t^*, \cdot, \cdot, \cdot)$, and $\mathrm{OShow}(osk_t^*, \cdot, \cdot, \cdot, \cdot)$.

4. $\mathcal{A}$ may request interaction with $\mathrm{UFormNym}(usk_j, opk)$ by specifying opk and a user index j. Let (N, r) be the output of UFormNym. The experiment stores $(j_\ell, opk_\ell, N_\ell, r_\ell) := (j, opk, N, r)$ on an ℓ-indexed list $\mathcal{N}$.

5. $\mathcal{A}$ may request interaction with $\mathrm{UIssue}(N_\ell, r_\ell, i, S)$ by specifying an index ℓ on $\mathcal{N}$ and any $S \subseteq A$ and any $\underline{i}$. Let *cred* be the output of UIssue. The experiment stores $cred_{\ell,k} := cred$ on a k-indexed list $\mathcal{C}_\ell$. $\mathcal{A}$ is given *cred*.

6. $\mathcal{A}$ may request interaction with $\mathrm{UShow}(N_{\ell'}, r_{\ell'}, N_\ell, r_\ell, cred_{\ell',k}, \phi, \mathfrak{E}, wit)$ by specifying $\phi \in \Phi$, $\underline{\mathfrak{E}, wit}$, two indices ℓ, ℓ' on $\mathcal{N}$, and an index k on $\mathcal{C}_{\ell'}$.

7. $\mathcal{A}$ may request a transcript for $\mathrm{UFormNym}(usk_j, opk_t^*) \leftrightarrow \mathrm{OFormNym}(osk_t^*)$ by specifying a user index j and $t \in \{0, 1\}$. Let (N, r) be the output of UFormNym. The experiment stores $(j_\ell, t_\ell, N_\ell, r_\ell) := (j, t, N, r)$ on $\mathcal{N}$.

8. $\mathcal{A}$ may request a transcript for $\mathrm{UIssue}(N_\ell, r_\ell, i, S) \leftrightarrow \mathrm{OIssue}(osk_t^*, N_\ell, i, S)$ by specifying an index ℓ on $\mathcal{N}^*$, $t \in \{0, 1\}$, $\underline{i}$, and $S \subseteq A$. Let *cred* be the output of UIssue. The experiment stores $cred_{\ell,k} := cred$ on $\mathcal{C}_\ell$. $\mathcal{A}$ is given *cred*.

9. $\mathcal{A}$ may request a transcript for $\mathrm{UShow}(N_{\ell'}, r_{\ell'}, N_\ell, r_\ell, cred_{\ell',k}, \phi, \mathfrak{E}, wit) \leftrightarrow \mathrm{OShow}($ $osk_t^*, N_\ell, opk_{t'}^*, \phi, \mathfrak{E})$ by specifying ϕ, $\underline{\mathfrak{E}, wit}$, $t, t' \in \{0, 1\}$, and two indices ℓ, ℓ' on $\mathcal{N}^*$ and an index k on $\mathcal{C}_{\ell'}^*$.

10. Eventually, $\mathcal{A}$ outputs a formula $\phi^* \in \Phi$, a set $\underline{V^*}$, epoch information $\underline{\mathfrak{E}^*}$, and a pseudonym N^*.

11. The experiment runs $z \leftarrow \text{OShow}(osk_0^*, N^*, opk_1^*, \phi^*, \underline{\mathfrak{E}^*})$ to interact with $\mathcal{A}$.
12. If $z = 1$, $\underline{\text{EpochVerify}(opk_1^*, V^*, \mathfrak{E}^*) = 1}$, and $\mathcal{A}$ never requested interaction with $\text{OIssue}(osk_1^*, \cdot, i, S)$ where $S \in \phi^*$ and $\underline{i \in V^*}$, the experiment outputs 1.
13. Otherwise it outputs 0.

Note that $\mathcal{A}$ gets oracle access to EpochCreate for the two honest organizations, which it can use to create arbitrary revocation situations.

Alongside ϕ^*, the challenge that $\mathcal{A}$ chooses now includes a set V^* of valid identifiers and $\mathfrak{E}^*$, which should be accepted by EpochVerify (without this requirement, there is no link between V^* and $\mathfrak{E}^*$, which would make accumulator proofs impossible). $\mathcal{A}$ wins the game if it can pass the show protocol for ϕ^* and $\mathfrak{E}^*$ without having queried a credential that both fulfills the attribute formula *and* is unrevoked.

Again, the necessary changes are minor.

7 Conclusion

In this thesis, we have shown how the Camenisch et al. accumulator [CKS09] can be applied for revocation in group signature schemes. For this, we formally defined revocation semantics of group signature schemes with accumulator revocation and gave a generic construction based on our ring signature construction. We also defined revocation semantics for anonymous credential systems.

In the case of group signatures, accumulator revocation turns out to be very useful. In contrast to the standard revocation list semantics, our definition implies backward-unlinkability, which means that signatures of users whose signing rights were revoked are still not traceable to the signer (and old signatures stay valid). This is desirable in many contexts where revocation occurs in cases other than misuse. As a result of using accumulators, epoch information in these schemes is quite short in size and can be sent alongside messages. Anyone can verify the signature offline and in time independent of the number of revoked users (as opposed to similar approaches with revocation lists where this effort is linear in the number of revoked users). The burden of updating (accumulator) witnesses lies with the signers (who need to do this once for every new epoch they want to sign messages in), which is reasonable as signing typically occurs less often than verifying.

For anonymous credential systems, however, accumulator revocation shows some weaknesses. First, the burden of updating accumulator witnesses lies with the users (although this computation can be outsourced to untrusted third parties [CKS09]). This may prove problematic, for example, on smart cards with limited computational resources and limited internet access. This holds especially true since the smart card *terminal* cannot easily fetch the credential's witness for the smart card, considering that the terminal should not learn the credential identifier needed to compute the witness. Second, as pointed out in Section 6.2, revocation of credentials can be a dangerous tool given to organizations that they can use to defeat a user's anonymity. For example, setting the set of valid credential identifiers to $V = \{i\}$ basically yields an oracle to distinguish a shown credential from credential i. In theory, users can judge from the set V whether or not they want to let the organization narrow its credential identifier down to one of the values in V. However, in practice this is difficult: even if the set V seemingly contains many credential identifiers, the user's identifier i might still be the only one that was ever issued a credential for. Furthermore, having to check the set V for anonymity threats negates the desirable effect of accumulators that they represent V in a very compact form. Storing or processing the complete set V might again represent a problem for smart cards.

Bibliography

[BB04] Dan Boneh and Xavier Boyen. Short Signatures Without Random Oracles. In *Advances in Cryptology - EUROCRYPT 2004*, number 3027 in Lecture Notes in Computer Science, pages 56–73. Springer Berlin Heidelberg, 2004.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology–CRYPTO 2004*, pages 41–55. Springer, 2004.

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles. In *Theory of Cryptography*, number 3876 in Lecture Notes in Computer Science, pages 60–79. Springer Berlin Heidelberg, 2006.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *Advances in Cryptology — EUROCRYPT 2003*, number 2656 in Lecture Notes in Computer Science, pages 614–629. Springer Berlin Heidelberg, 2003.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 390–399. ACM, 2006.

[Bra97] Stefan Brands. Rapid Demonstration of Linear Relations Connected by Boolean Operators. In *Advances in Cryptology — EUROCRYPT '97*, number 1233 in Lecture Notes in Computer Science, pages 318–333. Springer Berlin Heidelberg, 1997.

[BS04] Dan Boneh and Hovav Shacham. Group Signatures with Verifier-local Revocation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 168–177, New York, NY, USA, 2004. ACM.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In *Public Key Cryptography – PKC 2009*, number 5443 in Lecture Notes in Computer Science, pages 481–500. Springer Berlin Heidelberg, 2009.

[CS97] Jan Camenisch and Markus Stadler. *Proof systems for general statements about discrete logarithms.* Citeseer, 1997.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives. In *Topics in Cryptology — CT-RSA 2015*, number 9048 in Lecture Notes in Computer Science, pages 127–144. Springer International Publishing, April 2015.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Prove All NP Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design (Extended Abstract). In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, number 263 in Lecture Notes in Computer Science, pages 171–185. Springer Berlin Heidelberg, 1987.

[Lys02] Anna Lysyanskaya. *Signature schemes and applications to cryptographic protocol design*. PhD thesis, Massachusetts Institute of Technology, 2002.

[NF06] Toru Nakanishi and Nobuo Funabiki. A Short Verifier-Local Revocation Group Signature Scheme with Backward Unlinkability. In *Advances in Information and Computer Security*, number 4266 in Lecture Notes in Computer Science, pages 17–32. Springer Berlin Heidelberg, 2006.

[Oka06] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In *Theory of cryptography*, pages 80–99. Springer, 2006.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.

[Sch15] Berry Schoenmakers. Cryptographic Protocols. 2015.