An extended abstract of this paper will appear in the Proceedings of the Seventh Conference on Security and Cryptography for Networks – SCN 2010, Antalia, Italy, September 2010.
The original publication is available at `www.springerlink.com`.

Get Shorty via Group Signatures without Encryption

P. Bichsel[1], J. Camenisch[1], G. Neven[1], N.P. Smart[2], and B. Warinschi[2]

[1] IBM Research – Zurich,
Switzerland.
{pbi,jca,nev}@zurich.ibm.com
[2] Dept Computer Science,
Univeristy of Bristol,
United Kingdom.
{nigel,bogdan}@cs.bris.ac.uk

Abstract. Group signatures allow group members to anonymously sign messages in the name of a group such that only a dedicated opening authority can reveal the exact signer behind a signature. In many of the target applications, for example in sensor networks or in vehicular communication networks, bandwidth and computation time are scarce resources and many of the existent constructions simply cannot be used. Moreover, some of the most efficient schemes only guarantee anonymity as long as no signatures are opened, rendering the opening functionality virtually useless.
In this paper, we propose a group signature scheme with the shortest known signature size and favorably comparing computation time, whilst still offering a strong and practically relevant security level that guarantees secure opening of signatures, protection against a cheating authority, and support for dynamic groups. Our construction departs from the popular sign-and-encrypt-and-prove paradigm, which we identify as one source of inefficiency. In particular, our proposal does not use standard encryption and relies on re-randomizable signature schemes that hide the signed message so as to preserve the anonymity of signers.
Security is proved in the random oracle model assuming the XDDH, LRSW and SDLP assumptions and the security of an underlying digital signature scheme. Finally, we demonstrate how our scheme yields a group signature scheme with verifier-local revocation.

Key words: Group signatures, pairings, group signature security definition.

1 Introduction

Group signatures, introduced in 1991 by Chaum and van Heyst [19], allow members of a group to anonymously sign messages on behalf of the whole group. For example, they allow an employee of a company to sign a document in such a way that the verifier only learns that it was signed by an employee, but not by which employee. Group membership is controlled by a *Group Manager*, who can add users (called *Group Members*) to the group. In addition, there is an *Opener* who can reveal the identity of signers in the case of disputes. In some schemes, such as the one we propose, the tasks of adding members and revoking anonymity are combined into a single role. In the systems proposed in [3, 16, 34], group membership can be selectively revoked, i.e., without affecting the signing ability of the remaining members.

Security notions. Since 1991 a number of security properties have been developed for group signatures including unforgeability, anonymity, traceability, unlinkability, and non-frameability. In 2003 Bellare, Micciancio, and Warinschi [4] developed what is now considered the standard security model for group signatures. They propose two security properties for static groups called *full anonymity* and *full traceability* and show that these capture the previous security requirements of unforgeability, anonymity, traceability, and unlinkability. Bellare, Shi, and Zhang [7] extended the notions of [4] to dynamic groups and added the notion of *non-frameability* (or exculpability), by which the Group Manager and Opener together cannot produce a signature that can be falsely attributed to an honest Group Member.

Boneh and Shacham [11] proposed a relaxed anonymity notion called *selfless anonymity* where signers can trace their own signatures, but not those of others. This weakening, however, leads to the following feature: if a group member signed a message but forgot that she signed it, then she can recover this information from the signature itself. Other schemes [10, 12, 13] weaken the anonymity notion by disallowing opening oracle queries, providing only so-called CPA-anonymity. This is a much more serious limitation: in practice it means that all security guarantees are lost as soon as a single signature is opened, thereby rendering the opening functionality virtually useless. As we've witnessed for the case of encryption [8], CCA2-security is what can make it into practice.

In this work, we consider a hybrid between the models of [7] and [11] that combines the dynamic group setting and the non-frameability notion of [7] with the selfless anonymity notion and the combined roles of Group Manager and Opener of [11]. We stress however that we prove security under the practically relevant CCA2-anonymity notion, rather than the much weaker CPA-anonymity notion. Yet still, our scheme compares favourably with all known schemes that offer just CPA-anonymity.

Construction paradigms. Many initial group signature schemes were based on the Strong-RSA assumption [2, 3, 16]. In recent years the focus has shifted to schemes based on bilinear maps [10, 11, 17, 26, 33], which are the most efficient group signatures known today, both in terms of bandwidth and computational efficiency.

Most existing group signature schemes follow the construction paradigm where a group signature consists of an anonymous signature, an encryption of the signer's identity under the Opener's public key, and a non-interactive zero-knowledge (NIZK) proof that the identity contained in the encryption is indeed that of the signer. While very useful as an insight, this construction paradigm seems to stand in the way of more efficient schemes. In this paper, we depart from the common paradigm and construct a group signature scheme that consists solely of an anonymous signature scheme and a NIZK proof, removing the need to encrypt the identity of the signer. We thereby obtain the most efficient group signature scheme currently known, both in terms of bandwidth and computational resources (see Appendix 6).

It is surprising that we can do without a separate encryption scheme, given that group signatures as per [4] are known to imply encryption [1]. This implication however does not hold for group signatures with selfless anonymity, giving us the necessary slack to construct more efficient schemes while maintaining a practically relevant security level.

Our scheme. In our construction each Group Member gets a Camenisch-Lysyanskaya (CL) [17] signature on a random message as a secret key. To produce a group signature, the Group Member re-randomizes this signature and produces a NIZK proof that she knows the message underlying the signature. The novel feature is that the Opener (alias Group Manager) can use information collected during the joining phase to test which user created the signature, without the need for a separate encryption.[3] A disadvantage is that opening thereby becomes a linear operation in the number of Group Members. Since opening signatures is a rather exceptional operation and is performed by the Group Manager who probably has both the resources and the commercial interest to expose traitors, we think that this is a reasonable price to pay.

CL signatures and NIZK proofs have been combined before to produce "group-like" signatures, most notably in the construction of pairing-based DAA schemes [14, 21, 22]. DAA schemes are not genuine group signatures, however, as there is no notion of an Opener.

Finally, we note that from a certain class of group signature schemes as per our definitions (that includes our scheme), one can build a group signature scheme with verifier-local revocation (VLR) [11]. Such a scheme allows verifiers to check whether a signature was placed by a revoked group member by matching it against a public revocation list. The converse is not true, i.e., a VLR scheme does not automatically yield a group signature as per our definitions, as it does not provide a way to open individual signatures (rather than revoking all signatures by one signer). We refer to Section 3.2 for details.

[3] If the random messages were known to the Group Manager, he could open group signatures simply by verifying the re-randomized signatures against the issued random messages. To achieve non-frameability, however, the random message is only known to the Group Member, so opening in our scheme is slightly more involved.

2 Preliminaries

Notation. If S is a set, we denote the act of sampling from S uniformly at random and assigning the result to the variable x by $x \leftarrow S$. If S consists of a single element $\{s\}$, we abbreviate this to $x \leftarrow s$. We let $\{0,1\}^*$ and $\{0,1\}^t$ denote the set of binary strings of arbitrary length and length t respectively, and let ε denote the empty string. If A is an algorithm, we denote the action of obtaining x by invoking A on inputs $y_1, \ldots, y_n$ by $x \leftarrow A(y_1, \ldots, y_n)$, where the probability distribution on x is determined by the internal coin tosses of A. We denote an interactive protocol P as $P = (P_0, P_1)$. Executing the protocol on input in_0 and in_1, resulting in the respective output out_0 and out_1, we write as $\langle out_0; out_1 \rangle \leftarrow \langle P_0(in_0); P_1(in_1) \rangle$. If $\mathbf{arr}$ is an array or list we let $\mathbf{arr}[i]$ denote the ith element in the array/list.

Digital Signature Scheme. We will use a digital signature scheme consisting of three algorithms, namely a key generation algorithm $\mathsf{DSKeyGen}$, a signing algorithm DSSign, and a signature verification algorithm $\mathsf{DSVerify}$. In our setting the key generation will be executed between a user and a certification authority (CA). It might be an interactive algorithm leading to the user getting a secret key sk and the CA as well as the user get the public key pk corresponding to the secret key. The signing algorithm accepts a secret key sk and a message m as input and returns a signature $\bar{\sigma} \leftarrow \mathsf{DSSign}(sk, m)$. The signature is constructed such that the verification algorithm upon input a message m', a public key pk, and a signature $\bar{\sigma}$ returns $\mathsf{DSVerify}(pk, m', \bar{\sigma})$, which is true if both $m' \equiv m$, and sk corresponds to pk and false otherwise. The signature scheme must satisfy the notion of unforgeability under chosen-message attacks [29].

Number-Theoretic Background. Our construction will make extensive use of asymmetric pairings on elliptic curves. In particular we will use the following notation, for a given security parameter η,

- $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order $q = \Theta(2^\eta)$.
- We write the group operations multiplicatively, and elements in $\mathbb{G}_1$ will generally be denoted by lower case letters, elements in $\mathbb{G}_2$ by lower case letters with a "tilde" on them, and elements in $\mathbb{Z}_q$ by lower case Greek letters.
- We fix a generator g (resp. $\tilde{g}$) of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$).
- There is a computable map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:
  - For all $x \in \mathbb{G}_1$, $\tilde{y} \in \mathbb{G}_2$ and $\alpha, \beta \in \mathbb{Z}_q$ we have $\hat{e}(x^\alpha, \tilde{y}^\beta) = \hat{e}(x, \tilde{y})^{\alpha\beta}$.
  - $\hat{e}(g, \tilde{g}) \neq 1$.

Following [28] we call a pairing of Type-1 if $\mathbb{G}_1 = \mathbb{G}_2$, of Type-2 if $\mathbb{G}_1 \neq \mathbb{G}_2$ and there exists a computable homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, and of Type-3 if $\mathbb{G}_1 \neq \mathbb{G}_2$ and no such homomorphism exists. In addition, in [20, 32] a further Type-4 pairing is introduced in which $\mathbb{G}_2$ is a group of order q^2, namely the product of $\mathbb{G}_1$ with the $\mathbb{G}_2$ used in the Type-3 pairing setting. In practice Type-3 pairings offer the most efficient implementation choices, in terms of both bandwidth and computational efficiency.

Associated to pairings are the following computational assumptions, which we shall refer to throughout this paper:

Assumption 1 (LRSW) *With the notation above we let $\tilde{x}, \tilde{y} \in \mathbb{G}_2$, with $\tilde{x} = \tilde{g}^{\alpha}$, $\tilde{y} = \tilde{g}^{\beta}$. Let $O_{\tilde{x},\tilde{y}}(\cdot)$ be an oracle that, on input of a value $\mu \in \mathbb{Z}_q$, outputs a triple $A = (a, a^{\beta}, a^{\alpha+\mu\alpha\beta}) \in \mathbb{G}_1^3$ for a randomly chosen $a \in \mathbb{G}_1$. Then for all probabilistic polynomial time adversaries $\mathcal{A}$, the quantity $\nu(\eta)$, defined as follows, is a negligible function:*

$$\nu(\eta) := \Pr[\alpha \leftarrow \mathbb{Z}_q; \beta \leftarrow \mathbb{Z}_q; \tilde{x} \leftarrow \tilde{g}^{\alpha}; \tilde{y} \leftarrow \tilde{g}^{\beta}; (\mu, a, b, c) \leftarrow \mathcal{A}^{O_{\tilde{x},\tilde{y}}(\cdot)}(\tilde{x}, \tilde{y}) : \mu \notin Q \wedge a \in \mathbb{G}_1 \wedge b = a^{\beta} \wedge c = a^{\alpha+\mu\alpha\beta}]$$

where Q is the set of queries passed by $\mathcal{A}$ to its oracle $O_{\tilde{x},\tilde{y}}(\cdot)$.

This assumption was introduced by Lysyanskaya et al. [30], in the case $\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$ for groups that are not known to admit an efficient bilinear map. The authors showed in the same paper, that this assumption holds for generic groups, and is independent of the decisional Diffie-Hellman (DDH) assumption. However, it is always applied in protocols for which the groups admit a pairing, and the above asymmetric version is the version that we will require.

Assumption 2 (XDDH; SXDH) *We say XDDH to hold in the pairing groups if DDH is hard in $\mathbb{G}_1$, i.e., if given a tuple $(g, g^{\mu}, g^{\nu}, g^{\omega})$ for $\mu, \nu \leftarrow \mathbb{Z}_q$ it is hard to decide whether $\omega = \mu\nu \mod q$ or random. We say SXDH holds if DDH is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Note that neither XDDH nor SXDH hold in the case of Type-1 pairings. For the others types of pairings XDDH is believed to hold, and only for Type-3 pairings SXDH is believed to hold.

To demonstrate the non-frameability of our scheme we require an additional assumption, which we call the symmetric Discrete Logarithm Assumption (SDLP).

Assumption 3 (SDLP) *Given the tuple $(g^{\mu}, \tilde{g}^{\mu}) \in \mathbb{G}_1 \times \mathbb{G}_2$ computing μ is a hard problem.*

This is a non-standard assumption which, however, implicitly underlies many asymmetric pairing versions of protocols in the literature that are described in the symmetric pairing setting only. Note that the input to the SDLP problem can always be checked to be a valid input, as given $(h, \tilde{h})$ one can always check whether $\hat{e}(g, \tilde{h}) = \hat{e}(h, \tilde{g})$.

The above three assumptions are what we require to prove our scheme secure. However, for comparison with other schemes in the literature, we recap on the following two problems, introduced in [9] and [10], respectively.

Assumption 4 (q-SDH) *In a pairing situation as above, this assumption implies that given a q-tuple $(\tilde{g}^{\gamma}, \tilde{g}^{\gamma^2}, \ldots, \tilde{g}^{\gamma^q})$ for some hidden value of γ, it is hard to output a pair $(g^{1/(\gamma+\alpha)}, \alpha)$ for some $\alpha \in \mathbb{Z}_q$.*

Assumption 5 (DLIN) *Given $a, b, c, a^{\alpha}, b^{\beta}, c^{\gamma} \in \mathbb{G}_1$, this assumption says it is hard to determine whether $\alpha + \beta = \gamma$.*

CL Signatures. Our group signature scheme is based on the pairing-based Camenisch-Lysyanskaya (CL) signature scheme [17] (Scheme A in their paper), which is provably secure under the LRSW assumption. The scheme assumes three cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of prime order $q = \Theta(2^\eta)$, with a pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and two generators $g \in G_1$ and $\tilde{g} \in \mathbb{G}_2$.

The secret key of the CL signature scheme consists of $\alpha, \beta \leftarrow \mathbb{Z}_q$ and the public key is defined as $(\tilde{x}, \tilde{y}) \leftarrow (\tilde{g}^\alpha, \tilde{g}^\beta) \in \mathbb{G}_2^2$. Computing a signature $s \in \mathbb{G}_1^3$ on a message $m \in \mathbb{Z}_q$ is done by choosing $a \leftarrow \mathbb{G}_1$, calculating $b \leftarrow a^\beta$ and $c \leftarrow a^{\alpha+m\alpha\beta}$, and setting $s \leftarrow (a, b, c)$. Finally, a tuple $(a, b, c) \in \mathbb{G}_1^3$ is a valid signature on a message $m \in \mathbb{Z}_q$ if both $\hat{e}(a, \tilde{x}) = \hat{e}(b, \tilde{g})$ and $\hat{e}(a, \tilde{x}) \cdot \hat{e}(b, \tilde{x})^m = \hat{e}(c, \tilde{g})$ hold.

Theorem 1 ([17]). *The CL signature scheme A is existentially unforgeable against adaptive chosen message attacks [29] under the LRSW assumption.*

CL signatures are re-randomizable, i.e., given a valid signature $(a, b, c) \in \mathbb{G}_1^3$ on a message m, the signature $(a^r, b^r, c^r) \in \mathbb{G}_1^3$ will also be valid for any $r \in \mathbb{Z}_q^*$. This re-randomization property is central to our new group signature scheme.

Sigma Protocols. We will use a number of protocols to prove knowledge of discrete logarithms (and, more generally, of pre-images of group homomorphisms) and properties about them. This section recaps some basic facts about such protocols and the notation we will use.

Let $\phi : \mathbb{H}_1 \to \mathbb{H}_2$ be a group homomorphism with $\mathbb{H}_1$ and $\mathbb{H}_2$ being two groups of order q and let $y \in \mathbb{H}_2$. We will use additive notation for $\mathbb{H}_1$ and multiplicative notation for $\mathbb{H}_2$. By $PK\{(x) : y = \phi(x)\}$ we denote the Σ-protocol for a zero-knowledge proof of knowledge of x such that $y = \phi(x)$ [15, 18]. Σ-protocols for group homomorphisms are three-move protocols where the prover chooses $\mathsf{rnd} \leftarrow \mathbb{H}_1$ and sends $\mathsf{Comm} \leftarrow \phi(\mathsf{rnd})$ to the verifier; the verifier sends back a random $\mathsf{Cha} \leftarrow \mathbb{H}_1$; the prover then sends $\mathsf{Rsp} = \mathsf{rnd} - \mathsf{Cha} \cdot x$; and the verifier checks that $\phi(\mathsf{Rsp})\phi(x)^{\mathsf{Cha}} = \mathsf{Comm}$. It is well-known that basic Σ-protocols for group homomorphisms are honest-verifier zero-knowledge proofs of knowledge of the pre-image of the group homomorphism. There is a number of different ways to turn any honest-verifier Σ-protocol into a protocol that is full zero-knowledge with perfect simulation and negligible soundness error (e.g., [23, 25]). We denote the full zero-knowledge variant of a Σ-protocol $PK\{\ldots\}$ as $FPK\{\ldots\}$.

The well-known Schnorr identification protocol is the special case $PK\{(x) : y = g^x\}$, i.e., $\phi(x) = g^x$ where g is a generator of a subgroup of order q of $\mathbb{Z}_p$. Let $\phi_1 : \mathbb{H}_1 \to \mathbb{H}_2$ and $\phi_2 : \mathbb{H}_1 \to \mathbb{H}_2$. We often write $y_1 = \phi_1(x_1) \wedge y_2 = \phi_2(x_2)$ to denote $\phi(x_1, x_2) := (\phi_1(x_1), \phi_2(x_2))$ or $y_1 = \phi_1(x) \wedge y_2 = \phi_2(x)$ to denote $\phi(x) := (\phi_1(x), \phi_2(x))$.

The "signature" variant of a Σ-protocol is obtained by applying the Fiat-Shamir heuristic [27] to the above Σ-protocol. We denote such a "signature-proof-of-knowledge" on a message $m \in \{0,1\}^*$ by, $SPK\{(x) : y = \phi(x)\}(m)$. That is, when we say that $\Sigma \leftarrow SPK\{(x) : y = \phi(x)\}(m)$ is computed, we mean that a random $\mathsf{rnd} \leftarrow \mathbb{H}_1$ is chosen and the pair $\Sigma \leftarrow (\mathsf{Cha}, \mathsf{Rsp})$ is computed where $\mathsf{Cha} \leftarrow \mathcal{H}(\phi \| y \| \phi(\mathsf{rnd}) \| m)$, $\mathsf{Rsp} \leftarrow \mathsf{rnd} - \mathsf{Cha} \cdot x$ and $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q$ is a suitable hash function. Note that

$\Sigma \in \mathbb{Z}_q \times \mathbb{H}_1$. We say that $\Sigma = (\mathsf{Cha}, \mathsf{Rsp})$ is valid with respect to y and ϕ if $\mathsf{Cha} = \mathcal{H}(\phi \| y \| y^{\mathsf{Cha}} \phi(\mathsf{Rsp}) \| m)$ holds; typically y and ϕ will be clear from the context and we will just say that "Σ is valid." We further note that a unique specification of the statement (e.g., $(x) : y = \phi(x)$) that SPK "proves" needs to be included as an argument to the hash function, i.e., here $\phi \| y$, where ϕ stands for the description of the whole algebraic setting. In the random oracle model [6], one can use the forking lemma [31, 5] to extract the secrets from these SPKs if correct care is taken that the prover can indeed be efficiently rewound. Moreover, in the random oracle model one can simulate SPKs for unknown secrets by choosing $\mathsf{Cha}, \mathsf{Rsp} \leftarrow \mathbb{Z}_q$ at random and programming the random oracle so that $\mathcal{H}(\phi \| y \| y^{\mathsf{Cha}} \phi(\mathsf{Rsp}) \| m) = \mathsf{Cha}$.

3 Definitions

As mentioned in the Introduction, we propose a notion that builds a hybrid between [7] and [11]. Consequently, our definitions describe a dynamic group signature scheme with a combined role of Group Manager and Opener that obtains selfless anonymity, traceability, and non-frameability.

3.1 Syntax

A group signature scheme consists of a set of users with a unique index i who can produce signatures on behalf of the group. Initially users must interact with a trusted party to establish a public key pair. Users can become Group Members via an interaction with the Group Manager. After the interaction the user obtains a secret signing key that she can use to produce signatures on behalf of the group. The Group Manager obtains a piece of information that he can later use to identify signatures created by the user. In addition, both parties obtain some piece of publicly available information, which certifies the fact that the particular user has joined the group.

As remarked earlier, in our models we put more trust in the Group Manager by requiring that he is also in charge of opening signatures. The syntax that we require is as follows.

Definition 1. *A group signature scheme* GS *extended by a PKI is given by a tuple*

$$(\mathsf{GSetup}, \mathsf{PKIJoin}, (\mathsf{GJoin}_U, \mathsf{GJoin}_M), \mathsf{GSign}, \mathsf{GVerify}, \mathsf{GOpen}, \mathsf{GJudge})$$

where:

1. GSetup *is a setup algorithm. It takes as input a security parameter* 1^η *and produces a tuple* $(gpk, gmsk)$*, where* gpk *is a group public key and* $gmsk$ *is the Group Manager's secret key. To simplify notation we assume that* $gmsk$ *always includes the group public key. Note that the group public key contains system parameters, which need to be checked by all entities not involved in there generation.*
2. $\mathsf{PKIJoin}$ *is an algorithm executed by a user to register with a certification authority (CA). It takes as input the index of the user* i *and the security parameter* 1^η*. The output of the protocol is the key pair* $(\mathbf{usk}[i], \mathbf{upk}[i])$ *consisting of user secret key and user public key or* $\perp$ *in case of a failure. The user public key* $\mathbf{upk}[i]$ *is sent to the CA, who makes it available such that anyone can get an authentic copy of it.*

3. $\mathsf{GJoin} = (\mathsf{GJoin}_M, \mathsf{GJoin}_U)$ *is a two-party interactive protocol used to add new users to the group. The input for the user is* $(i, \mathbf{usk}[i], gpk)$*, i.e., the index of the user, the user secret key, and the group public key. The input for the Group Manager is* $(i, \mathbf{upk}[i], gmsk)$*, i.e., the user index, the user public key, and the Group Manager's secret key.*
   As a result of the interaction, the user obtains her group signing key $\mathbf{gsk}[i]$*, and the Group Manager obtains some registration information* $\mathbf{reg}[i]$ *(which will later be used to trace signatures of* i*). If the protocol fails, the output of both parties is set to* $\perp$*.*
4. GSign *is the algorithm users employ to sign on behalf of the group. It takes as input an individual user signing key* $\mathbf{gsk}[i]$ *and the message* $m \in \{0,1\}^*$ *to be signed, and outputs a signature* σ*. We write* $\sigma \leftarrow \mathsf{GSign}(\mathbf{gsk}[i], m)$ *for the process of obtaining signature* σ *on* m *with secret key* $\mathbf{gsk}[i]$*.*
5. $\mathsf{GVerify}$ *is the signature verification algorithm. It takes as input* (gpk, m, σ)*, i.e., the group public key, a message and a group signature, and returns* 0 *if the signature is deemed invalid and* 1 *otherwise.*
6. GOpen *is the algorithm for opening signatures. It takes as input* $(gmsk, m, \sigma, \mathbf{reg})$*, i.e., the Group Manager's secret key, a message, a valid group signature on the message, and the registration information table* $\mathbf{reg}$*, and returns a user index* $i \in [n]$ *and a proof* π *that user* i *produced signature* σ*, or it returns* $\perp$*, indicating that opening did not succeed.*
   We assume that the opening algorithm, before outputting (i, π)*, always checks that the user* i *is registered, i.e., that* $\mathbf{reg}[i] \neq \perp$*, and that the proof* π *passes the judging algorithm (see the next item). If either of these checks fails, the opening algorithm outputs* $\perp$*.*
7. GJudge *is the judging algorithm. It takes as input a message* m*, a group signature* σ *on* m*, the group public key* gpk*, a user index* i*, the user public key* $\mathbf{upk}[i]$*, and a proof* π *and outputs* 1 *or* 0*, expressing whether the proof shows that user* i *created signature* σ *or not.*
   We assume that the judging algorithm verifies the signature using the $\mathsf{GVerify}$ *algorithm on input* gpk*,* m*, and* σ*.*

3.2 Security notions

In this section we give the security definitions that we require from group signature schemes. We describe the oracles that are involved in our definitions, as well as the restrictions that we put on their uses. These oracles use some shared global state of the experiments in which they are provided to the adversary. In particular, at the time of their use, the sets of honest and corrupt users are defined. Also the oracles have access to the global information contained in $\mathbf{upk}$. For honest users the oracles have access to $\mathbf{gsk}$ and if the Group Manager is uncorrupted they also have access to $\mathbf{reg}$. We assume that at the beginning of the execution, the content of each entry in these arrays is set to $\perp$ (uninitialized).

We consider a setting with n users divided (statically) into sets $\mathcal{HU}$ and $\mathcal{DU}$ of honest and dishonest users, respectively. Even though our definitions appear to consider static corruptions only, one can easily see (by taking an upper bound on the number

of users for n and guessing the indices of "target" users upfront) that they actually imply security in the dynamic case. However, the latter comes at the cost of losing a factor n in reduction tightness for traceability and non-frameability, and of $n^2/2$ for anonymity. For some notions the adversary $\mathcal{A}$ is actually a pair of algorithms $(\mathcal{A}_0, \mathcal{A}_1)$; we implicitly assume that $\mathcal{A}_0$ can pass state information to $\mathcal{A}_1$. Our security notions make use of the following oracles:

- $\mathsf{Ch}(b, \cdot, \cdot, \cdot)$ is the challenge oracle for defining anonymity. It accepts as input a triple formed from two identities $i_0, i_1 \in \mathcal{HU}$ and a message m, and returns a signature $\sigma^* \leftarrow \mathsf{GSign}(\mathbf{gsk}[i_b], m)$ under the signing key of user i_b, where b is a parameter of the experiment. This oracle can only be called once.
- $\mathsf{SetUPK}(\cdot, \cdot)$ takes an input the index of a user $i \in \mathcal{DU}$ and a value upk. If $\mathbf{reg}[i] \equiv \perp$ it sets the user's public key $\mathbf{upk}[i] \leftarrow upk$. The oracle can only be called before user i joins the group.
- $\mathsf{GJoin}_{UD}(\cdot)$ is an oracle that takes as input an honest user index $i \in \mathcal{HU}$ and executes the user side of the join protocol for i, i.e., $\mathsf{GJoin}_U(i, \mathbf{usk}[i], gpk)$. The local output of the protocol is stored in $\mathbf{gsk}[i]$. This oracle can be used by an adversary to execute the registration protocol with an honest user, the adversary playing the role of the Group Manager (when the latter is corrupt).
- $\mathsf{GJoin}_{DM}(\cdot)$ is an oracle that takes as input the index of a corrupt user $i \in \mathcal{DU}$ and simulates the execution of the join protocol for the (honest) Group Manager, i.e., $\mathsf{GJoin}_M(i, \mathbf{upk}[i], gmsk)$. The local output of the protocol is stored in $\mathbf{reg}[i]$. This oracle can be used by an adversary to execute the registration protocol with the (honest) Group Manager on behalf of any corrupt user.
- $\mathsf{GSign}(\cdot, \cdot)$ accepts as input pairs $(i, m) \in \mathcal{HU} \times \{0,1\}^*$ and obtains a signature on m under $\mathbf{gsk}[i]$ if the user is not corrupt, and its signing key is defined.
- $\mathsf{GOpen}(\cdot, \cdot)$ accepts as input a message-signature pair (m, σ) and returns the result of the function call $\mathsf{GOpen}(gmsk, m, \sigma, \mathbf{reg})$. The oracle refuses to open the signature attained through a call to the Ch oracle, i.e., $\sigma \equiv \sigma^*$.

Note that, depending on the precise group signature scheme, the oracles $\mathsf{GJoin}_{UD}(\cdot)$ and $\mathsf{GJoin}_{DM}(\cdot)$ may require multi-stages, i.e., interaction between the oracle and the adversary to complete the functionality. If this is the case we assume that these stages are executed by the adversary in a sequential order, as if the oracles are a single stage. Thus, we do not allow the adversary to interleave separate executions of the GJoin protocol, or execute multiple of them in parallel.

Correctness. We define the correctness of a group signature scheme GS through a game in which an adversary is allowed to requests a signature on some message by any of the honest players. The adversary wins if either (1) the resulting signature does not pass the verification test, (2) the signature is opened as if it were produced by a different user, or (3) the proof produced by opening the signature does not pass the judging algorithm. The experiment is detailed in Figure 1. We say that GS is correct if for any adversary $\Pr[\mathbf{Exp}^{\mathrm{corr}}_{\mathsf{GS},A}(\eta) = 1]$ is 0.

$\mathbf{Exp}^{\mathrm{corr}}_{\mathsf{GS},\mathcal{A}}(\eta)$
$\mathcal{HU} \leftarrow \{1,\ldots,n\}$; $\mathcal{DU} \leftarrow \emptyset$
$(gpk, gmsk) \leftarrow \mathsf{GSetup}(1^\eta)$
For $i \in \mathcal{HU}$
$(\mathbf{usk}[i], \mathbf{upk}[i]) \leftarrow \mathsf{PKIJoin}(i, 1^\eta)$
$\langle \mathbf{reg}[i]; \mathbf{gsk}[i] \rangle \leftarrow \langle \mathsf{GJoin}_M(i, \mathbf{upk}[i], gmsk); \mathsf{GJoin}_U(i, \mathbf{usk}[i], gpk) \rangle$
$(i, m) \leftarrow \mathcal{A}^{\mathsf{GSign}(\cdot,\cdot),\mathsf{GOpen}(\cdot,\cdot)}(gpk)$
If $i \notin \mathcal{HU}$ then return 0
$\sigma \leftarrow \mathsf{GSign}(\mathbf{gsk}[i], m)$
If $\mathsf{GVerify}(gpk, m, \sigma) = 0$ then return 1
$(j, \pi) \leftarrow \mathsf{GOpen}(gmsk, m, \sigma, \mathbf{reg})$
If $i \neq j$ or $\mathsf{GJudge}(m, \sigma, gpk, i, \mathbf{upk}[i], \pi) = 0$ then return 1
Return 0

$\mathbf{Exp}^{\text{anon-}b}_{\mathsf{GS},\mathcal{A}}(\eta)$
$\mathcal{DU} \leftarrow \mathcal{A}_0(1^\eta)$
$\mathcal{HU} \leftarrow \{1,\ldots,n\} \setminus \mathcal{DU}$
$(gpk, gmsk) \leftarrow \mathsf{GSetup}(1^\eta)$
For $i \in \mathcal{HU}$
$(\mathbf{usk}[i], \mathbf{upk}[i]) \leftarrow \mathsf{PKIJoin}(i, 1^\eta)$
$\langle \mathbf{reg}[i]; \mathbf{gsk}[i] \rangle \leftarrow \langle \mathsf{GJoin}_M(i, \mathbf{upk}[i], gmsk); \mathsf{GJoin}_U(i, \mathbf{usk}[i], gpk) \rangle$
$b' \leftarrow \mathcal{A}_1^{\mathsf{Ch}(b,\cdot,\cdot,\cdot),\mathsf{SetUPK}(\cdot,\cdot),\mathsf{GJoin}_{DM}(\cdot),\mathsf{GSign}(\cdot,\cdot),\mathsf{GOpen}(\cdot,\cdot)}(gpk)$
Return b'

$\mathbf{Exp}^{\mathrm{trace}}_{\mathsf{GS},\mathcal{A}}(\eta)$
$\mathcal{DU} \leftarrow \{1,\ldots,n\}$; $\mathcal{HU} \leftarrow \emptyset$
$(gpk, gmsk) \leftarrow \mathsf{GSetup}(1^\eta)$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{SetUPK}(\cdot,\cdot),\mathsf{GJoin}_{DM}(\cdot),\mathsf{GOpen}(\cdot,\cdot)}(gpk)$
If $\mathsf{GVerify}(gpk, m, \sigma) = 1$ and $\mathsf{GOpen}(gmsk, m, \sigma, \mathbf{reg}) = \perp$ then return 1
Else return 0

$\mathbf{Exp}^{\mathrm{nf}}_{\mathsf{GS},\mathcal{A}}(\eta)$
$(\mathcal{DU}, gpk) \leftarrow \mathcal{A}_0(1^\eta)$
$\mathcal{HU} \leftarrow \{1,\ldots,n\} \setminus \mathcal{DU}$
For $i \in \mathcal{HU}$
$(\mathbf{usk}[i], \mathbf{upk}[i]) \leftarrow \mathsf{PKIJoin}(i, 1^\eta)$
$(i, m, \sigma, \pi) \leftarrow \mathcal{A}_1^{\mathsf{SetUPK}(\cdot,\cdot),\mathsf{GJoin}_{UD}(\cdot),\mathsf{GSign}(\cdot,\cdot)}(1^\eta)$
If $i \notin \mathcal{HU}$ or $\mathsf{GVerify}(gpk, m, \sigma) = 0$ then return 0
If σ was oracle output of $\mathsf{GSign}(i, m)$ then return 0
If $\mathsf{GJudge}(m, \sigma, gpk, i, \mathbf{upk}[i], \pi) = 0$ then return 1
Return 0

Fig. 1. Experiments for defining the correctness and security of a group signature scheme. The particular restrictions on the uses of the oracles are described in Section 3.2.

Anonymity. *Anonymity* requires that group signatures do not reveal the identity of the signer. In the experiment that we consider, the adversary controls all of the dishonest

users. The adversary has access to a challenge oracle $\mathsf{Ch}(b, \cdot, \cdot, \cdot)$, which he can call only once with a triple (i_0, i_1, m), where i_0 and i_1 are the indices of two honest signers, and m is some arbitrary message. The answer of the oracle is a challenge signature $\sigma^* \leftarrow \mathsf{GSign}(\mathbf{gsk}[i_b], m)$. During the attack the adversary can (1) add corrupt users to the group of signers (via the $\mathsf{SetUPK}(\cdot, \cdot)$ and $\mathsf{GJoin}_M(\cdot)$ oracles), (2) require signatures of honest users on arbitrary messages via the GSign oracle, and (3) require opening of arbitrary signatures (except the signature σ^* obtained from the challenge oracle) via the GOpen oracle. The experiment is described in Figure 1. For any adversary that obeys the restrictions described above we define its advantage in breaking the anonymity of GS by

$$\mathbf{Adv}^{\mathrm{anon}}_{\mathsf{GS},A}(\eta) = \Pr[\mathbf{Exp}^{\text{anon-}1}_{\mathsf{GS},A}(\eta) = 1] - \Pr[\mathbf{Exp}^{\text{anon-}0}_{\mathsf{GS},A}(\eta) = 1]$$

We say that the scheme GS satisfies the anonymity property if for any probabilistic polynomial-time adversary, its advantage is a negligible function of η.

Traceability. Informally, *traceability* requires that no adversary can create a valid signature that cannot be traced to some user that had already been registered. We model the strong but realistic setting where all of the signers are corrupt and work against the group manager. In the game that we define, the adversary can add new signers using access to the GJoin_{DM} oracle and can request to reveal the signers of arbitrary signatures via the GOpen oracle. The goal of the adversary is to produce a valid message-signature pair (m, σ) that cannot be opened, i.e., such that the opening algorithm outputs $\perp$. For any adversary A we define its advantage in breaking traceability of group signature scheme GS by:

$$\mathbf{Adv}^{\mathrm{trace}}_{\mathsf{GS},A}(\eta) = \Pr[\mathbf{Exp}^{\mathrm{trace}}_{\mathsf{GS},A}(\eta) = 1]$$

We say that GS is traceable if for any probabilistic polynomial-time adversary, its advantage is a negligible function of the security parameter.

Non-frameability. Informally, *non-frameability* requires that even a cheating Group Manager cannot falsely accuse an honest user of having created a given signature. We model this property through a game that closely resembles that for traceability. The difference is that the adversary has the Group Manager's secret key (who is corrupt). During his attack the adversary can require honest users to join the group via the oracle GJoin_{UD}, and can obtain signatures of honest users through oracle GSign. The goal of the adversary is to produce a signature and a proof that this signature was created by an honest user (who did not actually create the signature). For any adversary A we define its advantage against non-frameability of group signature scheme GS by

$$\mathbf{Adv}^{\mathrm{nf}}_{\mathsf{GS},A}(\eta) = \Pr[\mathbf{Exp}^{\mathrm{nf}}_{\mathsf{GS},A}(\eta) = 1]$$

We say that scheme GS is non-frameable if for any probabilistic polynomial-time adversary, its advantage is a negligible function of η.

Remarks. The security definitions that we present depart from the more established ones in several ways that we describe and justify now. First, we repeat that even though our definitions appear to consider static corruptions only, they imply security in a dynamic setting.

Second, we borrow the *selfless anonymity* notion from [11] that departs from the one of [4] in that it does not allow the adversary access to the signing keys of the two signers involved in the query to the challenge oracle. Thus, we cannot grant the adversary access to the secret information of any honest user. This is a natural, mild restriction which, as discussed in the introduction, may lead to significantly more efficient schemes.

Third, our notion of traceability seems different than the notion of traceability of [4]. Indeed, according to our definition an attacker that creates a signature that opens as some honest identity is not considered an attack! We look at this scenario as a framing attack, however, and it is therefore covered under our non-frameability notion, a notion that was not modeled in [4].

Fourth, a detailed comparison of our security notion with the notion of [7] reveals that we do not provide a read and write oracle for the registration table $\mathbf{reg}$. This follows from the fact that we combine the Group Manager with the opening authority. Thereby, the entities cannot be corrupted individually, thus, the adversary has either full access (i.e., when the Group Manager is corrupted) or he does have no access.

Group Signatures with Verifier-Local Revocation. Let us discuss the relation of our scheme and definition with the group signature scheme with verifier-local revocation by Boneh and Shacham [11]. They define a group signature scheme with verifier-local revocation (VLR) as a scheme that has the additional feature of a revocation list. Essentially, VLR-verification of a group signature contains, in addition to the signature verification as described before, a check for each item in the revocation list whether or not it relates to the group signature at hand. If it does, then the signature is deemed invalid.

The scheme and definitions of Boneh and Shacham (1) do not have an open (or tracing) procedure and (2) assume that the group manager is fully trusted. The latter makes sense because if there is no open procedure, it is not possible to falsely blame a user for having produced a specific group signature. However, Boneh and Shacham point out that any VLR scheme has an implicit opening algorithm: one can make a revocation list consisting of only a single user and then run the VLR group signature verification algorithm. Thus, the verification fails only in the case where the user who generated the signature is (the only) entity in the revocation list, which leads to her identification. This shows that we can convert a VLR-scheme into a group signature scheme with an Opener, however, we stress that the obtained scheme does not satisfy non-frameability.

We now point out that the opposite direction also works: for a sub-class of group signature schemes according to our definition one can construct a group signature scheme with verifier local revocation. The subclass is the schemes for which the GOpen algorithm takes as input $(gpk, m, \sigma, \mathbf{reg})$ instead of $(gmsk, m, \sigma, \mathbf{reg})$ as per our definition (i.e., it does not need to make use of the group manager's secret key). We note that the scheme we propose in this paper falls into this sub-class. Now, the idea for obtain-

ing a VLR group signature scheme is as follows. The new key generation consists of the GSetup, PKIJoin, and $(\mathsf{GJoin}_U, \mathsf{GJoin}_M)$ where the group manager runs the users' parts as well and then just hands them their keys. The VLR group signing algorithm is essentially GSign. To revoke user i, the group manager adds $\mathbf{reg}[i]$ to the revocation list. Finally, the VLR-verification consist of GVerify and GOpen, i.e., it accepts a signature if GVerify accepts and if GOpen fails for all entries $\mathbf{reg}[i]$ in the revocation list. The security notions for VLR group signatures, namely selfless anonymity and traceability, follow from our notions of anonymity and traceability for group signatures. We do not give the precise formulation, but we note that a security model for VLR dynamic group signatures follows by combining our dynamic security model above, with the static VLR model from [11]. We also note that VLR group signatures do not provide forward-anonymity: a new revocation list can also be used on old signatures.

4 Our Group Signature Scheme

Overview of Our Scheme. Our group signature scheme is based on two special properties of CL signatures, namely on their re-randomizability and on the fact that the signature "does not leak" the message that it authenticates. Intuitively, a user's group signing key is a CL signature on a random message ξ that only the user knows. To create a group signature for a message m, the user re-randomizes the CL signature and attaches a signature proof of knowledge of ξ on m.

If non-frameability were not a requirement, we could simply let the Group Manager choose ξ, so that he can open group signatures by checking for which of the issued values of ξ the re-randomized CL signature is valid. To obtain non-frameability, however, the Group Manager must not know ξ itself. Hence, in our scheme ξ is generated jointly during an interactive GJoin protocol between the user and the Group Manager. Essentially, this protocol is a two-party computation where the user and the Group Manager jointly generate ξ, a valid CL signature on ξ, and a key derived from ξ that allows the Group Manager to trace signatures, but not to create them.

System Specification. We now present the algorithms that define our efficient group signature scheme. We assume common system parameters for a given security parameter η. Namely, we assume that an asymmetric pairing is fixed, i.e., three groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ of order $q > 2^\eta$ with an efficiently computable map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, together with generators g and $\tilde{g}$ of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Further, two hash functions $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q$, $\mathcal{G} : \{0,1\}^* \to \mathbb{Z}_q$ are defined.

GSetup(1^η): The Group Manager chooses random $\alpha, \beta \leftarrow \mathbb{Z}_q$, and computes $\tilde{x} \leftarrow \tilde{g}^\alpha$ and $\tilde{y} \leftarrow \tilde{g}^\beta$. It then sets the group public key of the scheme to $gpk \leftarrow (\tilde{x}, \tilde{y})$ and the group secret key to $gmsk \leftarrow (\alpha, \beta)$.

PKIJoin$(i, 1^\eta)$: The CA certifies public keys of a digital signature scheme as defined in Section 2. The user generates $(\mathbf{upk}[i], \mathbf{usk}[i]) \leftarrow \mathsf{DSKeyGen}(1^\eta)$ and sends $\mathbf{upk}[i]$ to the CA for certification.

$\mathsf{GJoin} = (\mathsf{GJoin}_M(i, \mathbf{upk}[i], gmsk), \mathsf{GJoin}_U(i, \mathbf{usk}[i], gpk))$**:** When a user i wants to join the group, she must have already run the $\mathsf{PKIJoin}$ algorithm. Then she runs the following protocol with the Group Manager. We assume that this protocol is run over secure channels and, for simplicity, that the parties only run one instance at a time. We also assume that if a verification for a party fails, the party informs the other party about the failure and the protocol is aborted.

1. The Group Manager chooses a random $\kappa \leftarrow \mathbb{Z}_q$, computes $t \leftarrow \mathcal{G}(\kappa)$, and sends t to the user.
2. The user i chooses $\tau \leftarrow \mathbb{Z}_q$, computes $s \leftarrow g^\tau$, $\tilde{r} \leftarrow \tilde{x}^\tau$, $k \leftarrow \hat{e}(g, \tilde{r})$, as well as $\bar{\sigma} \leftarrow \mathsf{DSSign}(\mathbf{usk}[i], k)$, sends $(s, \tilde{r}, \bar{\sigma})$ to the Group Manager and executes $FPK\{(\tau) : s = g^\tau \wedge \tilde{r} = \tilde{x}^\tau\}$ with the Group Manager.
3. The Group Manager uses $\mathsf{DSVerify}(\mathbf{upk}[i], \hat{e}(g, \tilde{r}), \bar{\sigma})$ to verify the signature. If it verifies correctly he computes $z \leftarrow s \cdot g^\kappa$ and $\tilde{w} \leftarrow \tilde{r} \cdot \tilde{x}^\kappa$, stores $(\tilde{w}, \tilde{r}, \kappa, \bar{\sigma})$ in $\mathbf{reg}[i]$, chooses $\rho \leftarrow \mathbb{Z}_q$, computes $a \leftarrow g^\rho$, $b \leftarrow a^\beta$, and $c \leftarrow a^\alpha \cdot z^{\rho\alpha\beta}$, and sends (a, b, c, κ) to the user. In addition, he executes
$$FPK\{(\alpha, \beta, \rho, \gamma) : c = a^\alpha z^\gamma \wedge a = g^\rho \wedge \tilde{x} = \tilde{g}^\alpha \wedge \tilde{y} = \tilde{g}^\beta \wedge 1 = b^\alpha / g^\gamma\}$$
with her, where $\gamma = \rho\alpha\beta$. Note that this proof allows the user to verify that $\alpha, \beta \neq 0$.
4. The user computes $\xi \leftarrow \tau + \kappa \bmod q$, and checks whether $t = \mathcal{G}(\kappa)$. She also verifies $\hat{e}(a, \tilde{y}) = \hat{e}(b, \tilde{g})$ and, if the verification is successful, stores the entry $\mathbf{gsk}[i] \leftarrow (\xi, (a, b, c))$.

Remarks: The value of ω stored in $\mathbf{reg}[i]$ allows the Opener to identify a user within the group signature scheme. In addition, the Opener can provably attribute this ω to $k = \hat{e}(g, \tilde{r})$. Consequently, a group signature can be provably attributed to k. By the unforgeability of the external signature scheme, the signature on k allows to attribute a group signature to a user public key $\mathbf{upk}[i]$. Furthermore, the FPK protocol that the Group Manager and the user execute in Step 3 of the protocol indeed proves that c was computed correctly w.r.t. a, b, $\tilde{x}$, and $\tilde{y}$. To this end, note that because of $\hat{e}(a, \tilde{y}) = \hat{e}(b, \tilde{g})$, we know that $b = a^\beta$ and thus $b = g^{\beta\rho}$. Subsequently, from $1 = b^\alpha / g^\gamma$ we can conclude that $\gamma = \rho\alpha\beta$ and hence that c was computed correctly by the Group Manager.

$\mathsf{GSign}(\mathbf{gsk}[i], m)$**:** Let a user i with signing key $\mathbf{gsk}[i] = (\xi, (a, b, c))$ sign the message m. She first re-randomizes the signature by choosing $\zeta \leftarrow \mathbb{Z}_q$ and computing $d \leftarrow a^\zeta$, $e \leftarrow b^\zeta$, and $f \leftarrow c^\zeta$, and then computes the SPK
$$\Sigma \leftarrow SPK\{(\xi) : \frac{\hat{e}(f, \tilde{g})}{\hat{e}(d, \tilde{x})} = \hat{e}(e, \tilde{x})^\xi\}(m)$$
proving that she knows the "message" for which (d, e, f) is a valid CL-signature. Finally, she outputs $\sigma \leftarrow (d, e, f, \Sigma) \in \mathbb{G}_1^3 \times \mathbb{Z}_q^2$ as the group signature on m.

$\mathsf{GVerify}(gpk, m, \sigma)$**:** To verify a signature $\sigma = (d, e, f, \Sigma)$ on the message m, the verifier first checks that $\hat{e}(d, \tilde{y}) = \hat{e}(e, \tilde{g})$, where $\tilde{g}, \tilde{y}$ are retrieved from gpk. Secondly, the verifier checks that the proof Σ is valid. If either of the checks fail, output 0; otherwise output 1.

$\mathsf{GOpen}(gmsk, m, \sigma, \mathbf{reg})$**:** Given signature $\sigma = (d, e, f, \Sigma)$ on m, the Group Manager verifies the signature using $\mathsf{GVerify}$. Then, for all entries $\mathbf{reg}[i] = (\tilde{w}_i, \tilde{r}_i, \kappa_i, \bar{\sigma}_i)$ he checks whether $\hat{e}(f, \tilde{g}) = \hat{e}(d, \tilde{x}) \cdot \hat{e}(e, \tilde{w}_i)$ holds. For the $\tilde{w}_i$ where the equation holds, the Group Manager retrieves κ_i and $\bar{\sigma}_i$, computes $k_i \leftarrow \hat{e}(g, \tilde{r}_i)$ and the SPK

$$\Pi \leftarrow SPK\{(\tilde{w}_i, \kappa_i) : \frac{\hat{e}(f, \tilde{g})}{\hat{e}(d, \tilde{x})} = \hat{e}(e, \tilde{w}_i) \wedge k_i = \frac{\hat{e}(g, \tilde{w}_i)}{\hat{e}(g, \tilde{x})^{\kappa_i}}\} \ ,$$

and outputs $(i, \pi = (k_i, \bar{\sigma}_i, \Pi))$.
Note that $\phi(\tilde{w}) := (\hat{e}(e, \tilde{w}), \hat{e}(g, \tilde{w}))$ is a group homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_T \times \mathbb{G}_T$ and therefore π can be obtained from applying the Fiat–Shamir transform to the underlying Σ-protocol as discussed earlier. Also note that the opening operation is linear in the number of users in the system, but we consider this reasonable as in most practical applications opening is a rather exceptional operation performed by a resourceful Group Manager.

$\mathsf{GJudge}(gpk, m, \sigma, i, \mathbf{upk}[i], \pi)$**:** The signature of the external signature scheme is verified using the signature verification algorithm $\mathsf{DSVerify}(upk[i], k, \bar{\sigma})$. If the signature verifies, use input gpk, m, $\sigma = (d, e, f, \Sigma)$, and π, to output 1 if algorithm $\mathsf{GVerify}(gpk, m, \sigma) = 1$ and Π is valid. Otherwise output 0.

Remarks. Following the explanations in Section 3.2, we can build a VLR scheme as follows. Transformation of the key generation and the signing algorithm are straightforward. To revoke a user i, the Group Manager publishes the corresponding entry $\tilde{w}_i$ from $\mathbf{reg}[i]$ to the revocation list $\mathbf{rlist}$. Finally, we modify the $\mathsf{GVerify}$ algorithm so that it checks not only that $\hat{e}(d, \tilde{y}) = \hat{e}(e, \tilde{g})$ and the proof Σ is valid, but also whether

$$\hat{e}(f, \tilde{g}) = \hat{e}(d, \tilde{x}) \cdot \hat{e}(e, \tilde{w}_i)$$

for any entry $\tilde{w}_i$ in $\mathbf{rlist}$. If this is the case, it rejects the signature. Thus, the verifier performs what has been a part of the tasks of the Opener in our basic group signature scheme.

5 Security Results

Verifying our scheme's correctness is not hard from its description (and the comments we made there). We now present our results that the scheme satisfies our anonymity, traceability, and non-frameability requirements. Proofs of the following theorems can be found in Appendix B.

Theorem 2. *In the random oracle model the group signature scheme is anonymous under the XDDH and the SDLP assumptions.*

Theorem 3. *In the random oracle model the group signature scheme is traceable under the LRSW assumption.*

Theorem 4. *In the random oracle model the group signature scheme is non-frameable under the SDLP assumption and the unforgeability of the underlying digital signature scheme.*

Security of our scheme as a VLR group signature scheme, in the random oracle model, follows from the above theorems.

6 Comparison With Previous Schemes

We compare efficiency of several schemes with respect to (1) signature size, (2) computational costs of signature generation, and (3) computational costs of signature verification. Let us begin with a detailed discussion of our scheme. The signature algorithm outputs the randomized CL signature (d, e, f), as well as the data needed to verify the signature proof of knowledge Σ. When looking at Σ in more detail, we can set

$$A \leftarrow \frac{\hat{e}(f, \tilde{g})}{\hat{e}(d, \tilde{x})} = \left(\frac{\hat{e}(c, \tilde{g})}{\hat{e}(a, \tilde{x})}\right)^{\zeta} \text{ and } B \leftarrow \hat{e}(e, \tilde{x}) = \hat{e}(b, \tilde{x})^{\zeta},$$

then the SPK is to prove knowledge of ξ such that $A = B^{\xi}$. Applying our description of SPK's obtained from Sigma protocols (see Section 2), the signer needs to compute, for random $\mathsf{rnd} \leftarrow \mathbb{Z}_q$,

$$\mathsf{Comm} \leftarrow B^{\mathsf{rnd}}, \quad \mathsf{Cha} \leftarrow \mathcal{H}(\phi \| A \| \mathsf{Comm} \| m), \quad \mathsf{Rsp} \leftarrow \mathsf{rnd} - \mathsf{Cha} \cdot \xi \pmod{q}.$$

The SPK is then given by the pair $(\mathsf{Cha}, \mathsf{Rsp})$, and hence verification is performed by checking whether $\mathsf{Cha} = \mathcal{H}\left(\phi \| A \| (B^{\mathsf{Rsp}} \cdot A^{\mathsf{Cha}}) \| m\right)$. Thus, a signature consists of three elements in $\mathbb{G}_1$ (d,e, and f) and two elements in $\mathbb{Z}_q$ (Cha and Rsp).

We now turn to computational cost, where by the following type of expression $1 \cdot P^2 + 2 \cdot P + 3 \cdot \mathbb{G}_T^2 + 1 \cdot \mathbb{G}_1$ we denote a cost of one product of two pairing values, two pairings, three multi-exponentiations in $\mathbb{G}_T$ with two terms, and one exponentiation in $\mathbb{G}_1$. Unfortunately, it is very hard to assign conversion factors between the different operations. The reason being that such factors heavily depend, for example, on the elliptic curve underlying a scheme, on the security parameters, or even optimisation of the implementation. Still, to provide a better readability we sort the operations with presumably decrementing complexity and cost.

With the above formulation of the required SPK, the cost for signing would be $2 \cdot \mathbb{G}_T + 3 \cdot \mathbb{G}_1$, since $\hat{e}(a, \tilde{x}), \hat{e}(b, \tilde{x})$ and $\hat{e}(c, \tilde{g})$, can be precomputed. Now we want to optimize the computation of the hash to further shorten the computation time of signing. Keep in mind that doing so, would require corresponding changes to the proofs as well. In our case, the change for optimization actually simplifies the proof. Thus, if we adapt the computation of the challenge to include d, e and f instead of A, i.e., $\mathsf{Cha} \leftarrow \mathcal{H}(\phi \| d \| e \| f \| \mathsf{Comm} \| m)$, the signer does not need to compute A. Consequently, the cost for signing accounts to $1 \cdot \mathbb{G}_T + 3 \cdot \mathbb{G}_1$. Note that this slight change in the computation of the challenge not only benefits the signer but also the verifier of the signature.

Verification in our scheme requires to check whether $\hat{e}(d, \tilde{y}) = \hat{e}(e, \tilde{g})$ holds. This is computed as one product of two pairings, which is more efficient than computing two pairings separately. In addition, verification consists of verifying the SPK, which amounts to $1 \cdot P^2 + 1 \cdot \mathbb{G}_1^2 + 1 \cdot \mathbb{G}_1$, assuming the calculation of the verification value as $\hat{e}(f^c, \tilde{g}) / \hat{e}(d^c e^{s_\xi}, \tilde{x})$. The total cost of verification of a signature consequently amounts

to $2 \cdot P^2 + 1 \cdot \mathbb{G}_1^2 + 1 \cdot \mathbb{G}_1$. Note that we use here the adapted computation of the challenge as described before. We apply similar changes to the calculation of the challenge value for all schemes in our comparison to reduce the required numbered of computations. In addition, we assume pre-computation of pre-known values.

We now compare our scheme with the current best schemes w.r.t. signature length. We only consider pairing-based schemes as RSA-based schemes need much larger groups to attain the same security level. Consequently, we can focus on just a small number of schemes.

- The CL scheme from [17] shares many similarities with our own. The basic security is based on the LRSW and the DDH assumption in $\mathbb{G}_T$. The basic construction is in the case of Type-1 pairings, and combines the CL-signature scheme with a Cramer-Shoup encryption. It is this Cramer-Shoup based component that creates the main divergence from our own scheme. Translating the construction to the Type-2 or Type-3 setting we obtain a more efficient construction based on the LRSW and the XDDH assumption.
- The DP scheme of Delerablée and Pointcheval [26] is based on the XDDH assumption and q-SDH. It is shown to provide full-anonymity under the XDDH assumption w.r.t. the so-called CCA attack, which is achieved by combining two ElGamal encryptions. The scheme is also shown to provide full-traceability under the q-SDH assumption.
- The BBS group signature scheme [10] is similar to the DP scheme [26]. However, it provides full-anonymity under the DLIN assumption only with respect to a so-called CPA attack (i.e., the adversary is not allowed to make any Open oracle queries). As we strive to provide a comparison between systems that have similar security guarantees, we consider a variant of the BBS scheme that we call BBS* and describe in Appendix A..

We summarize the efficiency discussion in Table 1. Note that all schemes provide anonymity w.r.t. the CCA attack, are based on the random oracle model, and provide strong exculpability. As pointed out in the discussion before, they use slightly different underlying assumptions, namely q-SDH or LRSW. A further difference is that our scheme, as opposed to the schemes we compare against, combines Group Manager and Opener into one entity.

Scheme	Size of Sig.		Sign Cost						Verification Cost							
	$\mathbb{G}_1$	$\mathbb{Z}_q$	$\mathbb{G}_T^5$	$\mathbb{G}_T^3$	$\mathbb{G}_T^2$	$\mathbb{G}_T$	$\mathbb{G}_1^2$	$\mathbb{G}_1$	P^2	P	$\mathbb{G}_T^3$	$\mathbb{G}_2^2$	$\mathbb{G}_1^4$	$\mathbb{G}_1^3$	$\mathbb{G}_1^2$	$\mathbb{G}_1$
Ours	3	2				1		3	2						1	1
CL	7	4			1		1	11	2			1		2	2	1
DP	4	5		1			1	6		1	1	1		1	2	
BBS*	4	5	1				3	5	1				1	1	4	

Table 1. Comparison of signature lengths, signature generation costs and signature verification costs.

Table 1 shows that our scheme compares favourably with the other schemes, especially in terms of signature length and the signature generation operation. In particular, it reduces the signature size by almost a factor of two. Comparing verification costs shows all schemes on a similar level. Note that short signatures and small signature computation costs are particularly interesting as there are many scenarios where the group signature has to be generated and communicated by a resource constrained device.

Verification costs. To have an conclusive comparison of the verification costs, it is essential to know the exact cost for each operation. This follows from the fact that there exist various possibilities to verify a SPK proof. For example, verification of the BBS* (as described in Appendix A) SPK proof requires the calculation of the following value

$$\left(\frac{\hat{e}(u,v)}{\hat{e}(T_3,w)}\right)^c \cdot \frac{\hat{e}(T_3,v)^x \hat{e}(h,v)^y}{\hat{e}(e,w)^r \hat{e}(e,v)^\delta}$$

where $u, v, w, h,$ and e are pre-known. We can calculate this value as

$$\frac{\hat{e}(u^c T_3^x h^y e^{-\delta}, v)}{\hat{e}(T_3^c e^r, w)} \qquad \text{or as} \qquad \hat{e}(T_3, v^x w^{-c}) \frac{\hat{e}(u,v)^c \hat{e}(h,v)^y}{\hat{e}(e,w)^r \hat{e}(e,v)^\delta} \ ,$$

where the first computation amounts to $1 \cdot P^2 + 1 \cdot \mathbb{G}_1^4 + 1 \cdot \mathbb{G}_1^2$ operations and the second one accounts for $1 \cdot P + 1 \cdot \mathbb{G}_T^4 + 1 \cdot \mathbb{G}_2^2$ operations. We can see that a direct comparison of those different methods of computing the same value is very hard. Such difficulties, however, mostly arise in the verification equation and make the verification costs less transparent. Still, the numbers in Table 1 show that all compared schemes require similar computation efforts in verification.

VLR-Variant. We end this section by considering our VLR group signature variant as explained in Section 3.2. This version of our scheme has the same signature size as above, namely $3 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{Z}_q$. The signing cost is also the same, namely $1 \cdot \mathbb{G}_T + 3 \cdot \mathbb{G}_1$, but verification includes the opening computations for all revoked users, thus verification requires time

$$|\mathbf{rlist}| \cdot P + 3 \cdot P^2 + 1 \cdot \mathbb{G}_1^2 + 1 \cdot \mathbb{G}_1.$$

As noted previously the BS VLR group signature scheme from [11] requires Type-4 pairings to implement it, as is explained in [32]. Security in their scheme is based on the q-SDH and DLIN assumptions, with DLIN being required to prove selfless-anonymity. A signature requires two elements in $\mathbb{G}_1$ and five elements in $\mathbb{Z}_q$ and this can be computed in $1 \cdot P^2 + 2 \cdot \mathbb{G}_1^2 + 4 \cdot \mathbb{G}_1$ or $1 \cdot \mathbb{G}_T^3 + 1 \cdot \mathbb{G}_1^2 + 3 \cdot \mathbb{G}_1$. Verification costs depend on the size of the revocation list $|\mathbf{rlist}|$, and are given by

$$(1 + |\mathbf{rlist}|) \cdot P + 1 \cdot P^2 + 1 \cdot \mathbb{G}_1^3 + 3 \cdot \mathbb{G}_1^2.$$

Those times split up into signature verification costs ($1 \cdot P^2 + 1 \cdot \mathbb{G}_1^3 + 3 \cdot \mathbb{G}_1^2$) and revocation check costs ($(1 + |\mathbf{rlist}|) \cdot P$). These are slightly faster times than those quoted in [11] as we assume an efficient Type-4 representation of $\mathbb{G}_2$ is used. Note

we have not counted the cost of hashing into $\mathbb{G}_2$ which could be expensive depending on the precise elliptic curve chosen. However, we note that our scheme is significantly more efficient in terms of bandwidth and computational resources than that of [11], even before considering the time needed to hash onto the Type-4 $\mathbb{G}_2$ group in the BS scheme.

7 Acknowledgements

The work described in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II and ICT-2007-216483 PRIMELIFE. The fourth author was supported by a Royal Society Wolfson Merit Award and a grant from Google. All authors would like to thank the referee's of a prior version of this paper.

References

1. Michel Abdalla and Bogdan Warinschi. On the minimal assumptions of group signature schemes. In Javier López, Sihan Qing, and Eiji Okamoto, editors, *ICICS 04*, volume 3269 of *LNCS*, pages 1–13. Springer, October 2004.
2. Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 255–270. Springer, August 2000.
3. Giuseppe Ateniese, Dawn Xiaodong Song, and Gene Tsudik. Quasi-efficient revocation in group signatures. In Matt Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 183–197. Springer, March 2002.
4. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, May 2003.
5. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 390–399. ACM Press, October / November 2006.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
7. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153. Springer, February 2005.
8. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 1–12. Springer, August 1998.
9. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, May 2004.
10. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, August 2004.

11. Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 04*, pages 168–177. ACM Press, October 2004.
12. Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 427–444. Springer, May / June 2006.
13. Xavier Boyen and Brent Waters. Full-domain subgroup hiding and constant-size group signatures. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 1–15. Springer, April 2007.
14. Ernie Brickell, Liqun Chen, and Jiangtao Li. A new direct anonymous attestation scheme from bilinear maps. In *Trusted Computing - Challenges and Applications – TRUST 2008*, volume 4968 of *LNCS*, pages 1–20. Springer-Verlag, 2008.
15. Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the portability of generalized schnorr proofs. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 425–442. Springer, April 2009.
16. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, August 2002.
17. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, August 2004.
18. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, August 1997.
19. David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, April 1991.
20. Liqun Chen, Michael Cheng, and Nigel P. Smart. Identity-based key agreement protocols from pairings. *International Journal of Information Security*, 6:213–241, 2007.
21. Liqun Chen, Paul Morrissey, and Nigel P. Smart. Pairings in trusted computing (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 1–17. Springer, September 2008.
22. Liqun Chen, Paul Morrissey, and Nigel P. Smart. DAA: Fixing the pairing based protocols. Cryptology ePrint Archive, Report 2009/198, 2009. `http://eprint.iacr.org/`.
23. Ronald Cramer, Ivan Damgård, and Philip D. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000*, volume 1751 of *LNCS*, pages 354–372. Springer, January 2000.
24. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
25. Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, May 2000.
26. Cécile Delerablée and David Pointcheval. Dynamic fully anonymous short group signatures. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06*, volume 4341 of *LNCS*, pages 193–210. Springer, September 2006.
27. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, August 1987.
28. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008.

29. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.
30. Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *SAC 1999*, volume 1758 of *LNCS*, pages 184–199. Springer, August 2000.
31. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.
32. Hovav Shacham. *New Paradigms in Signature Schemes*. PhD thesis, Stanford University, 2005.
33. Hovav Shacham. A Cramer-Shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074, 2007. `http://eprint.iacr.org/`.
34. Gene Tsudik and Shouhuai Xu. Accumulating composites and improved group signing. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 269–286. Springer, November / December 2003.

A Sketch of BBS*

This variant of the BBS group signature scheme is based on remarks by Boneh et al. [10] (general scheme and non-frameability) and [33] (CCA anonymity). In particular, the variant we consider attains exculpability by an interactive protocol between Group Manager and user for the joint computation of a triple (A_i, x_i, y_i) such that $A_i^{x_i+\gamma} h^{y_i} = u$. Here y_i is secret to the user, γ is the Group Manager's secret, and $u, h \in \mathbb{G}_1$ are public parameters. Given that all schemes we compare are secure under XDDH, we employ standard Cramer-Shoup encryption [24] instead of the linear Cramer-Shoup encryption proposed by Shacham [33] which is secure even if XDDH does not hold.

In more detail, the setup and key generation algorithms produce $u, v \leftarrow \mathbb{G}_1$ and $c \leftarrow u^{\chi_1} v^{\chi_2}, d \leftarrow u^{\mu_1} v^{\mu_2}$, as well as $e \leftarrow v^{\iota}$. As a result of the join protocol, each user gets a tuple (A_i, x_i, y_i) fulfilling $A_i^{x_i+\gamma} h^{y_i} = u$ and the Group Manager uses his secret value γ to compute $w \leftarrow v^{\gamma}$. The group public key consists of (u, v, c, d, e, h, w) and the secret key of the Opener contains $(\chi_1, \chi_2, \mu_1, \mu_2)$.

To sign a message, user i chooses $r \leftarrow \mathbb{Z}_q$, $\delta \leftarrow r \cdot x_i$, and computes $T_1 \leftarrow u^r$, $T_2 \leftarrow v^r, T_3 \leftarrow e^r A_i, T_4 \leftarrow c^r d^{r\mathcal{H}(T_1,T_2,T_3)}$. Moreover, she computes the proof

$$\Sigma \leftarrow \mathrm{SPK}\{(r, x_i, \delta, y_i) : T_1 = u^r \wedge T_2 = v^r \wedge T_4 = c^r d^{r\mathcal{H}(T_1,T_2,T_3)} \wedge 1 = T_1^{x_i} u^{-\delta} \wedge \frac{\hat{e}(u,v)}{\hat{e}(T_3,w)} = \frac{\hat{e}(T_3,v)^{x_i} \hat{e}(h,v)^{y_i}}{\hat{e}(e,w)^r \hat{e}(e,v)^{\delta}}\} \ ,$$

and outputs the signature $\sigma \leftarrow (T_1, T_2, T_3, T_4, \Sigma)$.

The verification of a signature consists of checking the validity of the proof Σ. Opening a signature can be performed by the Opener using his secret key to decrypt the Cramer-Shoup encryption of the value A_i.

B Security Proofs

B.1 Proof of Theorem 2

We show that a simulator $\mathcal{B}$, given an adversary $\mathcal{A}$ having a non-negligible advantage in the anonymity game $\mathbf{Adv}^{\text{anon}}_{\mathsf{GS},A}(\eta) \geq \epsilon$, can solve a DDH problem in $\mathbb{G}_1$. We denote the DDH challenge given to $\mathcal{B}$ as $(g_0, g_1, g_2, g_3) = (g, g^\mu, g^\nu, g^\omega)$, where $\mathcal{B}$ will output a guess δ' indicating whether $\omega \leftarrow \mu\nu$, i.e., $\delta = 1$, or $\omega \leftarrow \mathbb{Z}_q$, i.e., $\delta = 0$.

The proof idea is to let honest users sign with signatures on a secret value $\xi_i = \mu r_i$, where $r_i \leftarrow \mathbb{Z}_q$. More concretely, $\mathcal{B}$ generates $(gpk, gmsk)$ in the normal way and creates a tuple $(a_i, b_i, c_i) = (g^{\rho_i}, a_i^\beta, a_i^{\alpha+\alpha\beta\mu r_i})$ for each honest user i. When $\mathcal{A}$ asks the simulator for a challenge signature, $\mathcal{B}$ uses its knowledge of $gmsk$ and $\mathbf{gsk}[i], \forall i \in \mathcal{HU}$ to create a signature of the form $(d^*, e^*, f^*) = (g^\nu, g^{\nu\beta}, g^{\nu\alpha}(g^\omega)^{\alpha\beta r_i})$. This constitutes a valid group signature for a user with $\xi_i = \mu r_i$ and randomization parameter ν, assuming $\mathcal{B}$ has been given a DDH tuple with $\omega = \mu\nu$. Consequently, $\mathcal{A}$ only has an advantage in solving the anonymity game, if the DDH challenge was a DDH tuple. Otherwise, $\mathcal{A}$ does not gain any information from the given signature.

In more detail, given the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, $\mathcal{B}$ retrieves the sets $\mathcal{HU}, \mathcal{DU} \subseteq \{1, \ldots, n\}$ from $\mathcal{A}$. He uses $g \leftarrow g_0$, selects a generator $\tilde{g} \in \mathbb{G}_2$ as well as $\alpha, \beta \leftarrow \mathbb{Z}_q$, and computes $\tilde{x} \leftarrow \tilde{g}^\alpha$ and $\tilde{y} \leftarrow \tilde{g}^\beta$. Then, he sets the secret key of the group manager to $gmsk \leftarrow (\alpha, \beta)$ and the group public key $gpk \leftarrow (g, \tilde{g}, \tilde{x}, \tilde{y})$, which he supplies to $\mathcal{A}$.

Then, the simulator $\mathcal{B}$ generates a PKI key pair $(\mathbf{usk}[i], \mathbf{upk}[i]) \leftarrow \mathsf{PKIJoin}(i, 1^\eta)$ for all honest users $i \in \mathcal{HU}$. $\mathcal{B}$ chooses $\rho_i, r_i \leftarrow \mathbb{Z}_q$ and calculates their group signing key as $(a_i, b_i, c_i) = (g_0^{\rho_i}, a^\beta, a^\alpha g_1^{\rho_i \alpha\beta r_i})$. Note, that this tuple has the same distribution as the one specified by the protocol in Section 4. Moreover, $\mathcal{B}$ chooses a random $k_i \in \mathbb{G}_T$ and computes the signature $\bar{\sigma}_i \leftarrow \mathsf{DSSign}(\mathbf{usk}[i], k_i)$, which is distributed as in the proposed scheme. He stores $(a_i, b_i, c_i, r_i, k_i, \bar{\sigma}_i)$ in $\mathbf{gsk}[i]$.

Then $\mathcal{B}$ runs $\mathcal{A}$ and simulates the oracle queries as follows:

- $\mathcal{G}(\kappa)$: $\mathcal{B}$ maintains a list $L_\mathcal{G}$ of each previous random oracle response t to the correpsonding query κ. It returns t to each query κ, assigning a fresh random value $t \leftarrow \{0,1\}^\ell$ if (κ, t) is undefined.
- $\mathcal{H}(S)$: $\mathcal{B}$ maintains a list $L_\mathcal{H}$ of previous random oracle responses storing (S, Cha). It returns Cha to each query S, assigning a fresh random value $\mathsf{Cha} \leftarrow \{0,1\}^\ell$ if (S, Cha) is undefined.
- $\mathsf{SetUPK}(i, upk)$: $\mathcal{B}$ executes the role of the CA and publishes $\mathbf{upk}[i] \leftarrow upk$. Note that everyone can get an authentic copy of $\mathbf{upk}$ (which can be achieved by letting the simulator create a signature on the user public key using the CA's public key).
- $\mathsf{GJoin}_{DM}(i)$: For dishonest users $i \in \mathcal{DU}$, $\mathcal{B}$ simulates the Group Manager's side of the protocol as prescribed in the scheme in Section 4 using the knowledge of $gmsk$. The simulator stores the local output $(\tilde{w}, \tilde{r}, \kappa, \bar{\sigma})$ in $\mathbf{reg}[i]$.
- $\mathsf{GSign}(i, m)$: Given user $i \in \mathcal{HU}$ and a message m from $\mathcal{A}$, $\mathcal{B}$ retrieves $\mathbf{gsk}[i] = (a_i, b_i, c_i, r_i, k_i, \bar{\sigma}_i)$.
  It chooses $\zeta \leftarrow \mathbb{Z}_q$ to re-randomize the group signing key to obtain $(d, e, f) = (a_i^\zeta, b_i^\zeta, c_i^\zeta)$. By programming the random oracle $\mathcal{H}(\cdot)$, $\mathcal{B}$ can simulate the signature of knowledge Σ and returns $\sigma = (a, b, c, \Sigma)$ to $\mathcal{A}$. He adds (i, m, σ) to a list $\mathbf{sgn}$.

– $\mathsf{GOpen}(m, \sigma)$: As the signature can originate from an honest or a dishonest user, the simulator distinguishes two cases.
  1. If the signature stems from a dishonest user, it can be opened and the proof π created using the information from $\mathbf{reg}$ as in the real scheme.
  2. In case the signature was created on the behalf of an honest user, then either σ is a previous output of the $\mathsf{GSign}(\cdot, \cdot)$ oracle produced by $\mathcal{B}$, or it is a forgery produced by $\mathcal{A}$. The latter case is excluded by the non-frameability property of Theorem 4. In the former case, $\mathcal{B}$ can simply look up the corresponding user i from the list $\mathbf{sgn}$ using the message m and σ, and look up the corresponding tuple $\mathbf{gsk}[i] = (a_i, b_i, c_i, r_i, k_i, \bar{\sigma}_i)$. Then, by programming the random oracle $\mathcal{H}(\cdot)$ $\mathcal{B}$ simulates a signature of knowledge

$$\Pi \leftarrow SPK\{(\tilde{w}_i, \kappa) : \frac{\hat{e}(f, \tilde{g})}{\hat{e}(d, \tilde{x})} = \hat{e}(e, \tilde{w}_i) \wedge k_i = \frac{\hat{e}(g, \tilde{w}_i)}{\hat{e}(g, \tilde{x})^{\kappa}}\} \ .$$

  He sends $(i, \pi = (k_i, \bar{\sigma}, \Pi))$ to $\mathcal{A}$.
– $\mathsf{Ch}(\cdot, i_0, i_1, m)$: First, $\mathcal{B}$ chooses $b \leftarrow \{0, 1\}$ and looks up the group signing key $\mathbf{gsk}[i_b] = (a_{i_b}, b_{i_b}, c_{i_b}, r_{i_b}, k_{i_b}, \bar{\sigma}_{i_b})$. Secondly, he constructs

$$(d^*, e^*, f^*) \leftarrow (g_2, g_2^{\beta}, g_2^{\alpha} g_3^{\alpha\beta r_{i_b}}) \ .$$

  Assuming that the given DDH challenge is a DDH tuple, i.e., $\delta = 1$, implies $f^* = g_0^{\alpha\nu + \alpha\beta\nu\mu r_{i_b}}$, which has the same distribution compared to a real signature tuple. If $\delta = 0$, f^* is uniformly distributed in $\mathbb{G}_1$, independent of the choice of b. Finally, $\mathcal{B}$ simulates the signature of knowledge Σ by programming the random oracle $\mathcal{H}(\cdot)$.

At the end of its execution, the adversary $\mathcal{A}$ will output a guess b'. The simulator $\mathcal{B}$ outputs $\delta' = 1$ if the adversary $\mathcal{A}$ output $\gamma' = \gamma$, and outputs $\delta' = 0$ otherwise. We calculate the advantage of $\mathcal{B}$ in solving the DDH challenge as

$$\mathbf{Adv}_{\mathcal{B}}^{XDDH} = \Pr[\delta' = 1 | \delta = 1] - \Pr[\delta' = 1 | \delta = 0].$$

When $\delta = 0$, f^* is a uniformly distributed value in $\mathbb{G}_1$ independent of b, so that $\mathcal{A}$ outputs $b' = b$ with probability $\Pr[b' = b | \delta = 0] = \frac{1}{2}$. As $\mathcal{B}$ guesses $\delta' = 1$ when $b' = b$, we get:

$$\Pr[\delta' = 1 | \delta = 0] = \frac{1}{2} \ .$$

If $\delta = 1$ then the challenge signature is identically distributed as in a real attack scenario. Due to $\mathcal{B}$'s choice of δ' we see that $\Pr[\delta' = 1 | \delta = 1] = \Pr[b' = b | \delta = 1]$, which stands for the adversary $\mathcal{A}$ winning the anonymity game. Thus,

$$\begin{aligned} \Pr[b' = b | \delta = 1] &= \frac{1}{2} \left(\Pr[\mathbf{Exp}_{\mathsf{GS},A}^{\text{anon-1}}(\eta) = 1] + \Pr[\mathbf{Exp}_{\mathsf{GS},A}^{\text{anon-0}}(\eta) = 0] \right) \\ &= \frac{\mathbf{Adv}_{\mathsf{GS},A}^{\text{anon}}(\eta) + 1}{2}, \end{aligned}$$

lets us conclude that:

$$\mathbf{Adv}_{\mathcal{B}}^{XDDH} \geq \frac{\epsilon}{2} \ .$$

B.2 Proof of Theorem 3

We prove that given any adversary $\mathcal{A}$ winning the traceability game, one can construct an adversary $\mathcal{B}$ breaking the existential unforgeability of CL signatures. The theorem then follows from the security proof of CL signatures [17].

In a first step, we transform adversary $\mathcal{A}$ into an adversary $\mathcal{A}'$ to which the general forking lemma due to Bellare and Neven [5] can be applied; we recall the lemma in Appendix C. The lemma will then yield a forking algorithm $\mathcal{F}_{\mathcal{A}'}$ that produces two different but related untraceable group signatures, based on which $\mathcal{B}$ can compute a forgery for the CL signature scheme.

Given adversary $\mathcal{A}$, consider the following algorithm $\mathcal{A}'$. On input $(\tilde{x}, \tilde{y})$ and random tape R, it runs $\mathcal{A}$ on input $gpk = (\tilde{x}, \tilde{y})$ and random tape R' derived from R, simulating its oracle queries as follows while maintaining a counter ctr and lists $L_{\mathcal{G}}$, $L_{\mathcal{H}}$, and an associative array $\mathbf{reg}$:

- $\mathcal{G}(\kappa)$: $\mathcal{A}'$ looks up a tuple (κ, t) in the list $L_{\mathcal{G}}$ and returns t; if no such tuple is found, it chooses a random value $t \leftarrow \{0,1\}^{\ell}$ and adds (κ, t) to $L_{\mathcal{G}}$.
- $\mathcal{H}(S)$: $\mathcal{A}'$ looks up a tuple (S, j, Cha) in the list $L_{\mathcal{H}}$ and returns Cha. If no such tuple is found, it increases ctr, chooses a random value $\mathsf{Cha} \leftarrow \{0,1\}^{\ell}$ and adds (S, ctr, Cha) to $L_{\mathcal{H}}$.
- $\mathsf{SetUPK}(i, upk)$: $\mathcal{A}'$ sets $\mathbf{upk}[i] \leftarrow upk$ as the certified public key associated to user i.
- $\mathsf{GJoin}_{DM}(i)$: $\mathcal{A}'$ chooses $\kappa \leftarrow \mathbb{Z}_q$, computes $t \leftarrow \mathcal{G}(\kappa)$, and sends t to $\mathcal{A}$. After receiving $(s, \tilde{r}, \bar{\sigma})$, $\mathcal{A}'$ rewinds $\mathcal{A}$ to extract τ from the $FPK\{(\tau)\}$; when it fails, which only happens with probability of the knowledge error $1/q$, $\mathcal{A}'$ halts with output $(0, \varepsilon)$. Otherwise, it computes $\xi \leftarrow \tau + \kappa \bmod q$ and queries its signing oracle for a CL signature (a, b, c) on message ξ. It then sends (a, b, c, κ) to $\mathcal{A}$ and uses the zero-knowledge simulator to simulate $FPK\{(\alpha, \beta, \rho, \gamma)\}$, which it can do without any probability loss due to the perfect zero-knowledge property. Finally, it saves the tuple $(\tilde{w} \leftarrow \tilde{x}^{\xi}, \tilde{r}, \kappa, \bar{\sigma})$ in $\mathbf{reg}[i]$.
- $\mathsf{GOpen}(m, \sigma)$: If $\mathsf{GVerify}(gpk, m, \sigma) = 0$ then $\mathcal{A}'$ returns $\perp$. Else, it parses σ as (d, e, f, Σ) and looks for a tuple $(\tilde{w}, \tilde{r}, \kappa, \bar{\sigma}) \in \mathbf{join}$ such that $\hat{e}(f, \tilde{g}) = \hat{e}(d, \tilde{x}) \cdot \hat{e}(e, \tilde{w})$. If such a tuple is found, it constructs a proof π using values $\tilde{w}$, $\tilde{r}$ and κ as in the real GOpen algorithm and returns $(i, (k, \bar{\sigma}, \pi))$.

When $\mathcal{A}$ outputs its forgery $(m, \sigma = (d, e, f, (\mathsf{Cha}, \mathsf{Rsp}))$ we distinguish two cases depending on the validity of a forgery. If the forgery is invalid, meaning that $\mathsf{GVerify}(gpk, m, \sigma) = 0$ or σ can be opened by the procedure described in the GOpen oracle using one of the $\tilde{w}$ values in $\mathbf{join}$, then $\mathcal{A}'$ halts with output $(0, \varepsilon)$. Otherwise, it looks up the index j so that $(S, j, \mathsf{Cha}) \in L_{\mathcal{H}}$ for $S = A \| B \| C \| m$ where $A = \hat{e}(f, \tilde{g}) / \hat{e}(d, \tilde{x})$, $B = \hat{e}(e, \tilde{x})$, and $C = A^{\mathsf{Cha}} B^{\mathsf{Rsp}}$. Such a tuple must exist, since at the very latest $\mathcal{A}'$ would have forced its creation during the final verification $\mathsf{GVerify}(gpk, m, \sigma)$. We call the j-th $\mathcal{H}(\cdot)$ query made by $\mathcal{A}$ the "crucial" hash query. $\mathcal{A}'$ halts with output (j, σ).

Consider the general forking lemma with algorithm $\mathcal{A}'$ and an input generator $\mathcal{IG}$ that outputs gpk. If $\mathcal{A}$ wins the traceability game with probability ϵ, i.e., $\mathbf{Adv}^{\mathrm{trace}}_{\mathsf{GS},\mathcal{A}}(\eta) = \epsilon$, then the probability acc that $\mathcal{A}'$ outputs (j, σ) with $j \geq 1$ is

$$acc \;\geq\; \epsilon - n/q \,,$$

where the latter term is due to premature halting because of an extraction failure during any of the GJoin protocols.

By applying the general forking lemma, we obtain an algorithm $\mathcal{F}_{\mathcal{A}'}$ that with probability

$$frk \;\geq\; \frac{acc^2}{q_{\mathrm{H}}} - \frac{1}{q} \;\geq\; \frac{\epsilon^2}{q_{\mathrm{H}}} - \frac{3n^2+1}{q} \tag{1}$$

outputs a tuple $(1, \sigma_1, \sigma_2)$, where q_{H} is (at most) the number of $\mathcal{H}(\cdot)$ queries made by $\mathcal{A}$.

Based on this algorithm $\mathcal{F}_{\mathcal{A}'}$, consider algorithm $\mathcal{B}$ that forges CL signatures by running $\mathcal{F}_{\mathcal{A}'}$ to obtain two signatures σ_1 and σ_2 where $\sigma_1 = (d_1, e_1, f_1, (\mathsf{Cha}_1, \mathsf{Rsp}_1))$ and $\sigma_2 = (d_2, e_2, f_2, (\mathsf{Cha}_2, \mathsf{Rsp}_2))$. Let

$$\begin{array}{lll} A_1 = \hat{e}(f_1, \tilde{g})/\hat{e}(d_1, \tilde{x}) & B_1 = \hat{e}(e_1, \tilde{x}) & C_1 = A_1^{\mathsf{Cha}_1} B_1^{\mathsf{Rsp}_1} \\ A_2 = \hat{e}(f_2, \tilde{g})/\hat{e}(d_2, \tilde{x}) & B_2 = \hat{e}(e_2, \tilde{x}) & C_2 = A_2^{\mathsf{Cha}_2} B_2^{\mathsf{Rsp}_2} \; . \end{array}$$

Since the two executions of $\mathcal{A}$ are identical in inputs, random tape, and oracle responses up to the point where the "crucial" hash queries are made, the arguments of these hash queries must be identical too, so that $A_1 = A_2$, $B_1 = B_2$, and $C_1 = C_2$. Because $B_1 = B_2$ we have that $e_1 = e_2$. Since both signatures are valid we have that $\hat{e}(d_1, \tilde{y}) = \hat{e}(e_1, \tilde{g})$ and $\hat{e}(d_2, \tilde{y}) = \hat{e}(e_2, \tilde{g})$, so that $d_1 = d_2$. Finally, because $A_1 = A_2$ we also have that $f_1 = f_2$. The forking algorithm however guarantees us that the responses to these queries Cha_1 and Cha_2 are different and smaller than q, so that $\mathsf{Cha}_1 - \mathsf{Cha}_2 \neq 0 \bmod q$. By putting the equations for C_1 and C_2 together one can see that $\xi = (\mathsf{Rsp}_2 - \mathsf{Rsp}_1)/(\mathsf{Cha}_1 - \mathsf{Cha}_2) \bmod q$ satisfies the equation $\hat{e}(f_1, \tilde{g})/\hat{e}(d_1, \tilde{x}) = \hat{e}(e_1, \tilde{x})^{\xi}$, which is the second verification equation of CL signatures. The validity of the group signature σ_1 ensures that the first CL verification equation is also satisfied, so that (d_1, e_1, f_1) is a valid CL signature on message ξ. Moreover, ξ does not occur in a tuple of $\mathbf{reg}$, because in that case the opening of σ_1 at the end of the execution of $\mathcal{A}'$ would have succeeded. Since the only messages ξ for which $\mathcal{B}$ previously queried CL signatures are those occurring in $\mathbf{reg}$, $\mathcal{B}$ can output $(\xi, (d_1, e_1, f_1))$ as its own forgery; its overall probability of doing so is at least the probability frk depicted in Equation (1).

B.3 Proof of Theorem 4

The goal of the adversary $\mathcal{A}$ in the non-frameability game is to create a group signature σ on a message m together with a valid proof π that attributes σ to an honest user i even though σ was never output by the GSign oracle on inputs i, m. We distinguish between two types of attacks. In the first type, the value for k in $\pi = (k, \bar{\sigma}, \Pi)$ is different from the value that user i signed during the join protocol. It is easy to see that this type of attack is impossible by the unforgeability of the underlying signing algorithm DSSign, as $\bar{\sigma}$ is a valid signature under $\mathbf{upk}[i]$ that was never signed by user i. We now focus on the second type of attacks, where the value for k in π is the same as signed by user i when joining.

We construct a simulator $\mathcal{B}$ that solves a given SDLP problem $(g_1, \tilde{g}_1) = (g^{\mu}, \tilde{g}^{\mu})$. Note, that w.l.o.g. we assume that the bases in the SDLP problem match the bases that

the group signature scheme is using, as an SDLP problem using different bases can be transformed into the problem denoted before. The simulator makes use of an adversary $\mathcal{A}$ having a non-negligible advantage in the non-frameability game $\mathbf{Adv}^{\mathrm{nf}}_{\mathsf{GS},A}(\eta) \geq \epsilon$. The simulator's output is ε, if he was not able to solve the DL problem, or μ being the solution to the problem.

The proof idea is to create an algorithm $\mathcal{A}'$ to which we will apply the general forking lemma. Through the lemma we attain an algorithm $\mathcal{F}_{\mathcal{A}'}$. This algorithm will output two signatures (σ_0, σ_1) that are related such that the simulator can extract the "message" that they have been issued upon. Assuming that $\mathcal{B}$ manages to construct those messages dependent on μ will allow him to solve the SDLP problem.

In more detail, the algorithm $\mathcal{A}'$ gets gpk and a random tape R. It derives R' from R and runs $\mathcal{A}$ on input (gpk, R'). Algorithm $\mathcal{A}'$ maintains a counter ctr and lists $L_{\mathcal{G}}$, $L_{\mathcal{H}}$.

- $\mathcal{G}(\kappa)$: $\mathcal{A}'$ looks up a tuple (κ, t) in the list $L_{\mathcal{G}}$ and returns t; if no such tuple is found, it chooses a random value $t \leftarrow \{0,1\}^{\ell}$ and adds (κ, t) to $L_{\mathcal{G}}$.
- $\mathcal{H}(S)$: $\mathcal{A}'$ looks up (S, j, Cha) in the list $L_{\mathcal{H}}$ and returns Cha. If no such tuple is found, it increases ctr, chooses a random value $\mathsf{Cha} \leftarrow \mathbb{Z}_q$ and adds (S, ctr, Cha) to $L_{\mathcal{H}}$.
- $\mathsf{SetUPK}(i, upk)$: $\mathcal{A}'$ sets $\mathbf{upk}[i] \leftarrow upk$ as the certified public key associated to user i.
- $\mathsf{GJoin}_{UD}(i)$: Given an honest user $i \in \mathcal{HU}$, $\mathcal{A}'$ extracts the value of κ from the random oracle $\mathcal{G}(\cdot)$ by looking for a pair $(\kappa, t) \in L_{\mathcal{G}}$. (For large enough values of ℓ, exactly one such pair must exist, because otherwise $\mathcal{A}$ must either have created a collision on $\mathcal{G}$, or predicted an output of $\mathcal{G}$ before querying it.) Then he chooses $r_i \leftarrow \mathbb{Z}_q$ and computes $s \leftarrow g_1^{r_i}/(g^{\kappa})$ as well as $\tilde{r} \leftarrow \tilde{g}_1^{r_i}/(\tilde{g}^{\kappa})$. Then he computes the signature $\bar{\sigma} \leftarrow \hat{e}(g, \tilde{r})$ and sends $(s, \tilde{r}, \bar{\sigma})$ to $\mathcal{A}$. The algorithm simulates the proof of knowledge without any probability loss due to the perfect zero-knowledge property. $\mathcal{A}$ will supply the value of κ and will prove the correct computation of the values (a, b, c). Through the verification of this proof, $\mathcal{A}'$ is assured that the received signature constitutes a signature on $\xi = \mu r_i$. $\mathcal{A}'$ stores (ξ, a, b, c) in $\mathbf{gsk}[i]$.
- $\mathsf{GSign}(i, m)$: Given user $i \in \mathcal{HU}$ and a message m from $\mathcal{A}$, $\mathcal{A}'$ retrieves the signing key from $\mathbf{gsk}$. If $\mathbf{gsk}[i] = \perp$ does not exists, then $\mathcal{A}'$ return $\perp$. Otherwise, it computes $\sigma \leftarrow \mathsf{GSign}(\mathbf{gsk}[i], m)$, adds (m, σ) to the list $\mathbf{sgn}$, and returns σ.

Algorithm $\mathcal{A}$ will output the forged signature $\sigma = (d, e, f, (\mathsf{Cha}, \mathsf{Rsp}))$ on a message m, a proof π and an i indicating for which user $\mathcal{A}$ forged the signature. If any of $\mathsf{GVerify}(gpk, m, \sigma) = 0$, $i \notin \mathcal{HU}$, $(m, (d, e, f)) \in \mathbf{sgn}$, or $\mathsf{GJudge}(m, \sigma, i, \mathbf{upk}[i], \pi) = 0$ fails, then $\mathcal{A}'$ will halt and output $(0, \varepsilon)$.

Otherwise, $\mathcal{A}'$ looks up the tuple $(A \| B \| C \| m, j, \mathsf{Cha})$ where $A = \hat{e}(f, \tilde{g})/\hat{e}(d, \tilde{x})$, $B = \hat{e}(e_1, \tilde{x})$, and $C = A^{\mathsf{Cha}} B^{\mathsf{Rsp}}$. Such a tuple exists, since it would be created at the latest during the final execution of $\mathsf{GVerify}$ by $\mathcal{A}'$. We call the j-th hash query the "crucial" hash query. $\mathcal{A}'$ outputs (j, σ).

We use the forking lemma as given in Section C with the algorithm $\mathcal{A}'$ and input generator $\mathcal{IG}$ outputting gpk. Provided $\mathcal{A}$ wins the non-frameability game with probability ϵ, makes the probability of $\mathcal{A}'$ finding (j, σ) with $j \geq 1$ become $acc = \epsilon$. Through

the general forking lemma we get an algorithm $\mathcal{F}_{\mathcal{A}'}$, which succeeds with probability

$$frk \geq \frac{acc^2}{q_{\mathcal{H}}} - \frac{1}{q} = \frac{\epsilon^2}{q_{\mathcal{H}}} - \frac{1}{q}$$

to produce a tuple $(1, \sigma_0, \sigma_1)$. The number of queries of $\mathcal{A}$ has the upper bound $q_{\mathcal{H}}$.

Running the algorithm $\mathcal{F}_{\mathcal{A}'}$, the simulator $\mathcal{B}$ obtains two signatures σ_1 and σ_2, where $\sigma_1 = (d_1, e_1, f_1, (\mathsf{Cha}_1, \mathsf{Rsp}_1))$ and $\sigma_2 = (d_2, e_2, f_2, (\mathsf{Cha}_2, \mathsf{Rsp}_2))$. By a similar reasoning as in the traceability proof in Appendix B.2, we have that $(d_1, e_1, f_1) = (d_2, e_2, f_2)$, $\mathsf{Cha}_1 \neq \mathsf{Cha}_2$, and $\mathsf{Cha}_1 \neq \mathsf{Cha}_2 \bmod q$. The simulator $\mathcal{B}$ computes $\xi = (\mathsf{Rsp}_2 - \mathsf{Rsp}_1)/(\mathsf{Cha}_1 - \mathsf{Cha}_2) \bmod q$, which satisfies the equation $\hat{e}(f_1, \tilde{g})/\hat{e}(d_1, \tilde{x}) = \hat{e}(e_1, \tilde{x})^{\xi}$. Due to our construction of the signatures, we have that $\xi = \mu r_i$ holds, where r_i can be looked up in $\mathbf{rand}$. We therefore have that $g^{\xi/r_i} = g_1$ and $\tilde{g}^{\xi/r_i} = \tilde{g}_1$, so that ξ/r_i is a solution to the SDLP problem. The simulator obtains this value with probability at least frk.

C Forking Lemma

We recall here the general forking lemma due to Bellare and Neven [5]. In the following, think of x as a public key, q_{H} as the number of queries to a random oracle, and $h_1, \ldots, h_{q_{\mathrm{H}}}$ as the responses.

Lemma 1. *Let $\mathcal{A}$ be a randomized algorithm that on input $x, h_1, \ldots, h_{q_{\mathrm{H}}}$ returns a pair $(j, \sigma) \in \{0, \ldots, q_{\mathrm{H}}\} \times \{0,1\}^*$. Let $\mathcal{IG}$ be a randomized algorithm called the input generator. The accepting probability of $\mathcal{A}$, denoted acc, is defined as the probability that $j \geq 1$ in the experiment*

$$x \leftarrow \mathcal{IG} \ ; \ h_1, \ldots, h_{q_{\mathrm{H}}} \leftarrow \{0,1\}^{\ell} \ ; \ (j, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_{q_{\mathrm{H}}}) \ .$$

The forking algorithm *$\mathcal{F}_{\mathcal{A}}$ associated to $\mathcal{A}$ is the randomized algorithm that on input x proceeds as follows:*

Algorithm $\mathcal{F}_{\mathcal{A}}(x)$
Pick random coins R for $\mathcal{A}$
$h_1, \ldots, h_{q_{\mathrm{H}}} \leftarrow \{0,1\}^*$
$(j, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_{q_{\mathrm{H}}}; R)$
If $j = 0$ then return $(0, \varepsilon)$
$h'_j, \ldots, h'_{q_{\mathrm{H}}} \leftarrow \{0,1\}^{\ell}$
$(j', \sigma') \leftarrow \mathcal{A}(x, h_1, \ldots, h_{j-1}, h'_j, \ldots, h'_{q_{\mathrm{H}}}; R)$
If ($j = j'$ and $h_j \neq h'_j$) then return $(1, \sigma, \sigma')$
Else return $(0, \varepsilon, \varepsilon)$.

Let

$$frk \ = \ \Pr\left[b = 1 \ : \ x \leftarrow \mathcal{IG} \ ; \ (b, \sigma, \sigma') \leftarrow \mathcal{F}_{\mathcal{A}(x)}\right] \ .$$

Then

$$frk \ \geq \ \frac{acc^2}{q_{\mathrm{H}}} - \frac{1}{2^{\ell}} \ .$$