Collected Papers where Every Theorem Is Filled with Grief

Hovav Shacham

December 2005

Contents

Chapter 1

Introduction

In a digital signature scheme, Alice uses her private key to *sign* a message of her choice. This procedure creates a *signature*, a short string that binds Alice to the message and the message to her. Anyone who has Alice's public key, the signature, and the message can verify that the signature is *valid*, i.e., was produced by Alice on the message at hand. No one but Alice can generate a signature on any message that verifies as valid under Alice's public key.

Digital signatures thus provide authenticity and integrity. That is, a signature by Alice on a message demonstrates that it was Alice who signed (and, therefore, intended to send) that message; and that the message is exactly the message sent by Alice, and was not tampered with. In a legal setting, they are sometimes said to provide nonrepudiation; however, this term is not well defined [78].

Signatures are a standard cryptographic primitive with many applications in higher-level protocols.

My Thesis. Groups featuring a computable bilinear map are particularly well suited for signature-related primitives.

For some signature variants the only construction known is based on bilinear maps. Where constructions based on, e.g., RSA are known, bilinear-map–based constructions are simpler, more efficient, and yield shorter signatures.

Evidence. In this thesis, we describe several constructions that support the claim above.

First, we consider Boneh-Lynn-Shacham (BLS) and Boneh-Boyen short signatures. BLS signatures with security comparable to 1024-bit RSA are 160 bits long, the shortest of any scheme based on standard assumptions. BB signatures can be as short as BLS or (in a variant with longer signatures) can be proved secure without random oracle.

Next, we present several extension and variants of BLS signatures. Amongst these is the Boneh-Gentry-Lynn-Shacham (BGLS) aggregate signature scheme. In an aggregate signature scheme, it is possible, given n signatures on n distinct messages from n distinct users, to aggregate all these signatures into a single short signature. This single aggregate suffices to convince a verifier that the the users did indeed sign their respective messages.

BGLS aggregates are based on BLS signatures and are 160 bits long, regardless of how many signatures are aggregated. No construction is known for aggregate signatures that does not employ bilinear maps.

We also show that BGLS aggregates give rise to verifiably encrypted signatures, a signature variant with applications in contract signing.

In a digression, we show how one can construct sequential aggregate signatures based only on the existence of trapdoor permutations. Sequential aggregate signatures is variant of aggregate signatures in which signing-and-aggregation is a single operation, in which each signer adds her signature to the aggregate signature of all the signers before her.

Next, we present the Boneh-Boyen-Shacham (BBS) group signature scheme. Group signatures provide anonymity for signers. Any member of the group can sign messages, but the resulting signature keeps the identity of the signer secret. In some systems there is a third party that can trace the signature, or undo its anonymity, using a special trapdoor. BBS group signatures with security comparable to 1024-bit RSA are 1443 bits long, shorter than any previous scheme by an order of magnitude. The signing operation is also an order of magnitude more efficient than in previous schemes.

Finally, we consider variants and extensions of the BBS group signature scheme, including a group signature with a novel revocation mechanism that we call verifier-local revocation (VLR). In a VLR group signature, messages announcing the revocation of some users need only be processed by the verifiers; the signers are stateless. We present the Boneh-Shacham VLR group signature scheme, which has signatures even shorter than in BBS.

1.1 Previous Publication

The BLS short signature scheme of Section 3.3 and the notes on elliptic curve families in Section 2.3 originally appeared in "Short Signatures from the Weil Pairing," joint work with Dan Boneh and Ben Lynn, of which an extended abstract was presented at Asiacrypt 2001 [27] and which appeared in the Journal of Cryptology [28].

The BGLS aggregate signature scheme of Section 4.4 and the BGLS2 verifiably encrypted signature scheme of Section 4.5 originally appeared in "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," joint work with Dan Boneh, Craig Gentry, and Ben Lynn, which was presented at Eurocrypt 2003 [26].

The LMRS sequential aggregate signature scheme of Chapter 5 originally appeared in "Sequential Aggregate Signatures from Trapdoor Permutations," joint work with Anna Lysyanskaya, Silvio Micali, and Leonid Reyzin, which was presented at Eurocrypt 2004 [80].

The BBS group signature scheme of Chapter 6, along with its extensions in Sections 7.2 and 7.3 originally appeared in "Short Group Signatures," joint work with Xavier Boyen and Dan Boneh, which was presented at Crypto 2004 [24].

The BS group signature with verifier-local revocation of Section 7.4 originally appeared in "Group Signatures with Verifier-Local Revocation," joint work with Dan Boneh, which was presented at ACM CCS 2004 [29].

Chapter 2

Mathematical Background

2.1 Mathematical Setting

2.1.1 Groups

Throughout the thesis, we use the following notation:

- G_1 is a multiplicative cyclic group of prime order p;
- G_2 is a multiplicative group of exponent p, whose order is some power of p.
- ψ is a homomorphism from G_2 onto G_1.
- g_2 is an order-p element of G_2 and g_1 is a generator of G_1 such that $\psi(g_2) = g_1$.

The elements g_1 and g_2 are selected at random as part of system setup. Having selected g_2, we typically restrict G_2 to its cyclic order-p subgroup $\langle g_2 \rangle$. The restriction of ψ to this subgroup gives an isomorphism onto G_1.[1]

One could set $G_1 = G_2$, but we allow for the more general case where $G_1 \neq G_2$ so that we can take advantage of certain families of non-supersingular elliptic curves as described in Sections 2.3.5 and 2.3.5.

Some schemes described in this thesis make explicit use of the map ψ. For others (e.g., the BLS short signature scheme of Section 3.3), the map is used only in the proof of security. Even so, the map isn't merely a proof artifact. We give in Section 3.3.1 an example of a bilinear group pair on which the BLS signature scheme is insecure precisely because ψ does not exist.

2.1.2 The Bilinear Map

We also employ bilinear maps. For these, we use the following notation:

- G_T is a multiplicative cyclic group of order p.
- e is a map $e : G_1 \times G_2 \to G_T$ with the following properties:

[1]When G_2 is not restricted in this way, it is possible to use the pairing to test whether two points $g_2, h \in G_2$ are such that $h \in \langle g_2 \rangle$. Protocols in which messages include elements of G_2 can thus leak information. None of the protocols in this thesis transmits elements of G_2.

– Bilinear: for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.

– Non-degenerate: $e(\psi(g_2), g_2) \neq 1$ for all but at most a $(2/p)$-fraction of $g_2 \in G_2$.

When provided a generator g_2 by an untrusted party, one can use the pairing to verify that $e(\psi(g_2), g_2) \neq 1$ holds.

2.1.3 Running Times

Throughout this thesis, we use a concrete analysis in which time is measured according to some fixed computational model — say, state transitions in a probabilistic (oracle) Turing machine — and then normalized so that the following operations take unit time:

- computing the group operation on G_1 and on G_2;
- evaluating the homomorphism ψ;
- selecting an element of G_1 or G_2 uniformly at random; and
- evaluating the bilinear map e.

2.1.4 Hashing

Some schemes in this thesis make use of a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. Others require that H map onto G_1 or (in the case of BS group signatures, Section 7.4.2) onto G_2. In Section 2.3.3, we discuss how one might instantiate a hash function onto these groups.

2.2 Complexity Assumptions

2.2.1 Computational and Decisional co-Diffie-Hellman

With the setup above we obtain natural generalizations of the CDH and DDH problems:

Computational co-Diffie-Hellman (co-CDH) on (G_1, G_2): Given $g_2, g_2^a \in G_2$ and $h \in G_1$ as input, compute $h^a \in G_1$.

Decision co-Diffie-Hellman (co-DDH) on (G_1, G_2): Given $g_2, g_2^a \in G_2$ and $h, h^b \in G_1$ as input, output `yes` if $a = b$ and `no` otherwise.

We call a tuple of the form (g_2, g_2^a, h, h^a) a co-Diffie-Hellman tuple. When $G_1 = G_2$ these problems reduce to standard CDH and DDH.

We define the success probability of an algorithm $\mathcal{A}$ in solving the Computational co-Diffie-Hellman problem on (G_1, G_2) as

$$\mathbf{Adv}_{\mathcal{A}}^{\text{co-cdh}} \overset{\text{def}}{=} \Pr\left[\mathcal{A}(g_2, g_2^a, h) = h^a : g_2 \xleftarrow{\text{R}} G_2, a \xleftarrow{\text{R}} \mathbb{Z}_p, h \xleftarrow{\text{R}} G_1\right] \ .$$

The probability is over the uniform random choice of g_2 from G_2, a from $\mathbb{Z}_p$, h from G_1, and over the coin tosses of $\mathcal{A}$. We say that an algorithm $\mathcal{A}$ (t, ϵ)-breaks Computational co-Diffie-Hellman on (G_1, G_2) if $\mathcal{A}$ runs in time at most t, and $\mathbf{Adv}_{\mathcal{A}}^{\text{co-cdh}}$ is at least ϵ.

We are interested in the case where a computable bilinear map exists, but Computational co-Diffie-Hellman is hard, motivating the following definition:

Definition 2.2.1. We say that two groups (G_1, G_2) as in Section 2.1 above are a (t, ϵ)-bilinear group pair if no algorithm (t, ϵ)-breaks Computational co-Diffie-Hellman on (G_1, G_2). If $G_1 = G_2$, we say that G_1 is a bilinear group.

Joux and Nguyen [70] showed that an efficiently-computable bilinear map e provides an algorithm for solving the Decision co-Diffie-Hellman problem as follows: For a tuple (g_2, g_2^a, h, h^b) where $h \in G_1$ we have

$$a = b \bmod p \quad \iff \quad e(h, g_2^a) = e(h^b, g_2) \ .$$

This test succeeds except when $e(\psi(g_2), g_2) = 1$ and therefore $e(h, g_2) = 1$; but this only happens with negligible probability. Consequently, Decision co-Diffie-Hellman can be solved in 2 time units on a bilinear group pair (G_1, G_2).

When we wish to highlight that a scheme requires only that it be easy to solve Decision co-Diffie-Hellman on (G_1, G_2), we refer to (G_1, G_2) as a Gap co-Diffie-Hellman group pair. Specifically, two groups (G_1, G_2) as in Section 2.1 above (omitting the bilinear map e) are a (t, ϵ)-Gap co-Diffie-Hellman group pair if there is a procedure for solving Decision co-Diffie-Hellman on (G_1, G_2) in 2 time units, but no algorithm (t, ϵ)-break Computational co-Diffie-Hellman on (G_1, G_2). If $G_1 = G_2$, we say that G_1 is a Gap Diffie-Hellman group.

Currently, the only examples of such Gap co-Diffie-Hellman groups arise from bilinear maps. It is possible that other constructions for useful Gap co-Diffie-Hellman groups exist.

2.2.2 The Strong Diffie-Hellman Assumption

We present the q-Strong Diffie-Hellman (SDH) problem. This problem, introduced by Boneh and Boyen [23], has similar properties to the Strong-RSA problem [10], as we will see.

q-Strong Diffie-Hellman Problem: Given a $(q + 2)$-tuple $(g_1, g_2, g_2^{\gamma}, g_2^{(\gamma^2)}, \dots, g_2^{(\gamma^q)})$ as input, with $g_1 = \psi(g_2)$, output a pair $(g_1^{1/(\gamma+x)}, x)$, where $x \in \mathbb{Z}_p$.

We define the success probability of an algorithm $\mathcal{A}$ in solving the q-Strong Diffie-Hellman problem on (G_1, G_2) as

$$\mathbf{Adv}_{\mathcal{A}}^{\text{sdh}}(q) \stackrel{\text{def}}{=} \Pr\left[\begin{array}{r} \mathcal{A}(g_1, g_2, g_2^{\gamma}, \dots, g_2^{(\gamma^q)}) = (g_1^{\frac{1}{\gamma+x}}, x) \ : \\ g_2 \stackrel{\mathrm{R}}{\leftarrow} G_2, \gamma \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p^*, g_1 \leftarrow \psi(g_2) \end{array}\right]$$

The probability is over the uniform random choice of g_2 from G_2 and γ from $\mathbb{Z}_p$, and over the coin tosses of $\mathcal{A}$. An algorithm $\mathcal{A}$ (t, q, ϵ)-breaks Strong Diffie-Hellman on (G_1, G_2) if $\mathcal{A}$ runs in time at most t, and $\mathbf{Adv}_{\mathcal{A}}^{\text{sdh}}(q)$ is at least ϵ.

Definition 2.2.2. We say that the (q, t, ϵ)-SDH assumption holds in (G_1, G_2) if no t-time algorithm (t, q, ϵ)-breaks Strong Diffie-Hellman on (G_1, G_2).

Occasionally we drop the t and ϵ and refer to the q-SDH assumption rather than the (q, t, ϵ)-SDH assumption.

Mitsunari et al. [91] use a related assumption (where x is specified in advance rather than chosen by the adversary) in a tracing-traitors system.

To gain confidence in the q-SDH assumption, Boneh and Boyen prove [23] that it holds in generic groups in the sense of Shoup [109].

2.2.3 The Decision Linear Diffie-Hellman Assumption

With the setup as above, along with arbitrary generators u, v, and h of G_1, consider the following problem:

Decision Linear Problem in G_1: Given $u, v, h, u^a, v^b, h^c \in G_1$ as input, output `yes` if $a + b = c$ and `no` otherwise.

One can easily show that an algorithm for solving Decision Linear in G_1 gives an algorithm for solving DDH in G_1. The converse is believed to be false. That is, it is believed that Decision Linear is a hard problem even in bilinear groups where DDH is easy (e.g., when $G_1 = G_2$). More precisely, we define the advantage of an algorithm $\mathcal{A}$ in deciding the Decision Linear problem in G_1 as

$$\mathbf{Adv}_{\mathcal{A}}^{\text{linear}} \stackrel{\text{def}}{=} \left| \begin{array}{c} \Pr\left[\mathcal{A}(u, v, h, u^a, v^b, h^{a+b}) = \texttt{yes} : u, v, h \stackrel{\text{R}}{\leftarrow} G_1, a, b \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p\right] \\ - \Pr\left[\mathcal{A}(u, v, h, u^a, v^b, \eta) = \texttt{yes} : u, v, h, \eta \stackrel{\text{R}}{\leftarrow} G_1, a, b \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p\right] \end{array} \right| .$$

The probability is over the uniform random choice of the parameters to $\mathcal{A}$, and over the coin tosses of $\mathcal{A}$. We say that an algorithm $\mathcal{A}$ (t, ϵ)-decides Decision Linear in G_1 if $\mathcal{A}$ runs in time at most t, and $\mathbf{Adv}_{\mathcal{A}}^{\text{linear}}$ is at least ϵ.

Definition 2.2.3. We say that the (t, ϵ)-Decision Linear Assumption holds in G_1 if no t-time algorithm has advantage at least ϵ in solving the Decision Linear problem in G_1.

Boneh, Boyen, and Shacham show [24] that the Decision Linear Assumption holds in generic bilinear groups [109].

Linear Encryption

The Decision Linear problem gives rise to the Linear encryption scheme, a natural extension of ElGamal encryption. Unlike ElGamal encryption, Linear encryption can be secure even in groups where a DDH-deciding algorithm exists. In this scheme, a user's public key is a triple of generators $u, v, h \in G_1$; her private key is the exponents $x, y \in \mathbb{Z}_p$ such that $u^x = v^y = h$. To encrypt a message $M \in G_1$, choose random values $a, b \in \mathbb{Z}_p$, and output the triple $(u^a, v^b, m \cdot h^{a+b})$. To recover the message from an encryption (T_1, T_2, T_3), the user computes $T_3/(T_1^x \cdot T_2^y)$. By a natural extension of the proof of security of ElGamal, Linear encryption is semantically secure against a chosen-plaintext attack, assuming Decision Linear holds.

2.2.4 Implications of DDH Hardness on G_1

When G_1 and G_2 are distinct groups, the Boneh-Boyen-Shacham proof [24] shows that, in the generic model, the standard Decision Diffie-Hellman (DDH) problem is hard in the group G_1 (even though DDH in G_2 is easy). For DDH to be hard in a specific group G_1, the map $\psi : G_2 \to G_1$ must be computationally one-way. This requirement may hold when the bilinear groups are instantiated using the Weil or Tate pairing over MNT and Barreto-Naehrig curves, which we discuss in Section 2.3.4 below. In this instantiation, G_1 is defined over the ground field of the curve where as G_2 is defined over a low-degree extension. Supersingular curves do not have this property since DDH is known to be easy on all cyclic subgroups [55].

Now suppose that for MNT curves the DDH assumption holds in G_1. In this case we can construct even shorter group signatures and group signatures that satisfy CCA2-full-anonymity.

Shorter Group Signatures. If DDH holds in G_1 then ElGamal encryption is secure in G_1 and can be used as the encryption in the BBS group signature of Section 6.3: $T_1 = u^\alpha$, $T_2 = A \cdot v^\alpha$. (The preimages $\psi^{-1}(u), \psi^{-1}(v) \in G_2$ of $u, v \in G_1$ must not be revealed.) The group signature then comprises only two elements of G_1 and four of $\mathbb{Z}_p$. With parameters chosen as in Section 6.3, we obtain a 1022-bit group signature whose security is comparable to that of standard 1024-bit RSA signatures. This is about 30% shorter than the signatures in Section 6.3.

Full-CCA-Anonymity. Likewise, if DDH holds in G_1 then the Cramer-Shoup encryption scheme [44] is secure in G_1, and can be used in the BBS group signature scheme. Since Cramer-Shoup encryption is semantically secure against an adaptive CCA2 attack, the resulting group signature scheme is CCA2-fully-anonymous and thus secure in the full BMW model. Cramer-Shoup encryption entails a four-tuple (T_1, T_2, T_3, T_4) of elements of G_1. The proof of security entails four elements of $\mathbb{Z}_p$. Instantiated with the same parameters as above, the resulting group signature is 1364 bits long.

We emphasize that currently nothing is known about the complexity of the DDH problem in the ground field of an MNT curve and relying on this assumption seems risky. This question deserves further study.

2.3 Elliptic Curves and Bilinear Maps

We quickly summarize the results from elliptic curves on which we rely. For more details, see Blake, Seroussi, and Smart [20], Galbraith [53], Menezes [84], Lang [77], and Silverman [111].

2.3.1 Notation and Background

Let q be a prime power. We use $E/\mathbb{F}_q$ to denote an elliptic curve with coefficients in $\mathbb{F}_q$. For $l \geq 1$, we use $E(\mathbb{F}_{q^l})$ to denote the group of points on E in $\mathbb{F}_{q^l}$. We use $\#E(\mathbb{F}_{q^l})$ to denote the number of points in $E(\mathbb{F}_{q^l})$.

Let r be a prime dividing $\#E(\mathbb{F}_q)$ such that $r \nmid q$. The embedding degree of $E/\mathbb{F}_q$ is the smallest positive integer k such that $r \mid q^k - 1$. Then $\mathbb{F}_{q^k}^*$ contains $\boldsymbol{\mu}_r$, the group of rth roots of unity.

Let $E(\mathbb{F}_{q^k})[r]$ be the group of r-torsion points in $E(\mathbb{F}_{q^k})$, i.e., $\{P \in E(\mathbb{F}_{q^k}) \mid rP = \mathcal{O}\}$. Then $E(\mathbb{F}_{q^k})[r]$ is a group of exponent r, and is isomorphic to $\mathbb{Z}_r \times \mathbb{Z}_r$. Similarly, let $rE(\mathbb{F}_{q^k})$ be the group $\{rP \mid P \in E(\mathbb{F}_{q^k})\}$. Then $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ is also a group of exponent r, and is isomorphic to $\mathbb{Z}_r \times \mathbb{Z}_r$.

For the curves of interest, $E(\mathbb{F}_{q^k})[r] \wedge rE(\mathbb{F}_{q^k}) = \emptyset$, and we can represent elements of $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ using $E(\mathbb{F}_{q^k})[r]$.

The Weil and Tate pairings. The Weil pairing is a map $e : E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})[r] \to \boldsymbol{\mu}_r$ with the following properties:

(i) Identity: for all $R \in E(\mathbb{F}_{q^k})[r]$, $e(R, R) = 1$.

(ii) Bilinear: for all $R_1, R_2 \in E(\mathbb{F}_{q^k})[r]$ and $a, b \in \mathbb{Z}$ we have $e(aR_1, bR_2) = e(R_1, R_2)^{ab}$.

(iii) Non-degenerate: if for $R \in E(\mathbb{F}_{q^k})[r]$ we have $e(R, R') = 1$ for all $R' \in E(\mathbb{F}_{q^k})[r]$, then $R = \mathcal{O}$.

(iv) Computable: for all $R_1, R_2 \in E[p]$, the pairing $e(R_1, R_2)$ can be computed in polynomial time [89].

The Tate pairing [52] (with final powering by $(q^k-1)/r$) is a map $e : E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \boldsymbol{\mu}_r$. This pairing is again bilinear, computable, and non-degenerate, though the non-degeneracy condition is more complicated.

The Groups G_1, G_2, and G_T. We define G_1 to be $E(\mathbb{F}_q)[r]$, the r-torsion points of E over the base field $\mathbb{F}_q$. This group has r points. (Pairing evaluation is more efficient when the first argument has coordinates in the base field.) We define G_2 to be $E(\mathbb{F}_{q^k})[r]$ for the Weil pairing, or $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ for the Tate pairing. (Again, these are equivalent for our purposes.) This is a group of exponent r, with r^2 points.[2] Finally, G_T is the group $\boldsymbol{\mu}_r$.

The Trace Map. We present a computable homomorphism $\psi : G_2 \to G_1$, using the trace map, tr, which sends points in $E(\mathbb{F}_{q^k})$ to $E(\mathbb{F}_q)$. Let $\sigma_1, \ldots, \sigma_k$ be the Galois maps of $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$. Also, for $R = (x, y) \in E(\mathbb{F}_{q^k})$ define $\sigma_i(R) = (\sigma_i(x), \sigma_i(y))$. Then the trace map $\mathrm{tr} : E(\mathbb{F}_{q^k}) \to E(\mathbb{F}_q)$ is defined by:

$$\mathrm{tr}(R) = (1/k)\big[\sigma_1(R) + \cdots + \sigma_k(R)\big] \ .$$

In fact, for points in $E(\mathbb{F}_{q^k})[r] = G_2$, the output of tr is in $E(\mathbb{F}_q)[r] = G_1$. It is easy to show that tr is a homomorphism from G_2 to G_1. It is computable in time polynomial in k and $\log q$ as required.

2.3.2 Intractability of co-CDH on (G_1, G_2)

The remaining question is the difficulty of the co-CDH problem on (G_1, G_2). We review necessary conditions for CDH intractability. The best known algorithm for solving co-CDH on (G_1, G_2) is to compute discrete-log in G_1. In fact, the discrete-log and CDH problems in G_1 are known to be computationally equivalent given some extra information about the group G_1 [82]. Therefore, it suffices to consider necessary conditions for making the discrete-log problem on $E(\mathbb{F}_q)$ intractable.

Let $\langle P \rangle$ be a subgroup of $E(\mathbb{F}_q)$ of order r with embedding degree k. We briefly discuss two standard ways for computing discrete-log in $\langle P \rangle$.

1. **MOV:** Use an efficiently computable homomorphism, as in the Menezes-Okamoto-Vanstone reduction [83], to map the discrete log problem in $\langle P \rangle$ to a discrete log problem in some extension of $\mathbb{F}_q$, say $\mathbb{F}_{q^i}$. We then solve the discrete log problem in $\mathbb{F}^*_{q^i}$ using the Number Field Sieve algorithm [107]. The image of $\langle P \rangle$ under this homomorphism must be a subgroup of $\mathbb{F}^*_{q^i}$ of order r. Thus we have $r \mid q^i - 1$, which by the definition of k implies that $i \geq k$. Hence, the MOV method can, at best, reduce the discrete log problem in $\langle P \rangle$ to a discrete log problem in a subgroup of $\mathbb{F}^*_{q^k}$. Therefore, to ensure that discrete log is hard in $\langle P \rangle$ we want curves where k is sufficiently large to make discrete log in $\mathbb{F}^*_{q^k}$ intractable.

2. **Generic:** Generic discrete log algorithms such as Baby-Step-Giant-Step and Pollard's Rho method [85] have a running time proportional to $\sqrt{p}\log p$. Therefore, we must ensure that p is sufficiently large.

In summary, we want curves $E/\mathbb{F}_q$ where both a generic discrete log algorithm in $E(\mathbb{F}_q)$ and the Number Field Sieve in $\mathbb{F}^*_{q^k}$ are intractable. At the same time, since elements of G_1 have

[2]Of the r^2 points, r will coincide with G_1, and r will have trace $\mathcal{O}$. This motivates the $(2/p)$ constant in Section 2.1.2 above.

representation of length $\lceil \log_2 q \rceil$ and elements G_2 have representation of length $\lceil k \log_2 q \rceil$, we wish to keep q as small as possible.

2.3.3 Hashing onto elliptic curves

Many schemes based in this thesis require a hash function $H : \{0,1\}^* \to G_1$. In the elliptic curve setting above, this requires a map onto $E(\mathbb{F}_q)[r]$. Since it is difficult to build hash functions that hash directly onto a subgroup of an elliptic curve we slightly relax the hashing requirement.

Let $\mathbb{F}_q$ be a field of characteristic greater than 2. Let $E/\mathbb{F}_q$ be an elliptic curve defined by $y^2 = f(x)$ and let $E(\mathbb{F}_q)$ have order m. Let $P \in E(\mathbb{F}_q)$ be a point of prime order r, where p^2 does not divide m. We wish to hash onto the subgroup $G_1 = \langle P \rangle$. Suppose we are given a hash function $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$. Such hash functions H' can be built from standard cryptographic hash functions. The security analysis will view H' as a random oracle. We use the following deterministic algorithm called MapToGroup to hash messages in $\{0,1\}^*$ onto G_1. Fix a small parameter $I = \lceil \log_2 \log_2(1/\delta) \rceil$, where δ is some desired bound on the failure probability.

MapToGroup$_{H'}$: The algorithm defines $H : \{0,1\}^* \to G_1$ as follows:

1. Given $M \in \{0,1\}^*$, set $i \leftarrow 0$;
2. Set $(x, b) \leftarrow H'(i \parallel M) \in \mathbb{F}_q \times \{0,1\}$, where i is represented as an I-bit string;
3. If $f(x)$ is a quadratic residue in $\mathbb{F}_q$ then do:

   3a. Let $y_0, y_1 \in \mathbb{F}_q$ be the two square roots of $f(x)$. We use $b \in \{0,1\}$ to choose between these roots. Choose some full ordering of $\mathbb{F}_q$ and ensure that y_1 is greater than y_0 according to this ordering (swapping y_0 and y_1 if necessary). Set $\tilde{P}_M \in E(\mathbb{F}_q)$ to be the point $\tilde{P}_M = (x, y_b)$.

   3b. Compute $P_M = (m/r)\tilde{P}_M$. Then P_M is in G_1. If $P_M \neq \mathcal{O}$, declare that MapToGroup$_{H'}(M) = P_M$ and stop; otherwise, continue with Step 4.

4. Otherwise, increment i, and go to Step 2; if i reaches 2^I, report failure.

The failure probability can be made arbitrarily small by picking an appropriately large I. For each i, the probability that $H'(i \parallel M)$ leads to a point on G_1 is approximately $1/2$ (where the probability is over the choice of the random oracle H'). Hence, the expected number of calls to H' is approximately 2, and the probability that a given message M will be found unhashable is $1/2^{(2^I)} \leq \delta$.

It can be shown [28] that BLS signatures remain secure when the when the hash function H is computed with MapToGroup$_{H'}$ and H' is a random oracle hash function $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$. Similar arguments apply to other schemes.

Hashing onto G_2. A similar procedure can be used to construct a hash function with domain in G_2. In this case, it is important that by G_2 we mean the full r^2-element group $E(\mathbb{F}_{q^k})[r]$. It is an open problem to construct a hash onto the r-element subgroup $\langle Q \rangle$ when $Q \notin E(\mathbb{F}_q)[r] = G_1$.

2.3.4 Suitable Curves

We briefly describe three families of elliptic curves with useful embedding degrees. The first two families have embedding degrees $k = 6$; the third has embedding degree $k = 12$.

curve	l	Sig Size $\lceil \log_2 3^l \rceil$	DLog Security $\lceil \log_2 r \rceil$	MOV Security $\lceil 6 \log_2 3^l \rceil$
E^-	79	126	126	752
E^+	97	154	151	923
E^+	121	192	155	1151
E^+	149	237	220	1417
E^+	163	259	256	1551
E^-	163	259	259	1551

Table 2.1: Supersingular elliptic curves with $k = 6$. Here r is the largest prime divisor of $\#E(\mathbb{F}_{3^l})$ The MOV reduction maps the curve onto a field of characteristic 3 of size 3^{6l}.

Supersingular Curves

Boneh, Lynn, and Shacham note [28] that the supersingular curves $E^{\pm} : y^2 = x^3 + 2x \pm 1$ over $\mathbb{F}_{3^l}$ have embedding degree $k = 6$, the most of any supersingular curves (cf. [118, 76]). They also show that curves E^+ and E^- have a useful automorphism that make the prime-order subgroups of $E^+(\mathbb{F}_{3^l})$ and $E^-(\mathbb{F}_{3^l})$ into bilinear groups (as opposed to bilinear group pairs).

Some useful instantiations of these curves are presented in Table 2.1. Note that we restrict these instantiations to those where ℓ is prime, to avoid Weil-descent attacks [56, 59], except for $\ell = 121$. It has recently been shown that certain Weil-descent attacks are not effective for this case [46].

Performance There is a substantial literature on speeding up pairing evaluation on supersingular curves over fields of low characteristic [54, 12, 11]. Accordingly, pairing-based protocols implemented using the curves $E^{\pm}$ will be much faster than using the other curves discussed in this section.

Shorter Representation for G_1. To obtain larger embedding degree, Rubin and Silverberg [106] propose certain Abelian varieties. They show that elements of G_1 using the supersingular curves proposed here can be shortened by 20%. The result is an n-bit signature where the pairing reduces the discrete log problem to a finite field of size approximately $2^{7.5n}$.

2.3.5 The bad news

MOV reduces the discrete log problem on $E^+(\mathbb{F}_{3^l})$ and $E^-(\mathbb{F}_{3^l})$ to a discrete log problem in $\mathbb{F}^*_{3^{6l}}$. A discrete-log algorithm due to Coppersmith [38, 107] is specifically designed to compute discrete log in small characteristic fields. Consequently, a discrete-log problem in $\mathbb{F}^*_{3^n}$ is much easier than a discrete-log problem in $\mathbb{F}^*_p$ where p is a prime of approximately the same size as 3^n. To get security equivalent to DSA using a 1024-bit prime, we would have to use a curve $E(\mathbb{F}_{3^l})$ where 3^{6l} is much larger than 1024 bits. This leads to much longer signatures, defeating the point of using these curves.

Discriminant D	Signature Size $\lceil\log_2 q\rceil$	DLog Security $\lceil\log_2 r\rceil$	MOV Security $\lceil 6\log_2 q\rceil$
13368643	149	149	894
254691883	150	147	900
8911723	157	157	942
62003	159	158	954
12574563	161	161	966
1807467	163	163	978
6785843	168	166	1008
28894627	177	177	1062
153855691	185	181	1110
658779	199	194	1194
1060147	203	203	1218
20902979	204	204	1224
9877443	206	206	1236

Table 2.2: Suitable MNT curves. Here E is a curve over the prime field $\mathbb{F}_q$ and r is the largest prime dividing its order. The MOV reduction maps the curve onto the field $\mathbb{F}_{q^6}$. D is the discriminant of the complex multiplication field of $E/\mathbb{F}_q$.

MNT Curves

Miyaji, Nakabayashi, and Takano [92] describe a family of ordinary (non-supersingular) elliptic curves with $k = 6$.

These curves are constructed using complex multiplication [20, chapter VIII].

Table 2.2 gives some values of the discriminant D that lead to suitable curves.

Barreto-Naehrig Curves

Recently, Barreto and Naehrig [13] described a family of ordinary curves with $k = 12$.

Table 2.3 describes some suitable curves produced by the Barreto-Naehrig method.

Signature Size $\lceil\log_2 q\rceil$	DLog Security $\lceil\log_2 r\rceil$	MOV Security $\lceil 6\log_2 q\rceil$
160	160	1920
192	192	2304
224	224	2688
256	256	3072

Table 2.3: Suitable Barreto-Naehrig curves. Here E is a curve over the prime field $\mathbb{F}_q$ and r is the largest prime dividing its order. The MOV reduction maps the curve onto the field $\mathbb{F}_{q^6}$. D is the discriminant of the complex multiplication field of $E/\mathbb{F}_q$.

Chapter 3

Short Signatures

3.1 Introduction

Short digital signatures are needed in environments with strong bandwidth constraints. For example, product registration systems often ask users to key in a signature provided on a CD label. When a human is asked to type in a digital signature, the shortest possible signature is needed. Similarly, due to space constraints, short signatures are needed when one prints a bar-coded digital signature on a postage stamp [102, 93]. As a third example, consider legacy protocols that allocate a fixed short field for non-repudiation [3, 69]. One would like to use the most secure signature that fits in the allotted field length.

The two most frequently used signatures schemes, RSA and DSA, produce relatively long signatures compared to the security they provide. For example, when one uses a 1024-bit modulus, RSA signatures are 1024 bits long. Similarly, when one uses a 1024-bit modulus, standard DSA signatures are 320 bits long. Elliptic curve variants of DSA, such as ECDSA, are also 320 bits long [4]. A 320-bit signature is too long to be keyed in by a human.

In this Chapter, we describe two signature schemes in which signature length is approximately 160 bits and which provide a level of security similar to that of 320-bit DSA signatures. The first, BLS, is secure against existential forgery under a chosen-message attack (in the random oracle model), assuming the Computational Diffie-Hellman problem (CDH) is hard on certain elliptic curves over a finite field. The second, BB, is secure assuming the Strong Diffie-Hellman problem is hard. A BB variant can also be proven secure without random oracles, but signatures in this variant are twice as long.

For both BLS and BB, generating a signature is a simple multiplication on the curve. Verifying the signature is done using a bilinear pairing on the curve. Both schemes inherently use properties of curves. Consequently they have no equivalent in $\mathbb{F}_q^*$, the multiplicative group of a finite field.

Constructing short signatures is an old problem. Several proposals show how to shorten DSA while preserving the same level of security. Naccache and Stern [93] propose a variant of DSA where the signature length is approximately 240 bits. Mironov [90] suggests a DSA variant with a similar length and gives a concrete security analysis of the construction in the random oracle model. Other work aims at reducing the length of signature in the RSA setting. For example, Gentry shows how to compress Rabin signatures to two-thirds of their original length. [63]. Another technique proposed for reducing signature length is signatures with message recovery [95, 102]. In such systems one encodes a part of the message into the signature thus shortening the total length of the message-signature pair. For long messages, one can then achieve a DSA signature overhead of 160 bits. However, for very short messages (e.g., 64 bits) the total length remains 320 bits. Using BLS or BB, the signature length is always on the order of 160 bits, however short the message. We also note that Patarin et al. [99, 43] construct short signatures whose security depends on the Hidden Field Equation problem.

The BLS signature scheme resembles the undeniable signature scheme of Chaum and Pedersen [35]. Because of its simple mathematical structure, the scheme has several useful properties. These are described in the next chapter. The BB signature scheme is related to the group signature schemes presented in Chapters 6 and 7.

3.2 Signature Security Definitions

Formally, a signature scheme is a triple of algorithms $\mathcal{SIG} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$, which behave as follows.

Sig.Kg. This randomized algorithm outputs a public verification key pk and a private signing key sk.

Sig.Sig(sk, M)**.** This algorithm takes as input a private key sk and a message M in some message space (typically $\{0,1\}^*$) to be signed, and returns a signature σ on M.

Sig.Vf(pk, M, σ)**.** The verification algorithm takes as input a public key pk, and a purported signature σ on a message M. It returns either `valid` or `invalid`.

The signing algorithm Sig can also be a randomized algorithm, in which case we say that the signature scheme is randomized. In a randomized signature scheme, the signing algorithm will typically issue different signatures if reinvoked with different randomness. Even if the signing algorithm is not randomized, there might still be more than one valid signature on a given message under a given public key.[1] A signature scheme where this never occurs — where for every valid public key and message there is only a single signature that Vf accepts as valid — is said to be unique.

We now recall the standard definition for signature scheme security. Existential unforgeability under a chosen message attack [66] for a signature scheme $\mathcal{SIG}$ is defined using the following game between a challenger and an adversary $\mathcal{A}$:

Setup. The challenger runs algorithm Kg to obtain a public key pk and private key sk. The adversary $\mathcal{A}$ is given pk.

Queries. Proceeding adaptively, $\mathcal{A}$ requests signatures with pk on at most q_S messages of his choice $M_1, \dots, M_{q_s} \in \{0,1\}^*$. The challenger responds to each query with a signature $\sigma_i = \mathsf{Sig}(sk, M_i)$. In the random oracle model, the adversary can also make q_H queries to a hash oracle H.

Output. Eventually, $\mathcal{A}$ outputs a pair (M, σ) and wins the game if (1) M is not any of $M_1, \dots, M_{q_s}$, and (2) $\mathsf{Vf}(pk, M, \sigma) = \texttt{valid}$.

[1] An example is the BLS variant with tight security reduction given by Katz and Wang [71], where every message has *two* valid signatures, only one of which is ever output by the signing algorithm.

We define $\mathbf{Adv}^{\text{sig-ef-cma}}_{\mathcal{SIG},\mathcal{A}}$ to be the probability that $\mathcal{A}$ wins in the above game, taken over the coin tosses of Kg, of $\mathcal{A}$, and of Sig if it randomized.

For non-unique signature schemes, it is possible that the adversary can obtain a signature σ on a message M from its signing oracle and transform it into a different valid signature σ'. Under the definition above, this is not considered a forgery. (Under a related security definition, *strong* existential unforgeability, this would be a forgery [2]. The signature schemes in this thesis are all unique, however, so we do not consider strong unforgeability further.)

Definition 3.2.1. A forger $\mathcal{A}$ (t, q_S, q_H, ϵ)-breaks a signature scheme $\mathcal{SIG}$ if $\mathcal{A}$ runs in time at most t, $\mathcal{A}$ makes at most q_S signature queries and at most q_H queries to the hash function, and $\mathbf{Adv}^{\text{sig-ef-cma}}_{\mathcal{SIG},\mathcal{A}}$ is at least ϵ. A signature scheme is (t, q_S, q_H, ϵ)-existentially unforgeable under an adaptive chosen-message attack if no forger (t, q_S, q_H, ϵ)-breaks it.

3.3 Short Signatures based on CDH

We present a signature scheme that works on any Gap co-Diffie-Hellman group pair (G_1, G_2). We prove security of the scheme and, in the next section, show how it leads to short signatures. The scheme resembles the undeniable signature scheme proposed by Chaum and Pedersen [35]. Okamoto and Pointcheval [98] briefly note that gap problems can give rise to signature schemes. However, most gap problems will not lead to short signatures.

Let (G_1, G_2) be (t, ϵ)-Gap co-Diffie-Hellman group pair where $|G_1| = |G_2| = p$. A signature σ is an element of G_1. The signature scheme comprises three algorithms, Kg, Sig, and Vf. It makes use of a full-domain hash function $H : \{0,1\}^* \to G_1$. In Section 2.3.3 we weaken the requirement on the hash function H. The security analysis views H as a random oracle [16, 17].

BLS.Kg. Pick random $x \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$ and compute $v \leftarrow g_2^x$. The public key pk is $v \in G_2$. The private key sk is x.

BLS.Sig(sk, M)**.** Parse the user's private key sk as $x \in \mathbb{Z}_p$. Compute $h \leftarrow H(M) \in G_1$ and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

BLS.Vf(pk, M, σ)**.** Parse the user's public key pk as $v \in G_2$. Compute $h \leftarrow H(M) \in G_1$ and verify that (g_2, v, h, σ) is a valid co-Diffie-Hellman tuple. If so, output `valid`; if not, output `invalid`.

A signature is a single element of G_1. To construct short signatures, therefore, we need co-GDH group pairs where elements in G_1 have a short representation. We briefly describe how to construct such groups in Section 2.3. Using the Barreto Naehrig curves of Section 2.3.5, we can obtain 160-bit signatures with 1024-bit security.

3.3.1 Security

We prove the security of the BLS signature scheme against existential forgery under adaptive chosen-message attacks in the random oracle model. Security follows from the hardness of co-CDH on (G_1, G_2). When $G_1 = G_2$ security is based on the standard Computational Diffie-Hellman assumption in G_1.

Theorem 3.3.1. *Let (G_1, G_2) be a (t', ϵ')-co-GDH group pair of order p. Then BLS on (G_1, G_2) is (t, q_S, q_H, ϵ)-secure against existential forgery under an adaptive chosen-message attack (in the random oracle model), for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + 1) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - c_{G_1}(q_H + 2q_S) \ .$$

Here c_{G_1} is a constant that depends on G_1, and e is the base of the natural logarithm.

Proof. Suppose $\mathcal{A}$ is a forger algorithm that (t, q_S, q_H, ϵ)-breaks the signature scheme. We show how to construct a t'-time algorithm $\mathcal{B}$ that solves co-CDH on (G_1, G_2) with probability at least ϵ'. This will contradict the fact that (G_1, G_2) is a (t', ϵ')-co-GDH group pair.

Let g_2 be a generator of G_2. Algorithm $\mathcal{B}$ is given $g_2, u \in G_2$ and $h \in G_1$, where $u = g_2^a$. Its goal is to output $h^a \in G_1$. Algorithm $\mathcal{B}$ simulates the challenger and interacts with forger $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{B}$ starts by giving $\mathcal{A}$ the generator g_2 and the public key $u \cdot g_2^r \in G_2$, where r is random in $\mathbb{Z}_p$.

H-queries. At any time algorithm $\mathcal{A}$ can query the random oracle H. To respond to these queries algorithm $\mathcal{B}$ maintains a list of tuples $\langle M_j, w_j, b_j, c_j \rangle$ as explained below. We refer to this list as the H-list. The list is initially empty. When $\mathcal{A}$ queries the oracle H at a point $M_i \in \{0,1\}^*$, algorithm $\mathcal{B}$ responds as follows:

1. If the query M_i already appears on the H-list in a tuple $\langle M_i, w_i, b_i, c_i \rangle$ then algorithm $\mathcal{B}$ responds with $H(M_i) = w_i \in G_1$.
2. Otherwise, $\mathcal{B}$ generates a random coin $c_i \in \{0, 1\}$ so that $\Pr[c_i = 0] = 1/(q_S + 1)$.
3. Algorithm $\mathcal{B}$ picks a random $b_i \in \mathbb{Z}_p$ and computes $w_i \leftarrow h^{1-c_i} \cdot \psi(g_2)^{b_i} \in G_1$.
4. Algorithm $\mathcal{B}$ adds the tuple $\langle M_i, w_i, b_i, c_i \rangle$ to the H-list and responds to $\mathcal{A}$ by setting $H(M_i) = w_i$.

Note that either way w_i is uniform in G_1 and is independent of $\mathcal{A}$'s current view as required.

Signature queries. Let M_i be a signature query issued by $\mathcal{A}$. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to H-queries to obtain a $w_i \in G_1$ such that $H(M_i) = w_i$. Let $\langle M_i, w_i, b_i, c_i \rangle$ be the corresponding tuple on the H-list. If $c_i = 0$ then $\mathcal{B}$ reports failure and terminates.
2. Otherwise, we know $c_i = 1$ and hence $w_i = \psi(g_2)^{b_i} \in G_1$. Define $\sigma_i = \psi(u)^{b_i} \cdot \psi(g_2)^{rb_i} \in G_1$. Observe that $\sigma_i = w_i^{a+r}$ and therefore σ_i is a valid signature on M_i under the public key $u \cdot g_2^r = g_2^{a+r}$. Algorithm $\mathcal{B}$ gives σ_i to algorithm $\mathcal{A}$.

Output. Eventually algorithm $\mathcal{A}$ produces a message-signature pair (M_f, σ_f) such that no signature query was issued for M_f. If there is no tuple on the H-list containing M_f then $\mathcal{B}$ issues a query itself for $H(M_f)$ to ensure that such a tuple exists. We assume σ_f is a valid signature on M_f under the given public key; if it is not, $\mathcal{B}$ reports failure and terminates. Next, algorithm $\mathcal{B}$ finds the tuple $\langle M_f, w, b, c \rangle$ on the H-list. If $c = 1$ then $\mathcal{B}$ reports failure and terminates. Otherwise, $c = 0$ and therefore $H(M_f) = w = h \cdot \psi(g_2)^b$. Hence, $\sigma = h^{a+r} \cdot \psi(g_2)^{b(a+r)}$. Then $\mathcal{B}$ outputs the required h^a as $h^a \leftarrow \sigma / (h^r \cdot \psi(u)^b \cdot \psi(g_2)^{rb})$.

This completes the description of algorithm $\mathcal{B}$. It remains to show that $\mathcal{B}$ solves the given instance of the co-CDH problem on (G_1, G_2) with probability at least ϵ'. To do so, we analyze the three events needed for $\mathcal{B}$ to succeed:

$\mathcal{E}_1$**:** $\mathcal{B}$ does not abort as a result of any of $\mathcal{A}$'s signature queries.

$\mathcal{E}_2$**:** $\mathcal{A}$ generates a valid message-signature forgery (M_f, σ_f).

$\mathcal{E}_3$**:** Event $\mathcal{E}_2$ occurs and $c = 0$ for the tuple containing M_f on the H-list.

$\mathcal{B}$ succeeds if all of these events happen. The probability $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3]$ is:

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \ . \tag{3.1}$$

The following claims give a lower bound for each of these terms.

Claim 1. *The probability that algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s signature queries is at least $1/e$. Hence,* $\Pr[\mathcal{E}_1] \geq 1/e$.

Proof. Without loss of generality we assume that $\mathcal{A}$ does not ask for the signature of the same message twice. We prove by induction that after $\mathcal{A}$ makes i signature queries the probability that $\mathcal{B}$ does not abort is at least $(1 - 1/(q_S + 1))^i$. The claim is trivially true for $i = 0$. Let M_i be $\mathcal{A}$'s i'th signature query and let $\langle M_i, w_i, b_i, c_i \rangle$ be the corresponding tuple on the H-list. Then prior to issuing the query, the bit c_i is independent of $\mathcal{A}$'s view — the only value that could be given to $\mathcal{A}$ that depends on c_i is $H(M_i)$, but the distribution on $H(M_i)$ is the same whether $c_i = 0$ or $c_i = 1$. Therefore, the probability that this query causes $\mathcal{B}$ to abort is at most $1/(q_S + 1)$. Using the inductive hypothesis and the independence of c_i, the probability that $\mathcal{B}$ does not abort after this query is at least $(1 - 1/(q_S + 1))^i$. This proves the inductive claim. Since $\mathcal{A}$ makes at most q_S signature queries the probability that $\mathcal{B}$ does not abort as a result of all the signature queries is at least $(1 - 1/(q_S + 1))^{q_S} \geq 1/e$. □

Claim 2. *If algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s signature queries then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Hence,* $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.

Proof. The public key given to $\mathcal{A}$ is from the same distribution as a public key produced by algorithm Kg. Responses to H-queries are as in the real attack since each response is uniformly and independently distributed in G_1. All responses to signature queries are valid. Therefore, $\mathcal{A}$ will produce a valid message-signature pair with probability at least ϵ. Hence, $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$. □

Claim 3. *The probability that algorithm $\mathcal{B}$ does not abort after $\mathcal{A}$ outputs a valid forgery is at least $1/(q_S + 1)$. Hence,* $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] = 1/(q_S + 1)$.

Proof. Given that events $\mathcal{E}_1$ and $\mathcal{E}_2$ happened, algorithm $\mathcal{B}$ will abort only if $\mathcal{A}$ generates a forgery (M_f, σ_f) for which the tuple $\langle M_f, w, b, c \rangle$ on the H-list has $c = 1$. At the time $\mathcal{A}$ generates its output it knows the value of c_i for those M_i for which it issued a signature query. All the remaining c_i's are independent of $\mathcal{A}$'s view. Indeed, if $\mathcal{A}$ did not issue a signature query for M_i then the only value given to $\mathcal{A}$ that depends on c_i is $H(M_i)$, but the distribution on $H(M_i)$ is the same whether $c_i = 0$ or $c_i = 1$. Since $\mathcal{A}$ could not have issued a signature query for M_f we know that c is independent of $\mathcal{A}$'s current view and therefore $\Pr[c = 0 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] = 1/(q_S + 1)$ as required. □

Using the bounds from the claims above in equation (3.1) shows that $\mathcal{B}$ produces the correct answer with probability at least $\epsilon/\big(e(q_S+1)\big) \geq \epsilon'$ as required. Algorithm $\mathcal{B}$'s running time is the same as $\mathcal{A}$'s running time plus the time it takes to respond to $(q_H + q_S)$ hash queries and q_S signature queries. Each query requires an exponentiation in G_1 which we assume takes time c_{G_1}. Hence, the total running time is at most $t + c_{G_1}(q_H + 2q_S) \leq t'$ as required. This completes the proof of Theorem 3.3.1. $\square$

The analysis used in the proof of Theorem 3.3.1 resembles Coron's analysis of the Full Domain Hash (FDH) signature scheme [39]. We note that the security analysis can be made tight using Probabilistic Full Domain Hash (PFDH) [40], at the cost of increasing signature length. The security reduction in Theorem 3.3.1 can also be made tight without increasing signature length via the technique of Katz and Wang [71].

The BLS signature scheme requires an algorithm for deciding DDH. In groups where a DDH-deciding algorithm is not available, Goh and Jarecki [65] show that it is still possible to construct a signature scheme based on CDH, at the cost of substantially greater signature length. (Previous signature schemes based on CDH had only loose security reductions, through the forking lemma [103].) The scheme analyzed by Goh and Jarecki has since been improved by Katz and Wang [71] and by Chevallier-Mames [37], but signatures in these variants are still longer than BLS signatures.

The Necessity of the Map $\psi : G_2 \to G_1$. Recall that the proof of security relied on the existence of an efficiently computable isomorphism $\psi : G_2 \to G_1$. To show the necessity of ψ we give an example of a bilinear map $e : G_1 \times G_2 \to G_T$ for which the co-CDH problem is believed to be hard on (G_1, G_2) and yet the resulting signature scheme is insecure.

Let q be a prime and let G_2 be a subgroup of $\mathbb{Z}_q^*$ of prime order p with generator g. Let G_1 be the group $G_1 = \mathbb{Z}_p$ with addition. Define the map $e : G_1 \times G_2 \to G_2$ as $e(x, y) = y^x$. The map is clearly bilinear since $e(ax, y^b) = e(x, y)^{ab}$. The co-CDH problem on (G_1, G_2) is as follows: Given $g, g^a \in G_2$ and $x \in G_1$ compute $ax \in G_1$. The problem is believed to be hard since an algorithm for computing co-CDH on (G_1, G_2) gives an algorithm for computing discrete log in G_2. Hence, (G_1, G_2) satisfies all the conditions of Theorem 3.3.1 except that there is no known computable isomorphism $\psi : G_2 \to G_1$. It is is easy to see that the resulting signature scheme from this bilinear map is insecure. Given one message-signature pair, it is easy to recover the private key.

We comment that one can avoid using ψ at the cost of making a stronger complexity assumption [112]. Without ψ the necessary assumption for proving security is that no polynomial time algorithm can compute $h^a \in G_1$ given $g_2, g_2^a \in G_2$ and $g_1, g_1^a, h \in G_1$. Since ψ naturally exists in all the group pairs (G_1, G_2) we are considering, there is no reason to rely on this stronger complexity assumption.

3.4 Short Signatures based on SDH

Boneh and Boyen give a simple signature scheme based on the Strong Diffie-Hellman assumption [23]. In their paper, they present several variants. We will describe two of them: one that gives a signature as short as BLS, secure in the random oracle model; and another that gives signatures secure without random oracles. The first of these was independently discovered by Zhang et al. [120]. It proceeds as follows.

BB.Kg. Select $\gamma \xleftarrow{R} \mathbb{Z}_p^*$, and set $w \leftarrow g_2^\gamma$. key pk is $w \in G_2$. The private key sk is γ.

BB.Sig(sk, M)**.** Parse the user's private key sk as $\gamma \in \mathbb{Z}_p$. Compute $x \leftarrow H(M) \in \mathbb{Z}_p$ and $\sigma \leftarrow g_1^{1/(\gamma+x)}$. (If it happens that $\gamma + x$ equals 0, the message cannot be signed.) The signature is $\sigma \in G_1$.

BB.Vf(pk, M, σ)**.** Parse the user's public key pk as $w \in G_2$. Compute $x \leftarrow H(M) \in \mathbb{Z}_p$ and verify that $e(\sigma, wg_2^x) = e(g_1, g_2)$ holds. If so, output `valid`; if not, output `invalid`.

It is easy to see that the scheme is valid: for a correctly formed signature, we have $e(\sigma, wg_2^x) = e(g_1^{1/(\gamma+x)}, g_2^\gamma \cdot g_2^x) = e(g_1, g_2)$, as required.

3.4.1 Proof of Security

We begin by describing a technique, used in Lemma 3.2 [23], of which we will make use again in proving the security of the BBS group signature scheme (Section 6.4) and the BS VLR group signature scheme (Section 7.4.4). A similar construction occurs also in the traitor-tracing scheme of Mitsunari, Sakai, and Kasahara [91].

Using this technique, we prove the security of the BB scheme in the random oracle model.

The Fundamental SDH Technique. We are given a q-SDH instance $(g_1', g_2', (g_2')^\gamma, (g_2')^{\gamma^2}, \ldots, (g_2')^{\gamma^q})$, where $g_1' = psi(g_2')$. We compute generators $g_1 \in G_1$, $g_2 \in G_2$, $w = g_2^\gamma$, and $q-1$ SDH pairs (A_i, x_i) such that $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$ for each i.

We do this as follows. Consider $q-1$ values $x_1, \ldots, x_{q-1}$ (chosen arbitrarily). Define formal products $f(X)$ and $g(X)$ as

$$f(X) = \prod_{i=1}^{q-1}(X + x_i) \qquad \text{and} \qquad g(X) = X \cdot \prod_{i=1}^{q-1}(X + x_i)$$

and, for each i, $1 \le i \le q-1$, define the products $f_i(X)$ as

$$f_i(X) = \prod_{\substack{1 \le j \le q-1 \\ i \ne j}} (X + x_j) \enspace .$$

Each of these is a polynomial of degree at most q; for each, we can compute the coefficients of the X-powers in $O(q)$ time.

Suppose $g(X)$ expands as $\sum_{i=0}^q a_i X^i$. Using the coefficients a_i and the SDH problem parameters, we can evaluate $(g_2')^{g(\gamma)}$ as $\prod_{i=0}^q \left[(g_2')^{\gamma^i}\right]^{a_i}$. The same holds true for the other formal products. Each such evaluation takes $O(q)$ time.

Now, we make the assignment

$$\begin{gathered} g_2 \leftarrow (g_2')^{f(\gamma)} \qquad w \leftarrow (g_2')^{g(\gamma)} \qquad g_1 \leftarrow \psi(g_2) \\ A_i \leftarrow \psi\big((g_2')^{f_i(\gamma)}\big), \quad 1 \le i \le q-1 \end{gathered}$$

It is easy to see that $w = g_2^\gamma$ holds and, for each i that $A_i^{\gamma+x_i} = g_1$ holds; thus we have $q-1$ SDH pairs for the SDH problem instance (g_1, g_2, w). Note that if g_2' is a random generator of G_2, so is g_2.

Now suppose we find another SDH pair (A, x), where $x \notin \{x_1, \ldots, x_{q-1}\}$. We transform this pair into an SDH pair for the original problem instance. Let $t(X)$ be the rational function $f(X)/(X+x)$. Using long division, we can write $t(X)$ as $t(X) = \frac{\alpha}{X+x} + \tau(X)$, where $\tau(X)$ is a $(q-2)$-degree polynomial. Because $x \notin \{x_1, \ldots, x_{q-1}\}$, α cannot be 0. By the the procedure used above, we can evaluate $g_2^{\tau(\gamma)}$. By the SDH equation and the setup above, we have $A^{\gamma+x} = g_1 = \psi\big((g_2')^{f(\gamma)}\big)$ and $A = \psi\big((g_2')^{t(\gamma)}\big) = \psi\big((g_2')^{\frac{\alpha}{\gamma+x}+\tau(\gamma)}\big)$. Now set

$$A' \leftarrow \left[\frac{A}{\psi\big(g_2^{\tau(\gamma)}\big)}\right]^{1/\alpha} = \left[\frac{\psi\big((g_2')^{\frac{\alpha}{\gamma+x}+\tau(\gamma)}\big)}{\psi\big(g_2^{\tau(\gamma)}\big)}\right]^{1/\alpha} = \psi\big((g_2')^{\frac{\alpha}{\gamma+x}}\big)^{1/\alpha} = (g_1')^{1/\gamma+x} \ ;$$

then (A', x) is an SDH solution for the original SDH parameters, as required. As before, this step takes $O(q)$ time.

We are now ready to prove that the BB scheme is secure.

Theorem 3.4.1. *Suppose (q', t', ϵ')-SDH holds on (G_1, G_2). Then BB on (G_1, G_2) is (t, q_S, q_H, ϵ)-secure against existential forgery under an adaptive chosen-message attack (in the random oracle model), for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + 1) \cdot \epsilon' \qquad \textit{and} \qquad t \leq t' - O(q_H^2) \ ,$$

and for $q \geq q_H + 1$. Here e is the base of the natural logarithm.

Proof. We assume that $\mathcal{A}$ is well-behaved in the sense that it always requests the hash of a message M before it requests a signature on M and at M^* before it forges at M^*. It is trivial to modify any forger algorithm $\mathcal{A}$ to have this property.

Given a q-SDH instance $(g_1', g_2', (g_2')^{\gamma}, (g_2')^{\gamma^2}, \ldots, (g_2')^{\gamma^q})$, we apply the fundamental SDH technique above, obtaining generators $g_1 \in G_1$, $g_2 \in G_2$, $w = g_2^{\gamma}$, and $q-1$ SDH pairs (A_i, x_i) such that x_i is uniformly chosen from $\mathbb{Z}_p$ and $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$ for each i. We will obtain from the adversary $\mathcal{A}$ another SDH pair (A, x), which will be transformed into a solution to the original q-SDH instance, again using the fundamental technique.

The proof now proceeds much as the proof of Theorem 3.3.1 did. We run $\mathcal{A}$ with parameters (g_1, g_2, w). To respond to the ith hash query, on message M_i, we generate a random coin $c_i \in \{0, 1\}$ so that $\Pr[c_i = 0] = 1/(q_S + 1)$. If c_i is 1, we set $h_i \leftarrow x_i$; otherwise, we set $h_i \xleftarrow{\text{R}} \mathbb{Z}_p$. In either case, we respond with $H(M_i) = h_i$. To signature query on M_i we respond with A_i if c_i is 1, or report failure and exit if c_i is 0. Finally, $\mathcal{A}$ outputs a forgery (M^*, σ^*), where $M^* = M_{i^*}$ for some i^*. If c_{i^*} is 1, we report failure and exit. Otherwise, we have an SDH pair (σ^*, h_{i^*}), and $h_{i^*} \notin \{x_1, \ldots, x_{q-1}\}$ with overwhelming probability.

The same independence analysis as in the Claims of Theorem 3.3.1 shows that we succeed with probability $\epsilon/\big(e(q_S+1)\big) \geq \epsilon'$, as required. The running time overhead is essentially just that of the fundamental technique, which is $O(q^2)$. □

3.4.2 A BB Variant Secure without Random Oracles

Boneh and Boyen also show that a simple modification of the BB scheme above can be proved secure in the standard model, i.e., without random oracles. We obtain this modified BB2 scheme as follows. We add to the private key a value $\gamma' \xleftarrow{\text{R}} \mathbb{Z}_p^*$ and to the public key the value $w' \leftarrow g_2^{\gamma'}$.

To sign a message $M \in \mathbb{Z}_p$, choose a random $r \in \mathbb{Z}_p$, compute $\sigma \leftarrow g_1^{1/(\gamma+r\gamma'+M)}$, and output the pair (σ, r). (If it happens that $\gamma + r\gamma' + M$ equals 0, try again with a different r.) To verify, check that $e\big(\sigma,\, w \cdot (w')^r \cdot g_2^M\big) = e(g_1, g_2)$ holds.

The BB2 proof of security also uses the fundamental SDH technique. We use the free choice of r in the signing oracle to force $\gamma + r\gamma' + M$ to hit one of the $\gamma + x_i$ values which we precomputed; this allows us to do away with the hash oracle. We must deal with two types of forger, as follows.

Type-I Forger. This adversary either makes a hash query $M_i = -\gamma$, or issues a forgery (σ^*, r^*) at M^* such that $r^*\gamma' + M^* \notin r_1\gamma' + M_1, \ldots, r_q\gamma' + M_q$. Against this adversary we pick $\gamma' \xleftarrow{\text{R}} \mathbb{Z}_p^*$ and set $w' \leftarrow g_2^{\gamma'}$. We answer a signing query on message M_i by setting $r_i \leftarrow (x_i - M_i)/\gamma'$ so that $x_i = r_i\gamma' + M_i$ and we can use the SDH pair (A_i, x_i). (We also check whether M_i equals $-\gamma$; if so, we can compute any SDH pair we wish.) Finally, the forgery (σ^*, r^*) on M^* gives us an SDH pair $(\sigma^*, r^*\gamma' + M^*)$ which is different from each pair (A_i, x_i) by hypothesis.

Type-II Forger. This adversary never makes a hash query $M_i = -\gamma$, and issues a forgery (σ^*, r^*) at M^* such that $r^*\gamma' + M^* = r_{i^*}\gamma' + M_{i^*}$ for some i^*. For this adversary, we choose $\gamma \xleftarrow{\text{R}} \mathbb{Z}_p$ ourselves, set $w \leftarrow g_2^{\gamma}$, and use the values from the fundamental SDH technique for γ' and w'; that is, we have pairs $(A_i, x_i$ such that $e(A_i, (w')g_2^{x_i}) = e(g_1, g_2)$. Now we answer a signature query on M_i by choosing $r_i \leftarrow (\gamma + M_i)/x_i)$ and $\sigma_i \leftarrow A_i^{1/r_i}$. Then

$$e\big(\sigma_i,\, w \cdot (w')^{r_i} \cdot g_2^{M_i}\big) = e\big(A_i^{1/r_i},\, (w')^{r_i} \cdot g_2^{\gamma+M_i}\big) = e(A_i,\, w' \cdot g_2^{x_i}) = e(g_1, g_2) \enspace ,$$

as required. Finally, the adversary returns the forgery (σ^*, r^*) on M^*, such that $r^*\gamma' + M^* = x_{i^*}$ for some i^*. (We can find i^* by testing $e\big((w')^{r^*} \cdot g_2^{M^*}\big) \stackrel{?}{=} e\big((w')^{r_i} \cdot g_2^{M_i}\big)$ for each i.) But this means that we have $r^*\gamma' + M^* = r_{i^*}\gamma' + M_{i^*}$ for $(r^*, M^*) \neq (r_{i^*}, M_{i^*})$ since otherwise the forgery would be trivial, and we recover γ' as $(M^* - M_{i^*})/(r^* - r_{i^*})$, from which we can compute any SDH pair we wish.

The omitted details of the reduction are quite straightforward. Note that BB2 is secure if q-SDH holds where $q = q_S + 1$, not $q_H + 1$ as for BB.

BB2 signatures are about twice as long as BLS signatures. However, they are much shorter than those in previous schemes with proofs in the standard model: in particular, the Cramer-Shoup scheme [45], which is based on the Strong RSA assumption. We note that the Waters identity-based encryption scheme [119] gives a signature secure in the standard model under CDH. This follows from Naor's observation (recorded by Boneh and Franklin [25]) that every IBE gives rise to a signature scheme.

3.4.3 Performance

Though $\mathsf{BLS.Sig}$ and $\mathsf{BB.Sig}$ appear to be equally fast, BB (and BB2) signing is in fact substantially faster. First, a hash function mapping into $\mathbb{Z}_p$ can be computed without the iterated trials suggested in Section 2.3.3 for hashing onto G_1. Second, the inversion in $\mathbb{Z}_p$, required for computing $1/(\gamma+x)$, is faster than taking roots in $\mathbb{Z}_p$, again required for hashing onto g_1. Third, for BB the exponentiation is with respect to the fixed base g_1, and is amenable to speedup using lookup tables.[2] For BLS, the best we can do is to find an addition chain for the fixed exponent x. Taken together, these differences make BB signing about 5 times as fast as BLS signing. BB verification is also somewhat faster, since it requires computing a single pairing rather than the product of two pairings.

3.5 Conclusions

We presented two short signature schemes, BLS and BB, based on bilinear maps on elliptic curves. In both schemes, a signature is only one element in a finite field, much shorter than all current variants of DSA for the same security. BLS is existentially unforgeable under a chosen message attack (in the random oracle model), assuming the Computational Diffie-Hellman problem is hard on certain elliptic-curve groups; BB is secure assuming the Strong Diffie-Hellman problem is hard.

Both schemes are simple and elegant, and therefore amenable to extension. In Chapter 4, we consider several variants of BLS. In Chapters 6 and 7 we build group signature schemes related to BB.

[2]Specifically, precomputing $\left[u, u^2, \ldots, u^{2^b-1}\right]; \left[u^{2^b}, u^{2^b\cdot 2}, \ldots, u^{2^b\cdot(2^b-1)}\right]; \left[u^{2^{2b}}, u^{2^{2b}\cdot 2}, \ldots, u^{2^{2b}\cdot(2^b-1)}\right]; \ldots$ allows one to evaluate u^x in $\lceil(\lg p)/b\rceil$ multiplications, at the cost of $2^b \cdot \lceil(\lg p)/b\rceil$ elements of storage. This technique was communicated to me by Xavier Boyen.

Chapter 4

Signature Variants and Extensions

4.1 Introduction

The BLS signature scheme presented in Chapter 3 is very flexible. In this chapter, we show how to add a number of features to it.

We begin by surveying, in Sections 4.2 and 4.3, how BLS has been extended to construct some of the standard signature variants in the literature, including threshold signatures, batch signature verification, and multisignatures. For further extensions, such as blind signatures, refer to Verheul [117], Boldyreva [22], Steinfeld et al. [114], and the survey of pairing-based cryptosystems by Paterson [100].

We then introduce, in Section 4.4, a generalization of multisignatures called an aggregate signatures: a single short object that stands for n signatures by n different signers on n different messages. Aggregate signatures have several important applications. For example, they can be used to reduce the size of certificate chains and reduce communication bandwidth in protocols such as SBGP. We construct an efficient aggregate signature scheme based on bilinear maps. Key generation, aggregation, and verification require no interaction. We prove security in a model that gives the adversary his choice of public keys and messages to forge.

Finally, as a further application for aggregate signatures we show in Section 4.5 that certain aggregate signature schemes give rise to simple verifiably encrypted signatures. These signatures enable user Alice to give Bob a signature on a message M encrypted using a third party's public key and enable Bob to verify that the encrypted signature is valid. Unlike previous schemes, our verifiably encrypted are short and can be validated efficiently; and verifiably encrypted signing requires no interaction with the third party.

4.2 Threshold Signatures

Boldyreva shows [22] that BLS signatures give rise to a robust t-out-of-n threshold signature scheme through standard secret sharing techniques [85, 62, 60]. In a threshold signature scheme, there are n parties where each possesses a share of a private key. Each party can use its share of the private key to produce a share of a signature on some message M. A complete signature on M can only be constructed if at least t shares of the signature are available.

A robust t-out-of-n threshold signature scheme derives from BLS as follows. A central authority generates a public/private key pair. Let $x \in \mathbb{Z}_p$ be the private key and $v = g_2^x \in G_2$ be the public key. The central authority picks a random polynomial $\omega \in \mathbb{Z}_p[X]$ of degree at most $t-1$ such that $\omega(0) = x$. For $i = 1, \ldots, n$, the authority gives user i the value $x_i = \omega(i)$, its share of the private key. The authority publishes the public key v and n values $u_i = g_2^{x_i} \in G_2$.

When a signature on a message $M \in \{0,1\}^*$ is needed each party that wishes to participate in signature generation publishes its share of the signature as $\sigma_i = H(M)^{x_i} \in G_1$. Without loss of generality, assume users $1, \ldots, t$ participate and generate shares $\sigma_1, \ldots, \sigma_t$. Anyone can verify that share σ_i is valid by checking that $(g_2, u_i, H(M), \sigma_i)$ is a co-Diffie-Hellman tuple. When all t shares are valid, the complete signature is recovered as

$$\sigma \leftarrow \prod_{i=1}^{t} \sigma_i^{\lambda_i} \quad \text{where} \quad \lambda_i = \frac{\prod_{i=1, j\neq i}^{t}(0-j)}{\prod_{i=1, j\neq i}^{t}(i-j)} \pmod{p} \ .$$

If fewer than t users are able to generate a signature on some message M then these users can be used to solve co-CDH on (G_1, G_2) [22]. This threshold scheme is robust: A participant who contributes a bad partial signature σ_i will be detected immediately since $(g_2, u_i, H(M), \sigma_i)$ will not be a co-Diffie-Hellman tuple.

The same result can also be achieved without recourse to a trusted third party for generating private key shares. The n users can generate these shares without the help of a trusted party using the protocol due to Gennaro et al. [61], which is a modification of a protocol due to Pedersen [101]. This protocol does not rely on the difficulty of DDH for security and can thus be employed on gap Diffie-Hellman groups.

4.3 Multisignatures and Batch Signature Verification

Suppose n users all sign the same message $M \in \{0,1\}^*$. We obtain n signatures $\sigma_1, \ldots, \sigma_n$. We show that these n signatures can be verified as a batch much faster than verifying them one by one. A similar property holds for other signature schemes [14].

Let (G_1, G_2) be a co-GDH group pair of prime order p. Suppose user i's private key is $x_i \in \mathbb{Z}_p$ and his public key is $v_i = g_2^{x_i} \in G_2$. Signature σ_i is $\sigma_i = H(M)^{x_i} \in G_1$. To verify the n signatures as a batch we use a technique due to Bellare et al. [14]:

1. Pick random integers $c_1, \ldots, c_n$ from the range $[0, B]$ for some value B. This B controls the error probability as discussed below.

2. Compute $V \leftarrow \prod_{i=1}^{n} v_i^{c_i} \in G_2$ and $U \leftarrow \prod_{i=1}^{n} \sigma_i^{c_i} \in G_1$.

3. Test that $(g_2, V, H(M), U)$ is a co-Diffie-Hellman tuple. Accept all n signatures if so; reject otherwise.

Theorem 3.3 of [14] shows that we incorrectly accept the n signatures with probability at most $1/B$. Hence, verifying the n signatures as a batch is faster than verifying them one by one. Note that if all signers are required to prove knowledge of their private keys, then taking $c_1 = \cdots = c_n = 1$ is sufficient, yielding even faster batch verification [22].

In fact, if we take $c_1 = \cdots = c_n = 1$, we can transmit $U = \prod_i \sigma_i$ instead of the individual signatures. Since U is the same size as any single signature, we get a substantial bandwidth savings. This is a multisignature: a single short object that stands for n signatures by n signers on the *same* message.

There is a large literature on multisignatures [97, 96]. Micali, Ohta, and Reyzin [87] were the first to define a security model for multisignatures. Boldyreva [22] introduced the BLS-based multisignature scheme given above and showed, using a variant of the Micali-Ohta-Reyzin definitions, that it secure in gap Diffie-Hellman groups. The proof requires, as above, that all signers prove knowledge of their private keys. This is to address an attack that we consider further in Section 4.4.2, in the context of BGLS aggregate signatures.

A similar batch verification procedure can be used to verify quickly, and a similar multisignature can be used to compress, n signatures on n messages issued by the *same* public key.

4.4 Aggregate Signatures

Many real-world applications involve signatures on many different messages generated by many different users. For example, in a Public Key Infrastructure (PKI) of depth n, each user is given a chain of n certificates. The chain contains n signatures by n Certificate Authorities (CAs) on n distinct certificates. Similarly, in the Secure BGP protocol (SBGP) [72] each router receives a list of k signatures attesting to a certain path of length k in the network. A router signs its own segment in the path and forwards the resulting list of $k + 1$ signatures to the next router. As a result, the number of signatures in routing messages is linear in the length of the path. Both applications would benefit from a method for compressing the list of signatures on distinct messages issued by distinct parties. Specifically, X.509 certificate chains could be shortened by compressing the n signatures in the chain into a single signature.

An aggregate signature scheme enables us to achieve precisely this type of compression. Suppose each of n users has a public-private key pair (pk_i, sk_i). User u_i signs message M_i to obtain a signature σ_i. Then there is a public aggregation algorithm that takes as input all of $\sigma_1, \ldots, \sigma_n$ and outputs a short compressed signature σ. Anyone can undertake the aggregation. Moreover, the aggregation can be incremental: Signatures σ_1, σ_2 can be aggregated into σ_{12} which can then be further aggregated with σ_3 to obtain σ_{123}. In the generation of a certificate chain, each CA can incrementally aggregate its own signature into the chain. There is also an aggregate verification algorithm that takes $pk_1, \ldots, pk_n$, $M_1, \ldots, M_n$, and σ and decides whether the aggregate signature is valid. Intuitively, the security requirement is that the aggregate signature σ is declared valid only if the aggregator who created σ was given all of $\sigma_1, \ldots, \sigma_n$. Precise security definitions are given in Sect. 4.4.1. Thus, an aggregate signature provides non-repudiation at once on many different messages by many users.

We construct an aggregate signature scheme based on BLS short signatures. Surprisingly, though BLS signatures can be instantiated on any gap group, general gap groups are insufficient for constructing efficient aggregate signatures. Our construction relies on properties of bilinear maps beyond their use — noted by Joux and Nguyen [70] — for solving DDH.

Related Work. Aggregate signatures are related to multisignatures, which we described in Section 4.3. Multisignatures are insufficient for the applications we have in mind, such as certificate chains and SBGP. For these applications we must be able to aggregate signatures on distinct messages.

Also relevant are threshold signatures, in particular the BLS-based threshold signature considered in Section 4.2 and the non-interactive threshold signature scheme due to Shoup [110], where we have a set of n signers, and a threshold t, such that signature shares from any $t < k \leq n$ signers can be combined into one signature. These differ from aggregate signatures in several important respects: threshold signatures require an expensive (or trusted) setup procedure; each piece of a threshold signature is not a stand-alone signature; pieces of a threshold signature can be combined into a signature only once there are enough of them; and a threshold signature looks the same no matter which of the signers contributed pieces to it.

Our application of aggregate signatures to compressing certificate chains is related to an open problem posed by Micali and Rivest [88]: Given a certificate chain and some special additional signatures, can intermediate links in the chain be cut out? Aggregate signatures allow the compression of certificate chains without any additional signatures, but a verifier must still be aware of all intermediate links in the chain. We note that batch RSA [50] also provides some signature compression, but only for signatures produced by a single signer.

4.4.1 Aggregate Signature Definitions

Consider a set $\mathbb{U}$ of users. Every user $u \in \mathbb{U}$ has a signing keypair (pk_u, sk_u). We wish to aggregate the signatures of some subset $U \subseteq \mathbb{U}$. Each user $u \in U$ produces a signature σ_u on a message M_u of her choice. These signatures are then combined into a single aggregate σ by an aggregating party. This aggregating party, which can be different from and untrusted by the users in U, has access to the users' public keys, to the messages, and to the signatures on them, but not to any private keys. The result of this aggregation is an aggregate signature σ whose length is the same as that of any of the individual signatures. This aggregate has the property that a verifier given σ along with the identities of the parties involved and their respective messages is convinced that each user signed her respective message.

Formally, an aggregate signature scheme $\mathcal{AS}$ is a signature scheme (as in Section 3.2) with two additional algorithms, Agg and AVf, which provide the aggregation capability. These algorithms behave as follows.

AS.Kg, AS.Sig, AS.Vf. As in standard signature schemes.

AS.Agg$(\{pk_i, M_i, \sigma_i\}_{i=1}^{k})$**.** For each user in the aggregate the algorithm takes, as input: a public key pk_i, a message M_i in some message space; and a signature σ_i on M_i. The algorithm combines the signatures into an aggregate signature σ, and outputs σ.

AS.AVf$(\{pk_i, M_i\}_{i=1}^{k}, \sigma)$**.** For each user in the aggregate, the algorithm takes a public key and a message. The algorithm also takes a purported aggregate signature σ. It returns either `valid` or `invalid`.

The notion of aggregate signature security we consider is a generalization of existential unforgeability of ordinary signatures, which we recalled in Section 3.2. Specifically, the adversary attempts to forge an aggregate signature, on messages of his choice, by some set of users; clearly, at least one of the users in the set must not be under the adversary's control.

We formalize the intuition above as the aggregate chosen-key security model. In this model, the adversary $\mathcal{A}$ is given a single public key. His goal is the existential forgery of an aggregate signature that includes the challenge key; aside from that challenge key the adversary may choose all public keys in the forgery. The adversary is also given access to a signing oracle on the challenge key. His advantage, $\mathbf{Adv}^{\text{agg-forge}}_{\mathcal{AS},\mathcal{A}}$, is defined to be his probability of success in the following game.

Setup. The challenger runs algorithm Kg to obtain a public key pk_1 and private key sk_1. The aggregate forger $\mathcal{A}$ is given pk_1.

Queries. Proceeding adaptively, $\mathcal{A}$ requests signatures with pk_1 on at most q_S messages of his choice $M^{(1)}), \ldots, M^{(q_s)} \in \{0,1\}^*$. The challenger responds to each query with a signature $\sigma^{(i)} = \mathsf{Sig}(sk, M^{(i)})$. In the random oracle model, $\mathcal{A}$ can also make q_H queries to a hash oracle H.

Response. Finally, $\mathcal{A}$ outputs $k-1$ additional public keys $pk_2, \ldots, pk_k$. Here k is at most n, a game parameter. These keys, along with the initial key pk_1, will be included in $\mathcal{A}$'s forged aggregate. $\mathcal{A}$ also outputs messages $M_1, \ldots, M_k$; and, finally, an aggregate signature σ by the k users, each on his corresponding message.

The forger wins if the aggregate signature σ is a valid aggregate under public keys $pk_1, \ldots, pk_k$ on respective messages $M_1, \ldots, M_k$ and σ is nontrivial, i.e., $\mathcal{A}$ did not request a signature on M_1 under pk_1. The probability is over the coin tosses of the key-generation algorithm and of $\mathcal{A}$.

Definition 4.4.1. An aggregate forger $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks an n-user aggregate signature scheme in the aggregate chosen-key model if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the hash function and at most q_S queries to the signing oracle; $\mathbf{Adv}^{\text{agg-forge}}_{\mathcal{AS},\mathcal{A}}$ is at least ϵ; and the forged aggregate signature is by at most n users. An aggregate signature scheme is $(t, q_H, q_S, n, \epsilon)$-secure against existential forgery in the aggregate chosen-key model if no forger $(t, q_H, q_S, n, \epsilon)$-breaks it.

4.4.2 Aggregate Signatures from Bilinear Maps

We describe the BGLS aggregate signature scheme. This scheme is based on the BLS scheme presented in Sect. 3.3 above. Individual signatures in the aggregate signature scheme are created and verified precisely as in BLS. Aggregate verification makes use of a bilinear map on G_1 and G_2.

The BGLS scheme allows the creation of signatures on arbitrary distinct messages $M_i \in \{0,1\}^*$. An individual signature σ_i is an element of G_1. The base groups G_1 and G_2, their respective generators g_1 and g_2, the computable isomorphism ψ from G_2 to G_1, and the bilinear map $e : G_1 \times G_2 \to G_T$, with target group G_T, are system parameters. The scheme employs a full-domain hash function $H : \{0,1\}^* \to G_1$, viewed as a random oracle.

We describe the algorithms making up BGLS below. The first three are the same as in BLS, and we recall them here for convenience.

BGLS.Kg. For a particular user, pick random $x \xleftarrow{\text{R}} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key pk is $v \in G_2$. The user's private key sk is $x \in \mathbb{Z}_p$.

BGLS.Sig(sk, M)**.** Parse the user's private key sk as $x \in \mathbb{Z}_p$. The message M to be signed is an arbitrary string. Compute $h \leftarrow H(M) \in G_1$ and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

BGLS.Vf(pk, M, σ)**.** Parse the user's public key pk as $v \in G_2$. Compute $h \leftarrow H(M)$; accept if $e(\sigma, g_2) = e(h, v)$ holds.

BGLS.Agg$(\{pk_i, M_i, \sigma_i\}_{i=1}^k)$**.** For each user in the aggregate the algorithm takes, as input: a public key pk_i, which we parse as $v_i \in G_2$; a message $M_i \in \{0,1\}^*$; and a signature $\sigma_i \in G_1$ on M_i. The messages must all be distinct. Compute $\sigma \leftarrow \prod_{i=1}^k \sigma_i$. The aggregate signature is $\sigma \in G_1$.

BGLS.AVf$(\{pk_i, M_i\}_{i=1}^k, \sigma)$. For each user in the aggregate, the algorithm takes a public key and a message, which we parse as above; further, the algorithm takes a purported aggregate signature $\sigma \in G_1$. To verify the aggregate signature σ,

1. ensure that the messages M_i are all distinct, and reject otherwise; and
2. compute $h_i \leftarrow H(M_i)$ for $1 \le i \le k = |U|$, and accept if $e(\sigma, g_2) = \prod_{i=1}^k e(h_i, v_i)$ holds.

A BGLS aggregate signature, like a BLS signature, is a single element of G_1. Note that incremental aggregation is possible.

The intuition behind BGLS aggregation is as follows. Each user u_i has a private key $x_i \in \mathbb{Z}_p$ and a public key $v_i = g_2^{x_i}$. User u_i's signature, if correctly formed, is $\sigma_i = h_i^{x_i}$, where h_i is the hash of the user's chosen message, M_i. The aggregate signature σ is thus $\sigma = \prod_i \sigma_i = \prod_i h_i^{x_i}$. Using the properties of the bilinear map, the left-hand side of the verification equation expands:

$$e(\sigma, g_2) = e\big(\prod\nolimits_i h_i^{x_i}, g_2\big) = \prod\nolimits_i e(h_i, g_2)^{x_i} = \prod\nolimits_i e(h_i, g_2^{x_i}) = \prod\nolimits_i e(h_i, v_i) \ ,$$

which is the right-hand side, as required.

A Potential Attack. The adversary's ability in the chosen-key model to generate keys suggests the following attack, previously considered in the context of multisignatures [87]. Alice publishes her public key v_A. Bob generates a private key x'_B and a public key $v'_B = g_2^{x'_B}$, but publishes as his public key $v_B = v'_B/v_A$, a value whose discrete log he does not know. Then $H(M)^{x'_B}$ verifies as an aggregate signature on M by both Alice and Bob. Note that in this forgery Alice and Bob both sign the same message M.

One countermeasure is to require the adversary to prove knowledge of the discrete logarithms (to base g_2) of his published public keys. For example, Boldyreva, in her multisignature scheme [22], requires, in effect, that the adversary disclose the corresponding private keys $x_2, \ldots, x_k$. Micali et al. [87] discuss a series of more sophisticated approaches based on zero-knowledge proofs, again with the effect that the adversary is constrained in his key selection. These defenses apply equally well to our aggregate signature scheme. For aggregate signatures, though, there is a simpler defense.

A Simple Defense. In the context of aggregate signatures we can defend against the attack above by simply requiring that an aggregate signature is valid only if it is an aggregation of signatures on *distinct* messages. This restriction, codified in Step 1 of BGLS.AVf, suffices to prove the security of the bilinear aggregate signature scheme in the chosen-key model. There is no need for zero-knowledge proofs or the disclosure of private keys.

The requirement that all messages in an aggregate be distinct is naturally satisfied for the applications to certificate chains and SBGP we have in mind. Even in more general environments it is easy to ensure that all messages are distinct: The signer simply prepends her public key to every message she signs prior to the application of the hash function H. The implicit prefix need not be transmitted with the signature, so signature and message length is unaffected.

The next theorem shows that this simple constraint is in fact sufficient for proving security in the chosen-key model.

Theorem 4.4.2. *Let (G_1, G_2) be a (t', ϵ')-bilinear group pair for co-Diffie-Hellman, with each group of order p, with respective generators g_1 and g_2, with an isomorphism ψ computable from G_2 to G_1,* *and with a bilinear map $e : G_1 \times G_2 \to G_T$. Then the BGLS aggregate signature scheme on (G_1, G_2) is $(t, q_H, q_S, n, \epsilon)$-secure against existential forgery in the aggregate chosen-key model for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + n) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - c_{G_1}(q_H + 2q_S + n + 4) - (n + 1) \ ,$$

where e is the base of natural logarithms, and exponentiation and inversion on G_1 take time c_{G_1}.

Proof. Suppose $\mathcal{A}$ is a forger algorithm that $(t, q_S, q_H, n, \epsilon)$-breaks BGLS. We show how to construct a t'-time algorithm $\mathcal{C}$ that solves co-CDH in (G_1, G_2) with probability at least ϵ'. This will contradict the fact that (G_1, G_2) are a (t', ϵ')-co-GDH group pair.

Let g_2 be a generator of G_2. Algorithm $\mathcal{C}$ is given $g_2, u \in G_2$ and $h \in G_1$, where $u = g_2^a$. Its goal is to output $h^a \in G_1$. Algorithm $\mathcal{C}$ simulates the challenger and interacts with forger $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{C}$ starts by giving $\mathcal{A}$ the generator g_2 and the public key $v_1 = u \cdot g_2^r \in G_2$, where r is random in $\mathbb{Z}_p$.

Hash Queries. At any time algorithm $\mathcal{A}$ can query the random oracle H. To respond to these queries, $\mathcal{C}$ maintains a list of tuples $\langle M^{(i)}, w^{(i)}, b^{(i)}, c^{(i)} \rangle$ as explained below. We refer to this list as the H-list. The list is initially empty. When $\mathcal{A}$ queries the oracle H at a point $M \in \{0,1\}^*$, algorithm $\mathcal{C}$ responds as follows:

1. If the query M already appears on the H-list in some tuple $\langle M, w, b, c \rangle$ then algorithm $\mathcal{C}$ responds with $H(M) = w \in G_1$.
2. Otherwise, $\mathcal{C}$ generates a random coin $c \in \{0,1\}$ so that $\Pr[c = 0] = 1/(q_S + n)$.
3. Algorithm $\mathcal{C}$ picks a random $b \in \mathbb{Z}_p$. If $c = 0$ holds, $\mathcal{C}$ computes $w \leftarrow h \cdot \psi(g_2)^b \in G_1$. If $c = 1$ holds, $\mathcal{C}$ computes $w \leftarrow \psi(g_2)^b \in G_1$.
4. Algorithm $\mathcal{C}$ adds the tuple $\langle M, w, b, c \rangle$ to the H-list and responds to $\mathcal{A}$ as $H(M) = w$.

Note that, either way, w is uniform in G_1 and is independent of $\mathcal{A}$'s current view as required.

Signature queries. Algorithm $\mathcal{A}$ requests a signature on some message M under the challenge key v_1. Algorithm $\mathcal{C}$ responds to this query as follows:

1. Algorithm $\mathcal{C}$ runs the above algorithm for responding to H-queries on M, obtaining the corresponding tuple $\langle M, w, b, c \rangle$ on the H-list. If $c = 0$ holds then $\mathcal{C}$ reports failure and terminates.
2. We know that $c = 1$ holds and hence $w = \psi(g_2)^b \in G_1$. Let $\sigma = \psi(u)^b \cdot \psi(g_2)^{rb} \in G_1$. Observe that $\sigma = w^{a+r}$ and therefore σ is a valid signature on M under the public key $v_1 = u \cdot g_2^r = g_2^{a+r}$. Algorithm $\mathcal{C}$ gives σ to algorithm $\mathcal{A}$.

Output. Finally, $\mathcal{A}$ halts. It either concedes failure, in which case so does $\mathcal{C}$, or it returns a value k (where $k \leq n$), $k - 1$ public keys $v_2, \dots, v_k \in G_2$, k messages $M_1, \dots M_k$, and a forged aggregate signature $\sigma \in G_1$. The messages M_i must all be distinct, and $\mathcal{A}$ must not have requested a signature on M_1. Algorithm $\mathcal{C}$ runs its hash algorithm at each M_i, $1 \leq i \leq k$, obtaining the k corresponding tuples $\langle M_i, w_i, b_i, c_i \rangle$ on the H-list.

Algorithm $\mathcal{C}$ now proceeds only if $c_1 = 0$ and, for $2 \le i \le k$, $c_i = 1$; otherwise $\mathcal{C}$ declares failure and halts. Since $c_1 = 0$, it follows that $w_1 = h \cdot \psi(g_2)^{b_1}$. For $i > 1$, since $c_i = 1$, it follows that $w_i = \psi(g_2)^{b_i}$. The aggregate signature σ must satisfy the aggregate verification equation, $e(\sigma, g_2) = \prod_{i=1}^{k} e(w_i, v_i)$. For each $i > 1$, $\mathcal{C}$ sets $\sigma_i \leftarrow \psi(v_i)^{b_i}$. Then, for $i > 1$,

$$e(\sigma_i, g_2) = e(\psi(v_i)^{b_i}, g_2) = e(\psi(v_i), g_2)^{b_i} = e(\psi(g_2), v_i)^{b_i} = e(\psi(g_2)^{b_i}, v_i) = e(w_i, v_i) \ ,$$

So σ_i is a valid signature on M_i (whose hash is w_i) by the key whose public component is v_i. Now $\mathcal{C}$ constructs a value σ_1: $\sigma_1 \leftarrow \sigma \cdot \left(\prod_{i=2}^{k} \sigma_i\right)^{-1}$. Then

$$e(\sigma_1, g_2) = e(\sigma, g_2) \cdot \prod_{i=2}^{k} e(\sigma_i, g_2)^{-1} = \prod_{i=1}^{k} e(w_i, v_i) \cdot \prod_{i=2}^{k} e(w_i, v_i)^{-1} = e(w_1, v_1) \ .$$

Thus σ_1 is a valid BLS signature by key $v_1 = u \cdot g_2^r = g_2^{a+r}$ on a message whose hash is $w_1 = h \cdot \psi(g_2)^{b_1}$. Then $\mathcal{C}$ calculates and outputs the required h^a as $h^a \leftarrow \sigma_1 \cdot (\psi(u)^{b_1} \cdot h^r \cdot \psi(g_2)^{rb_1})^{-1}$.

This completes the description of algorithm $\mathcal{C}$. It remains to show that $\mathcal{C}$ solves the given instance of the co-CDH problem in (G_1, G_2) with probability at least ϵ'. To do so, we analyze the three events needed for $\mathcal{C}$ to succeed:

$\mathcal{E}_1$**:** $\mathcal{C}$ does not abort as a result of any of $\mathcal{A}$'s signature queries.

$\mathcal{E}_2$**:** $\mathcal{A}$ generates a valid, nontrivial aggregate signature forgery $(k, v_2, \ldots, v_k, M_1, \ldots, M_k, \sigma)$.

$\mathcal{E}_3$**:** Event $\mathcal{E}_2$ occurs, and, in addition, $c_1 = 0$, and, for $2 \le i \le k$, $c_i = 1$, where for each i c_i is the c-component of the tuple containing M_i on the H-list.

$\mathcal{C}$ succeeds if all of these events happen. The probability $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3]$ decomposes as

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2]. \tag{4.1}$$

The following claims give a lower bound for each of these terms.

Claim 4. *The probability that algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}$'s aggregate signature queries is at least $(1 - 1/(q_S + n))^{q_S}$. Hence, $\Pr[\mathcal{E}_1] \ge (1 - 1/(q_S + n))^{q_S}$.*

Proof. Without loss of generality we assume that $\mathcal{A}$ does not ask for the signature of the same message twice. We prove by induction that after $\mathcal{A}$ makes l signature queries the probability that $\mathcal{C}$ does not abort is at least $(1-1/(q_S+n))^l$. The claim is trivially true for $l = 0$. Let $M^{(l)}$ be $\mathcal{A}$'s l'th signature query and let $\langle M^{(l)}, w^{(l)}, b^{(l)}, c^{(l)} \rangle$ be the corresponding tuple on the H-list. Then, prior to $\mathcal{A}$'s issuing the query, the bit $c^{(l)}$ is independent of $\mathcal{A}$'s view — the only value that could be given to $\mathcal{A}$ that depends on $c^{(l)}$ is $H(M^{(l)})$, but the distribution of $H(M^{(l)})$ is the same whether $c^{(l)} = 0$ or $c^{(l)} = 1$. Therefore, the probability that this query causes $\mathcal{C}$ to abort is at most $1/(q_S + n)$. Using the inductive hypothesis and the independence of $c^{(l)}$, the probability that $\mathcal{C}$ does not abort after this query is at least $(1 - 1/(q_S + n))^l$. This proves the inductive claim. Since $\mathcal{A}$ makes at most q_S signature queries the probability that $\mathcal{C}$ does not abort as a result of all signature queries is at least $(1 - 1/(q_S + n))^{q_S}$. $\square$

Claim 5. *If algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}$'s queries then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Hence, $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \ge \epsilon$.*

Proof. The public key given to $\mathcal{A}$ is from the same distribution as public keys produced by algorithm Kg. Responses to hash queries are as in the real attack since each response is uniformly and independently distributed in G_1. Since $\mathcal{C}$ did not abort as a result of $\mathcal{A}$'s signature queries, all its responses to those queries are valid. Therefore $\mathcal{A}$ will produce a valid and nontrivial aggregate signature forgery with probability at least ϵ. Hence $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$. $\square$

Claim 6. *The probability that algorithm $\mathcal{C}$ does not abort after $\mathcal{A}$ outputs a valid and nontrivial forgery is at least $(1 - 1/(q_S + n))^{n-1} \cdot 1/(q_S + n)$.*
Hence, $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq (1 - 1/(q_S + n))^{n-1} \cdot 1/(q_S + n)$.

Proof. Events $\mathcal{E}_1$ and $\mathcal{E}_2$ have occurred, and $\mathcal{A}$ has generated some valid and nontrivial forgery $(k, v_2, \ldots, v_k, M_1, \ldots, M_k, \sigma)$. For each i, $1 \leq i \leq k$, let $\langle M_i, w_i, b_i, c_i \rangle$ be the tuple corresponding to M_i on the H-list. Algorithm $\mathcal{C}$ will abort unless $\mathcal{A}$ generates a forgery such that $c_1 = 0$ and, for $i > 1$, $c_i = 1$.

Since all the messages $M_1, M_2, \ldots, M_k$ are distinct, the values $c_1, c_2, \ldots, c_k$ are all independent of each other; as before, $H(M_i) = w_i$ is independent of c_i for each i.

Since its forgery is nontrivial, $\mathcal{A}$ cannot have asked for a signature on M_1 under key v_1. It can thus have no information about the value of c_1; in the forged aggregate, $c_1 = 0$ occurs with probability $1/(q_S + n)$. For each $i > 1$, $\mathcal{A}$ either asked for a signature under key v_1 on M_i, in which case $c_i = 1$ with probability 1, or it didn't, and $c_i = 1$ with probability $1 - 1/(q_S + n)$. Regardless, the probability that $c_i = 1$ for all i, $2 \leq i \leq k$, is at least $(1 - 1/(q_S + n))^{k-1} \geq (1 - 1/(q_S + n))^{n-1}$.

Therefore $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq (1 - 1/(q_S + n))^{n-1} \cdot 1/(q_S + n)$, as required. $\square$

To complete the proof of the theorem, we use the bounds from the claims above in equation (4.1). Algorithm $\mathcal{C}$ produces the correct answer with probability at least

$$\left(1 - \frac{1}{q_S + n}\right)^{q_S + n - 1} \cdot \frac{1}{q_S + n} \cdot \epsilon \geq \frac{\epsilon / e}{q_S + n} \geq \epsilon' \ ,$$

as required.

Algorithm $\mathcal{C}$'s running time is the same as $\mathcal{A}$'s running time plus the time is takes to respond to $(q_H + q_S)$ hash queries and q_S signature queries, and the time to transform $\mathcal{A}$'s final forgery into the co-CDH solution. Each query requires an exponentiation in G_1. The output phase requires at most n additional hash computations, two inversions, two exponentiations, and $n + 1$ multiplications. We assume that exponentiation and inversion in G_1 take time c_{G_1}. Hence, the total running time is at most $t + c_{G_1}(q_H + 2q_S + n + 4) + n + 1 \leq t'$ as required. This completes the proof of the theorem. $\square$

Notes on the proof. If hash queries include the intended public key as well as the message — the countermeasure suggested above — then the challenger can set $c \leftarrow 1$ for any hash query $H(M, v)$ where v isn't v_1, the challenge key. Then all keys except the challenge key become irrelevant, and the reduction is precisely like that for BLS in Theorem 3.3.1; in particular, the multiplicative loss of security no longer depends on n.

One can also allow a single key to be responsible for several messages in an aggregate signature, and thus appear with multiplicity greater than 1 in the list $v_1, \ldots, v_k$, provided that no two keys sign the same message (more specifically, that the challenge key and some other key do not each sign a message with the same hash).

4.5 Verifiably Encrypted Signatures

We now show an application of aggregate signatures to verifiably encrypted signatures. These signatures are used in applications such as online contract signing [5, 9]. Suppose Alice wants to show Bob that she has signed a message, but does not want Bob to possess her signature of that message. (Alice will give her signature to Bob only when a certain event has occurred, e.g., Bob has given Alice his signature on the same message.) Alice can achieve this by encrypting her signature using the public key of a trusted third party, and sending this to Bob along with a proof that she has given him a valid encryption of her signature. Bob can verify that Alice has signed the message, but cannot deduce any information about her signature. Later in the protocol, if Alice is unable or unwilling to reveal her signature, Bob can ask the third party to reveal Alice's signature. We note that the resulting contract signing protocol is not abuse-free in the sense of Garay et al. [57].

We show that a variant of the bilinear aggregate signature scheme allows the creation of very efficient verifiably encrypted signatures. Previous constructions [5, 104] require zero knowledge proofs to verify an encrypted signature. The verifiably encrypted signatures in Section 4.5 are short and can be verified directly.

4.5.1 Verifiably Encrypted Signature Definitions

A verifiably encrypted signature scheme $\mathcal{VES}$ comprises seven algorithms. Three, Kg, Sig, and Vf, are analogous to those in ordinary signature schemes. The others, AKg, ESig, EVf, and Adj, provide the verifiably encrypted signature capability. The algorithms are described below. We refer to the trusted third party as the adjudicator.

VES.Kg, VES.Sig, VES.Vf. As in standard signature schemes.

VES.AKg. Generate a public-private key pair (apk, ask) for the adjudicator.

VES.ESig(sk, apk, M) Given a private key sk, an adjudicator's public key apk, and a message M, compute (probabilistically) a verifiably encrypted signature η on M.

VES.EVf(pk, apk, M, η)**.** Given a public key pk, an adjudicator's public key apk, a message M, and a verifiably encrypted signature η, verify that η is a valid verifiably encrypted signature on M under key pk.

VES.Adj(ask, pk, M, η)**.** Given an adjudicator's private key ask, a certified public key pk, and a verifiably encrypted signature η on some message M, extract and output σ, an ordinary signature on M under key pk.

Besides the ordinary notions of signature security in the signature component, we require three security properties of verifiably encrypted signatures: validity, unforgeability, and opacity. We describe these properties below, in the single-user setting.

Validity requires that verifiably encrypted signatures verify, and that adjudicated verifiably encrypted signatures verify as ordinary signatures, i.e., that $\mathsf{EVf}(M, \mathsf{ESig}(M))$ and $\mathsf{Vf}(M, \mathsf{Adj}(M, \mathsf{ESig}(M)))$ accept for all M and for all properly-generated keypairs and adjudicator keypairs. (The keys provided to the algorithms are here elided for brevity.)

Unforgeability requires that it be difficult to forge a valid verifiably encrypted signature. The advantage in existentially forging a verifiably encrypted signature of an algorithm $\mathcal{F}$ is

$$\mathbf{Adv}_{\mathcal{VES},\mathcal{F}}^{\text{ves-forge}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} \mathsf{EVf}(pk, apk, M, \eta) = \texttt{valid} : \\ \qquad\qquad\qquad (pk, sk) \xleftarrow{\text{R}} \mathsf{Kg}, \\ \qquad\qquad\qquad (apk, ask) \xleftarrow{\text{R}} \mathsf{AKg}, \\ \qquad\qquad\qquad (M, \eta) \xleftarrow{\text{R}} \mathcal{F}^{S,A}(pk, apk) \end{array}\right].$$

The probability is taken over the coin tosses of the key-generation algorithms, of the oracles, and of the forger. The forger is given access to a verifiably-encrypted–signature creation oracle $S = \mathsf{ESig}(sk, apk, \cdot)$ and an adjudication oracle $A = \mathsf{Adj}(ask, pk, \cdot, \cdot)$, along with a hash oracle. The forger $\mathcal{F}$ is additionally constrained in that its forgery on M must be nontrivial: $\mathcal{F}$ must not previously have queried either oracle at M. Note that an ordinary signing oracle is not provided; it can be simulated by a call to S followed by a call to A.

Definition 4.5.1. A verifiably encrypted signature forger $\mathcal{F}$ $(t, q_H, q_S, q_A, \epsilon)$-forges a verifiably encrypted signature if: Algorithm $\mathcal{F}$ runs in time at most t; $\mathcal{F}$ makes at most q_H queries to the hash function, at most q_S queries to the verifiably-encrypted–signature creation oracle S, at most q_A queries to the adjudication oracle A; and $\mathbf{Adv}_{\mathcal{VES},\mathcal{F}}^{\text{ves-forge}}$ is at least ϵ. A verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against existential forgery if no forger $(t, q_H, q_S, q_A, \epsilon)$-breaks it.

Opacity requires that it be difficult, given a verifiably encrypted signature, to extract an ordinary signature on the same message. The advantage in extracting a verifiably encrypted signature of an algorithm $\mathcal{E}$, again given access to a verifiably-encrypted–signature creation oracle S and an adjudication oracle A, along with a hash oracle, is

$$\mathbf{Adv}_{\mathcal{VES},\mathcal{E}}^{\text{ves-extr}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} \mathsf{Vf}(pk, M, \sigma) = \texttt{valid} : \\ \qquad\qquad\qquad (pk, sk) \xleftarrow{\text{R}} \mathsf{Kg}, \\ \qquad\qquad\qquad (apk, ask) \xleftarrow{\text{R}} \mathsf{AKg}, \\ \qquad\qquad\qquad (M, \sigma) \xleftarrow{\text{R}} \mathcal{E}^{S,A}(pk, apk) \end{array}\right].$$

The probability is taken over the coin tosses of the key-generation algorithms, of the oracles, and of the forger. The extraction must be nontrivial: the adversary must not have queried the adjudication oracle A at M. (It is allowed, however, to query S at M.) Verifiably encrypted signature extraction is clearly no more difficult than forgery in the underlying signature scheme.

Definition 4.5.2. An algorithm $\mathcal{E}$ $(t, q_H, q_S, q_A, \epsilon)$-extracts a verifiably encrypted signature if $\mathcal{E}$ runs in time at most t, makes at most q_H queries to the hash function, at most q_S queries to the verifiably-encrypted–signature creation oracle S, at most q_A queries to the adjudication oracle, and $\mathbf{Adv}_{\mathcal{VES},\mathcal{E}}^{\text{ves-extr}}$ is at least ϵ. A verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against extraction if no algorithm $(t, q_H, q_S, q_A, \epsilon)$-extracts it.

4.5.2 Aggregate Extraction

Our verifiably encrypted signature scheme depends on the assumption that given an aggregate signature of k signatures it is difficult to extract the individual signatures.

Consider the bilinear aggregate signature scheme on a group pair (G_1, G_2). We posit that it is difficult to recover the individual signatures σ_i given their aggregate σ, the public keys, and the message hashes. In fact, we posit that it is difficult to recover an aggregate σ' of any proper subset of the signatures. This we term the k-element aggregate extraction problem.

We formalize this assumption as follows. Let (G_1, G_2) be a bilinear group pair for co-Diffie-Hellman, each of order p, with respective generators g_1 and g_2, a computable isomorphism $\psi : G_2 \to G_1$ such that $g_1 = \psi(g_2)$, and a computable bilinear map $e : G_1 \times G_2 \to G_T$.

Consider a k-user aggregate in this setting. Each user has a private key $x_i \in \mathbb{Z}_p$ and a public key $v_i = g_2^{x_i} \in G_2$. Each user selects a distinct message $M_i \in \{0,1\}^*$ whose hash is $h_i \in G_1$ and creates a signature $\sigma_i = h_i^{x_i} \in G_1$. Finally, the signatures are aggregated, yielding $\sigma = \prod_i \sigma_i \in G_1$.

Let I be the set $\{1, \ldots, k\}$. Each public key v_i can be expressed as $g_2^{x_i}$, each hash h_i as $g_1^{y_i}$, each signature σ_i as $g_1^{x_i y_i}$, and the aggregate signature σ as g_1^z, where $z = \sum_{i \in I} x_i y_i$. The advantage of an algorithm $\mathcal{E}$ in extracting a sub-aggregate from a k-element aggregate is

$$\mathbf{Adv}_{\mathcal{E}}^{\text{aggr-extr}}(k) \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} (\emptyset \neq I' \subsetneq I) \wedge (\sigma' = g_1^{(\sum_{i \in I'} x_i y_i)}) : \\ \qquad\qquad x_1, \ldots, x_k, y_1, \ldots, y_k \xleftarrow{\text{R}} \mathbb{Z}_p,\ \sigma \leftarrow g_1^{(\sum_{i \in I} x_i y_i)}, \\ \qquad\qquad (\sigma', I') \xleftarrow{\text{R}} \mathcal{E}\big(g_2^{x_1}, \ldots, g_2^{x_k}, g_1^{y_1}, \ldots, g_1^{y_k}, \sigma\big) \end{array}\right].$$

The probability is taken over the choices of all x_i and y_i, and the coin tosses of $\mathcal{E}$.

Definition 4.5.3. An algorithm $\mathcal{E}$ (t, k, ϵ)-extracts a sub-aggregate from an k-element BGLS aggregate signature if $\mathcal{E}$ runs in time at most t and $\mathbf{Adv}_{\mathcal{E}}^{\text{aggr-extr}}(k)$ is at least ϵ. An instantiation of the BGLS aggregate signature scheme is (t, k, ϵ)-secure against aggregate extraction if no algorithm (t, k, ϵ)-extracts it.

Coron and Naccache have shown that the k-element aggregate extraction assumption is equivalent to co-CDH [42].

4.5.3 Verifiably Encrypted Signatures via Aggregation

We motivate our construction for verifiably encrypted signatures by considering aggregate signatures as a launching point. An aggregate signature scheme can give rise to a verifiably encrypted signature scheme if it is difficult to extract individual signatures from an aggregate, but easy to forge existentially under the adjudicator's key. Consider the following:

1. Alice wishes to create a verifiably encrypted signature, which Bob will verify; Carol is the adjudicator. Alice and Carol's keys are both generated under the underlying signature scheme's key-generation algorithm.

2. Alice creates a signature σ on M under her public key. She forges a signature σ' on some random message M' under Carol's public key. She then combines σ and σ', obtaining an aggregate η. The verifiably encrypted signature is the pair (η, M').

3. Bob validates Alice's verifiably encrypted signature (η, M') on M by checking that η is a valid aggregate signature by Alice on M and by Carol on M'.

4. Carol adjudicates, given a verifiably encrypted signature (η, M') on M by Alice, by computing a signature σ' on M' under her key, and removing σ' from the aggregate; what remains is Alice's ordinary signature σ.

In the BGLS aggregate signature scheme, it is difficult to extract individual signatures, under the aggregate extraction assumption. Moreover, existential forgery is easy when the random oracle hash function is set aside: Given a public key $v \in G_2$ and $r \in \mathbb{Z}_p$, $\psi(v)^r$ is a valid signature on a message whose hash is $\psi(g_2)^r = g_1^r$. Below, we describe formally and give a security proof for the verifiably encrypted signature scheme obtained in this way.

4.5.4 Verifiably-Encrypted Signatures from Bilinear Maps

The BGLS2 verifiably encrypted signature scheme is built on the BGLS aggregate signature scheme described in Section 4.4.2. It shares the key-generation algorithm with the underlying aggregate scheme. Moreover, the adjudicator's public and private information is simply an aggregate-signature keypair. The scheme comprises the seven algorithms described below:

BGLS2.Kg, BGLS2.AKg. Kg and AKg are both the same as BLS.Kg, the key generation algorithm of the BLS signature scheme (and thus also the same as BGLS.Kg).

BGLS2.Sig, BGLS2.Vf. Sig and Vf are the same as BLS.Sig and BLS.Vf, respectively.

BGLS2.ESig(sk, apk, M) Parse the user's private key sk as $x \in \mathbb{Z}_p$ and the adjudicator's public key apk as $v' \in G_2$. To sign the message $M \in \{0,1\}^*$, compute $h \leftarrow H(M) \in G_1$ and $\sigma \leftarrow h^x$. Select r at random from $\mathbb{Z}_p$ and set $\mu \leftarrow \psi(g_2)^r$ and $\sigma' \leftarrow \psi(v')^r$. Aggregate σ and σ' as $\omega \leftarrow \sigma\sigma' \in G_1$. The verifiably encrypted signature η is the pair (ω, μ). (It can also be viewed as an ElGamal encryption of σ under the adjudicator's key.)

BGLS2.EVf(pk, apk, M, η)**.** Parse the user's public key pk as $v \in G_2$, the adjudicator's public key apk as $v' \in G_2$, and the verifiably encrypted signature η as $(\omega, \mu) \in G_1^2$. Set $h \leftarrow H(M)$; accept if $e(\omega, g_2) = e(h, v) \cdot e(\mu, v')$ holds.

BGLS2.Adj(ask, pk, M, η)**.** Parse the adjudicator's private key ask as $x' \in \mathbb{Z}_p$. Parse the user's public key pk as $v \in G_2$, and check that it has been certified. Verify (using EVf) that the verifiably encrypted signature η is valid, and parse it as $(\omega, \mu) \in G_1^2$. Finally, output $\sigma = \omega/\mu^{x'} \in G_1$.

Potential Attacks. If the adjudicator does not first validate a purported verifiably encrypted signature, a malicious user can trick him into signing arbitrary messages under his adjudication key.

Similarly, the adjudicator should only adjudicate for certified public keys pk; we assume that the CA, in issuing a certificate on pk, verifies that the user knows the corresponding private key. Certification is necessary to prevent an attack devised by Hess [68] that arises if the adjudicator is willing to open signatures encrypted under keys generated by the attacker. In this attack, the adversary obtains from the signing oracle a verifiably encrypted signature (ω, μ) under the challenge key v. He computes from it (ω^r, μ^r) for some r, which is a verifiably encrypted signature on the same message under the related key v^r. He then queries the adjudication oracle and obtains the underlying signature σ^r, from which he can compute σ. Note that the adversary does not know the private key corresponding to v^r, and that the adjudication oracle must be willing to answer queries for keys other than the challenge key v.

Another countermeasure, suggested by Hess, is for users to tie the verifiably encrypted signatures to their identity by computing the hash h as $H(M, v)$ rather than $H(M)$.

4.5.5 Proofs of Security

It is easy to see that the BGLS2 scheme is valid. A verifiably encrypted signature correctly validates under EVf, which is simply the aggregate signature verification algorithm. Moreover, for any valid verifiably encrypted signature, $e(\omega/\mu^{x'}, g_2) = e(\omega, g_2) \cdot e(\mu, g_2)^{-x'} = e(h, v) \cdot e(\mu, v') \cdot e(\mu, v')^{-1} = e(h, v)$, so the output of Adj is a valid signature on message M under the key v.

The next two theorems prove the unforgeability and opacity of the scheme.

Theorem 4.5.4. *Let G_1 and G_2 be cyclic groups of prime order p, with respective generators g_1 and g_2, with a computable bilinear map $e : G_1 \times G_2 \to G_T$. Suppose that the BLS signature scheme is $(t', q'_H, q'_S, \epsilon')$-secure against existential forgery on (G_1, G_2). Then BGLS2 is $(t, q_H, q_S, q_A, \epsilon)$-secure against existential forgery on (G_1, G_2), for all $q_H \le q'_H$, $q_S \le q'_S$, $\epsilon \ge \epsilon'$, and all t satisfying $t \le t' - 2c_{G_1}(q_S + q_A + 1)$, where exponentiation and inversion on G_1 take time c_{G_1}.*

Proof. Given a BGLS2 forger algorithm $\mathcal{V}$, we construct a forger algorithm $\mathcal{F}$ for the underlying BLS signature scheme.

We assume that $\mathcal{V}$ is well-behaved in the sense that it always requests the hash of a message M before it requests a verifiably encrypted signature or an adjudication involving M, and that it never requests adjudication on a message M on which it had not previously asked for a verifiably encrypted signature. It is trivial to modify any forger algorithm $\mathcal{V}$ to have the first property. The second property is reasonable since the input to the adjudication oracle in this case would be a nontrivial verifiably encrypted signature forgery; $\mathcal{V}$ can be modified simply to output it and halt.

The BLS forger $\mathcal{F}$ is given a public key v, and has access to a signing oracle for v and a hash oracle. It simulates the challenger and runs interacts with $\mathcal{V}$ as follows.

Setup. Algorithm $\mathcal{F}$ generates a key, $(x', v') \xleftarrow{\text{R}} \mathsf{Kg}$, which serves as the adjudicator's key. Now $\mathcal{F}$ runs $\mathcal{V}$, providing as input the public keys v and v'.

Hash Queries. Algorithm $\mathcal{V}$ requests a hash on some string M. Algorithm $\mathcal{F}$ makes a query on M to its own hash oracle, receiving some value $h \in G_1$, with which it responds to $\mathcal{V}$'s query.

VerSig Creation Queries. Algorithm $\mathcal{V}$ requests a signature on some string M. (It will have already queried the hash oracle at M.) $\mathcal{F}$ queries its signing oracle (for v) at M, obtaining $\sigma \in G_1$. It then selects r at random from $\mathbb{Z}_p$, and returns to $\mathcal{V}$ the pair $(\sigma \cdot \psi(v')^r, \psi(g_2)^r)$.

Adjudication Queries. Algorithm $\mathcal{V}$ requests adjudication for (ω, μ), a verifiably encrypted signature on a message M under key v and adjudicator key v'. Algorithm $\mathcal{F}$ checks that the verifiably encrypted signature is valid, then returns $\omega/\mu^{x'}$.

Output. Finally, $\mathcal{V}$ halts, either declaring failure, in which case $\mathcal{F}$, too, declares failure and halts, or providing a valid and nontrivial verifiably encrypted signature (ω^*, μ^*) on a message M^*. $\mathcal{F}$ sets $\sigma^* \leftarrow \omega^*/(\mu^*)^{x'}$. We thus have

$$\begin{aligned} e(\sigma^*, g_2) &= e(\omega^*/(\mu^*)^{x'}, g_2) = e(\omega^*, g_2) \;/\; e(\mu^*, g_2^{x'}) \\ &= e(H(M^*), v) \cdot e(\mu^*, v') \;/\; e(\mu^*, v') = e(H(M^*), v) \;\;, \end{aligned}$$

so σ^* is a valid BLS signature on M^* under key v. (In the second line, we use the fact that (ω^*, μ^*) is a valid verifiably encrypted signature, so the verification equation $e(\omega^*, g_2) =$

$e(H(M^*), v) \cdot e(\mu^*, v')$ holds.) Since the forgery is nontrivial, $\mathcal{V}$ must not have queried the verifiably encrypted signature oracle at M^*, from which it follows that $\mathcal{F}$ did not query its signing oracle at M^*. Thus (M^*, σ^*) is a nontrivial BLS forgery; algorithm $\mathcal{F}$ outputs it and halts.

It remains only to analyze the success probability and running time of $\mathcal{F}$. Algorithm $\mathcal{F}$ succeeds whenever $\mathcal{V}$ does, that is, with probability at least ϵ.

Algorithm $\mathcal{F}$'s running time is the same as $\mathcal{V}$'s running time plus the time it takes to respond to q_H hash queries, q_S verifiably-encrypted signature queries, and q_A adjudication queries, and the time to transform $\mathcal{V}$'s final verifiably-encrypted signature forgery into a BLS signature forgery. Hash queries impose no overhead. Each verifiably-encrypted signature query requires $\mathcal{F}$ to perform two exponentiations in G_1. Each adjudication query requires $\mathcal{F}$ to perform an exponentiation and an inversion in G_1. The output phase also requires an exponentiation and an inversion. We assume that exponentiation and inversion in G_1 take time c_{G_1}. Hence, the total running time is at most $t + 2c_{G_1}(q_S + q_A + 1)$.

$\mathcal{F}$ queries its hash oracle whenever $\mathcal{V}$ queries its hash oracle, and its signing oracle whenever $\mathcal{V}$ queries its verifiably encrypted signature oracle.

Combining all this, we see that if $\mathcal{V}$ $(t, q_H, q_S, q_A, \epsilon)$-forges a bilinear verifiably encrypted signature on (G_1, G_2), then $\mathcal{F}$ $(t + 2c_{G_1}(q_S + q_A + 1), q_H, q_S, \epsilon)$-breaks the BLS signature scheme on (G_1, G_2). Conversely, if the BLS signature scheme is $(t', q'_H, q'_S, \epsilon')$-secure, then BGLS2 is $(t' - 2c_{G_1}(q_S + q_A + 1), q'_H, q'_S, q_A, \epsilon')$-secure against existential forgery. □

Note that the proof above treats the underlying signature scheme as a black box, except for two properties: (1) that it is possible to encrypt a signature to obtain a verifiably encrypted signature, and (2) that the adjudication procedure transforms all valid verifiably encrypted signatures into valid (unencrypted) signatures — not just those that ESig would output, as required by the validity property.

Theorem 4.5.5. *Let G_1 and G_2 be cyclic groups of prime order p, with respective generators g_1 and g_2, with a computable isomorphism $\psi : G_2 \to G_1$ such that $\psi(g_2) = g_1$ and a computable bilinear map $e : G_1 \times G_2 \to G_T$. Suppose that the bilinear aggregate signature scheme on (G_1, G_2) is $(t', 2, \epsilon')$-secure against aggregate extraction. Then BGLS2 is $(t, q_H, q_S, q_A, \epsilon)$-secure against extraction on (G_1, G_2) for all t and ϵ satisfying*

$$\epsilon \geq e(q_A + 1) \cdot \epsilon' \qquad \textit{and} \qquad t \leq t' - c_{G_1}(q_H + 4q_S + 2q_A + 3) \ ,$$

where e is the base of natural logarithms, and exponentiation and inversion on G_1 take time c_{G_1}.

Proof. Given a verifiably-encrypted–signature extractor algorithm $\mathcal{V}$, we construct an aggregate extractor algorithm $\mathcal{A}$. The extractor $\mathcal{A}$ is given values g_2^α and g_2^β in G_2, g_1^γ, g_1^δ, and $g_1^{\alpha\gamma+\beta\delta}$ in G_1. It runs $\mathcal{V}$, answering its oracle calls, and uses $\mathcal{V}$'s verifiably encrypted signature extraction to calculate $g_1^{\alpha\gamma}$, the answer to its own extraction challenge.

Let g_1 be a generator of G_1, and g_2 of G_2, such that $\psi(g_2) = g_1$. Algorithm $\mathcal{A}$ is given $g_2^\alpha, g_2^\beta \in G_2$ and $g_1^\gamma, g_1^\delta, g_1^{\alpha\gamma+\beta\delta} \in G_1$. Its goal is to output $g_1^{\alpha\gamma} \in G_1$. Algorithm $\mathcal{A}$ simulates the challenger and interacts with verifiably-encrypted–signature extractor $\mathcal{V}$ as follows.

Setup. Algorithm $\mathcal{A}$ sets $v \leftarrow g_2^\alpha$, the signer's public key, and $v' \leftarrow g_2^\beta$, the adjudicator's public key. It gives v and v' to $\mathcal{V}$.

Hash Queries. At any time algorithm $\mathcal{V}$ can query the random oracle H. To respond to these queries, $\mathcal{A}$ maintains a list of tuples $\langle M^{(i)}, w^{(i)}, b^{(i)}, c^{(i)} \rangle$ as explained below. We refer to this list as the H-list. The list is initially empty. When $\mathcal{V}$ queries the oracle H at a point $M \in \{0,1\}^*$, algorithm $\mathcal{A}$ responds as follows:

1. If the query M already appears on the H-list in some tuple $\langle M, w, b, c \rangle$ then algorithm $\mathcal{A}$ responds with $H(M) = w \in G_1$.
2. Otherwise, $\mathcal{A}$ generates a random coin $c \in \{0,1\}$ so that $\Pr[c = 0] = 1/(q_A + 1)$.
3. Algorithm $\mathcal{A}$ picks a random $b \in \mathbb{Z}_p$. If $c = 0$ holds, $\mathcal{A}$ computes $w \leftarrow g_1^\gamma \cdot g_1^b \in G_1$. If $c = 1$ holds, $\mathcal{A}$ computes $w \leftarrow g_1^b \in G_1$.
4. Algorithm $\mathcal{A}$ adds the tuple $\langle M, w, b, c \rangle$ to the H-list and responds to $\mathcal{V}$ as $H(M) = w$.

VerSig Creation Queries. $\mathcal{V}$ requests a verifiably-encrypted signature on some string M under challenge key v and adjudicator key v'. Algorithm $\mathcal{A}$ responds to this query as follows:

1. Algorithm $\mathcal{A}$ runs the above algorithm for responding to H-queries on M, obtaining the corresponding tuple $\langle M, w, b, c \rangle$ on the H-list.
2. $\mathcal{A}$ selects x at random from $\mathbb{Z}_p$. If c equals 0, $\mathcal{A}$ computes and returns $(\omega, \mu) = (\psi(g_2^\alpha)^b \cdot g_1^{\alpha\gamma+\beta\delta} \cdot \psi(g_2^\beta)^x, g_1^\delta \cdot g_1^x)$. If c equals 1, $\mathcal{A}$ computes and returns $(\omega, \mu) = (\psi(g_2^\alpha)^b \cdot \psi(g_2^\beta)^x, g_2^x)$. It is easy to verify that (ϵ, μ) is in either case a correct verifiably encrypted signature on the message with hash w.

Adjudication Queries. Algorithm $\mathcal{V}$ requests adjudication for (ω, μ), a verifiably encrypted signature on a message M under key v and adjudicator key v'. Algorithm $\mathcal{A}$ responds to this query as follows:

1. Algorithm $\mathcal{A}$ runs the above algorithm for responding to H-queries on M, obtaining the corresponding tuple $\langle M, w, b, c \rangle$ on the H-list.
2. Algorithm $\mathcal{A}$ checks that the verifiably encrypted signature is valid. If it is not, $\mathcal{A}$ returns $\star$, a placeholder value.
3. If c equals 0, $\mathcal{A}$ declares failure and halts. Otherwise, it computes and returns $\sigma \leftarrow \psi(g_2^\alpha)^b$. It is easy to verify that σ is the correct BLS signature under key v on the message with hash w.

Output. Finally, $\mathcal{V}$ halts. It either concedes failure, in which case so does $\mathcal{A}$, or returns a nontrivial extracted signature σ^* on some message M^*. For the extraction to be nontrivial, $\mathcal{V}$ must not have asked for adjudication on a verifiably encrypted signature of M^*. Algorithm $\mathcal{A}$ runs its hash algorithm at M^*, obtaining the k corresponding tuples $\langle M^*, w^*, b^*, c^* \rangle$ on the H-list.

$\mathcal{A}$ now proceeds only if $c^* = 0$; otherwise it declares failure and halts. Since $c^* = 0$, it follows that $w^* = g_1^\gamma \cdot g_1^{b^*}$. The extracted signature σ^* must satisfy the BLS verification equation, $e(\sigma^*, g_2) = e(h^*, v)$. $\mathcal{A}$ sets $\sigma \leftarrow \sigma^*/\psi(v)^{b^*}$. Then

$$\begin{aligned} e(\sigma, g_2) &= e(\sigma^*, g_2) \cdot e(\psi(v), g_2)^{-b^*} = e(w^*, v) \cdot e(\psi(g_2), v)^{-b^*} \\ &= e(g_1^\gamma, v) \cdot e(g_1, v)^{b^*} \cdot e(g_1, v)^{-b^*} = e(g_1^\gamma, g_2^\alpha). \end{aligned}$$

Where in the last equality we substitute $v = g_2^{\alpha}$. Thus $(g_2, g_2^{\alpha}, g_1^{\gamma}, \sigma)$ is a valid co-Diffie-Hellman tuple, so σ equals $g_2^{\alpha\gamma}$, the answer to the aggregate extraction problem; algorithm $\mathcal{A}$ outputs it and halts.

This completes the description of algorithm $\mathcal{A}$. It remains to show that $\mathcal{A}$ solves the given instance of the aggregate extraction problem on (G_1, G_2) with probability at least ϵ'. To do so, we analyze the three events needed for $\mathcal{A}$ to succeed:

$\mathcal{E}_1$: $\mathcal{A}$ does not abort as a result of any of $\mathcal{V}$'s adjudication queries.

$\mathcal{E}_2$: $\mathcal{V}$ generates a valid and nontrivial verifiably-encrypted signature extraction (M^*, σ^*).

$\mathcal{E}_3$: Event $\mathcal{E}_2$ occurs, and $c^* = 0$ holds, where c^* is the c-component of the tuple containing M^* on the H-list.

$\mathcal{A}$ succeeds if all of these events happen. The probability $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3]$ decomposes as

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2]. \tag{4.2}$$

The following claims give a lower bound for each of these terms.

Claim 7. *The probability that algorithm $\mathcal{A}$ does not abort as a result of $\mathcal{V}$'s adjudication queries is at least $1/e$. Hence, $\Pr[\mathcal{E}_1] \geq 1/e$.*

Proof. Without loss of generality we assume that $\mathcal{V}$ does not ask for adjudication of the same message twice. We prove by induction that after $\mathcal{V}$ makes l signature queries the probability that $\mathcal{A}$ does not abort is at least $(1 - 1/(q_A + 1))^l$. The claim is trivially true for $l = 0$. Let $\mathcal{V}$'s l'th adjudication query be for verifiably encrypted signature $(\omega^{(l)}, \mu^{(l)})$, on message $M^{(l)}$ under the challenge key v, and let $\langle M^{(l)}, w^{(l)}, b^{(l)}, c^{(l)} \rangle$ be the corresponding tuple on the H-list. Then prior to issuing the query, the bit $c^{(l)}$ is independent of $\mathcal{V}$'s view — the only values that could be given to $\mathcal{V}$ that depend on $c^{(l)}$ are $H(M^{(l)})$ and verifiably-encrypted signatures on $M^{(l)}$, but the distributions on these values are the same whether $c^{(l)} = 0$ or $c^{(l)} = 1$. Therefore, the probability that this query causes $\mathcal{A}$ to abort is at most $1/(q_A + 1)$. Using the inductive hypothesis and the independence of $c^{(l)}$, the probability that $\mathcal{A}$ does not abort after this query is at least $(1 - 1/(q_A + 1))^l$. This proves the inductive claim. Since $\mathcal{V}$ makes at most q_A adjudication queries the probability that $\mathcal{A}$ does not abort as a result of all signature queries is at least $(1 - 1/(q_A + 1))^{q_A} \geq 1/e$. $\square$

Claim 8. *If algorithm $\mathcal{A}$ does not abort as a result of $\mathcal{V}$'s adjudication queries then $\mathcal{V}$'s view is identical to its view in the real attack. Hence, $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.*

Proof. The challenge public key v given to $\mathcal{V}$ is from the same distribution as public keys produced by Kg; the adjudicator's public key v' given to $\mathcal{V}$ is from the same distribution as the adjudicator keys produces by AKg. Responses to hash queries are as in the real attack since each response is uniformly and independently distributed in G_1. Responses to verifiably-encrypted signature queries are also as in the real attack: They are valid, and their μ components are uniformly and independently distributed in G_1. Since $\mathcal{A}$ did not abort as a result of $\mathcal{V}$'s adjudication queries, all its responses to those queries are valid. Therefore $\mathcal{V}$ will produce a valid and nontrivial verifiably-encrypted signature extraction with probability at least ϵ. Hence $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$. $\square$

Claim 9. *The probability that algorithm $\mathcal{A}$ does not abort after $\mathcal{V}$ outputs a valid and nontrivial verifiably-encrypted signature extraction is at least* $1/(q_A+1)$ *Hence,* $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(q_A+1)$.

Proof. Given that events $\mathcal{E}_1$ and $\mathcal{E}_2$ happened, algorithm $\mathcal{A}$ will abort only if $\mathcal{V}$ generates a forgery (M^*, σ^*) for which the tuple $\langle M^*, w^*, b^*, c^* \rangle$ on the H-list has $c = 1$. Since its extraction is nontrivial, $\mathcal{V}$ could not have requested adjudication on any verifiably encrypted signature on M^*, and c^* must be independent of $\mathcal{V}$'s current view. Therefore $\Pr[c^* = 0 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(q_A+1)$ as required. □

Using the bounds from the claims above in equation (4.2) shows that $\mathcal{A}$ produces the correct answer with probability at least $\epsilon/e(q_A+1) \geq \epsilon'$ as required.

Algorithm $\mathcal{A}$'s running time is the same as $\mathcal{V}$'s running time plus the time is takes to respond to $\mathcal{A}$'s oracle queries and to transform $\mathcal{V}$'s verifiably-encrypted signature extraction into an aggregate extraction. Each verifiably-encrypted signature query, each adjudication query, and the output phase requires $\mathcal{A}$ to run its H-algorithm. It must therefore run this algorithm $(q_H+q_S+q_A+1)$ times. Each run requires an exponentiation in G_1. Algorithm $\mathcal{A}$ must run its verifiably-encrypted signing algorithm q_S times, and each run requires at most three exponentiation in G_1. Finally, $\mathcal{A}$'s output phase requires at most one exponentiation and one inversion in G_1. We assume that exponentiation and inversion in G_1 take time c_{G_1}. Hence, the total running time is at most $t + c_{G_1}(q_H + 4q_S + 2q_A + 3) \leq t'$ as required. □

4.5.6 Observations on Verifiably Encrypted Signatures

We note some extensions of the BGLS2 verifiably encrypted signature scheme discussed above.

- Anyone can convert an ordinary unencrypted signature to a verifiably encrypted signature. The same applies to unencrypted aggregate signatures.
- An adjudicator's private key can be shared amongst n parties using k-of-n threshold cryptography [62, 60], so that k parties are needed to adjudicate a verifiably encrypted signature.
- A message-signature pair in the BLS signature scheme is of the same form as an identity–private-key pair in Boneh-Franklin Identity-Based Encryption (IBE) [25]. Thus the verifiably encrypted signature scheme can potentially be modified to yield a verifiable encryption scheme for IBE private keys. Verifiably encrypted private keys have many applications [104].

4.6 Conclusions and Open Problems

In this chapter, we have some variants and extensions of which the BLS signature scheme is capable.

We first showed how BLS has been extended to construct many of the standard signature variants in the literature, including threshold signatures and multisignatures.

We then introduced the concept of aggregate signatures and constructed an efficient aggregate signature scheme, BGLS, based on bilinear maps. Key generation, aggregation, and verification require no interaction. We proved security of the system in a model that gives the adversary his choice of public keys and messages to forge.

Finally, we introduced verifiably encrypted signatures, and showed that our BGLS aggregate signature scheme gives rise to a simple verifiably encrypted signature scheme.

Unlike most previous signature constructions using bilinear maps [79, 48, 22], which only required a gap Diffie-Hellman group (i.e., DDH easy, CDH hard), the aggregate and verifiably encrypted signature constructions described in Sections 4.4.2 and 4.5.4 require the extra structure provided by the bilinear map. These constructions are an example where a bilinear map provides more power than a generic gap Diffie-Hellman group.

It is an open problem to construct aggregate signatures that work in any gap Diffie-Hellman group.

Chapter 5

Sequential Aggregate Signatures from Trapdoor Permutations

5.1 Introduction

Aggregate signatures, as introduced in Section 4.4, allow a single short aggregate to replace n signatures by n users on n messages. The BGLS aggregate signature scheme, like the other schemes in this thesis, requires a computable bilinear map. In this section we propose a variant — *sequential aggregate signatures* — with the advantage that it can be constructed based on a more general assumption: the existence of trapdoor permutations.

There is a trade-off. Sequential aggregate signatures do not have the property that an untrusted aggregating party can combine, after the fact, signatures that were generated independently. Instead, signing and aggregation are a single operation, and each signer folds her signature into the aggregate-so-far, which she obtains from the previous signer. More precisely, User i is given an aggregate on messages $M_1, \ldots, M_{i-1}$ under keys $pk_1, \ldots, pk_{i-1}$ and outputs an aggregate on messages $M_1, \ldots, M_{i-1}, M_i$ under keys $pk_1, \ldots, pk_{i-1}, pk_i$.

For some applications of aggregate signatures — in particular, certificate chains — the ability to combine preexisting individual signatures into an aggregate is unnecessary. Each CA, when producing a signature, has already obtained the signatures of CAs above it in the chain. Thus aggregation for certificate chains can be performed incrementally and sequentially.

In this chapter, we show how to realize sequential aggregate signatures using any family of certified[1] trapdoor permutations over a single domain, provided that the domain is a group under some operation. We prove security (with an exact security analysis) of our construction in the random oracle model; we give a tighter security guarantee for the special cases of *homomorphic* and *claw-free* trapdoor permutations.

Again, compared to the BGLS aggregate signature scheme considered in Section 4.4, the scheme presented here places more restrictions on the signers (because of the sequentiality requirement), but relies on a more general assumption.

We also show how to instantiate our construction with the RSA trapdoor permutation. This instantiation turns out to be more difficult than may be expected, because of the possibility of maliciously generated RSA keys: We need to provide security for User i regardless of whether other

[1]A trapdoor permutation is *certified* [19] if one can verify from its public description that it is actually a permutation.

users are honest. There are essentially four problems. The first is that our scheme assumes multiple trapdoor permutations over the same domain, which RSA does not provide. The second is that RSA is not a *certified* trapdoor permutation: for a maliciously generated public-key, it can indeed be very far from a permutation. The third is that the domain of RSA is not the convenient $\mathbb{Z}_N$, but rather $\mathbb{Z}_N^*$, which can be much smaller for maliciously generated N. Finally, the natural group operation on $\mathbb{Z}_N^*$ (multiplication) is not a group operation on $\mathbb{Z}_N$. We overcome these problems with techniques that may be of independent interest. In particular, we turn RSA into a *certified* trapdoor permutation over *all* of $\mathbb{Z}_N$.

Other Related Work. Aggregate signatures are related to multisignatures, which we considered in Section 4.3. In particular, our LMRS aggregate signature scheme has similarities with the multisignature scheme of Okamoto [97] (though the latter has no security proof and, indeed, is missing important details that would make the security proof possible, as shown by Micali et al. [86]).

5.2 Preliminaries

We recall the definitions of trapdoor permutations and the full-domain hash signatures based upon them. We also define certified trapdoor permutations, which are needed for building sequential aggregate signatures. Finally, we define claw-free permutations and homomorphic trapdoor permutations, whose properties we will use to achieve a better security reduction.

5.2.1 Trapdoor One-Way Permutations

Let D be a group over some operation $\odot$. For simplicity, we assume that choosing an element of D at random, computing $\odot$, and inverting $\odot$ each take unit time.

A trapdoor permutation family $\mathcal{TDP}$ over D is defined as a triple of algorithms, Gen, Ev, and Inv. The randomized generation algorithm Gen outputs the description s of a permutation along with the corresponding trapdoor t. The evaluation algorithm Ev, given the permutation description s and a value $x \in D$, outputs $a \in D$, the image of x under the permutation. The inversion algorithm Inv, given the permutation description s, the trapdoor t, and a value $a \in D$, outputs the preimage of a under the permutation.

We require that $\mathsf{Ev}(s, \cdot)$ be a permutation of D for all $(s,t) \overset{\text{R}}{\leftarrow} \mathsf{Gen}$, and that $x = \mathsf{Inv}(s, t, \mathsf{Ev}(s, x))$ hold for all $(s,t) \overset{\text{R}}{\leftarrow} \mathsf{Gen}$ and for all $x \in D$. The algorithms Gen, Ev, and Inv are also assumed to take unit time for simplicity.

Definition 5.2.1. The advantage of an algorithm $\mathcal{A}$ in inverting a trapdoor permutation family $\mathcal{TDP}$ is

$$\mathbf{Adv}^{\text{tdp-inv}}_{\mathcal{TDP},\mathcal{A}} \overset{\text{def}}{=} \Pr\left[x = \mathcal{A}(s, \mathsf{Ev}(s,x)) : (s,t) \overset{\text{R}}{\leftarrow} \mathsf{Gen}, x \overset{\text{R}}{\leftarrow} D\right] \quad .$$

The probability is taken over the coin tosses of Gen and of $\mathcal{A}$. An algorithm $\mathcal{A}$ (t,ϵ)-inverts a trapdoor permutation family if $\mathcal{A}$ runs in time at most t and $\mathbf{Adv}^{\text{tdp-inv}}_{\mathcal{TDP},\mathcal{A}}$ is at least ϵ. A trapdoor permutation family is (t,ϵ)-one-way if no algorithm (t,ϵ)-inverts the trapdoor permutation family.

Note that this definition of a trapdoor permutation family requires that all permutations in the trapdoor permutation family operate on the same domain D.

When it engenders no ambiguity, we consider the output of the generation algorithm Gen as a probability distribution Π on permutations, and write $(\pi, \pi^{-1}) \overset{\mathrm{R}}{\leftarrow} \Pi$; here π is the permutation $\mathsf{Ev}(s, \cdot)$, and π^{-1} is the inverse permutation $\mathsf{Inv}(s, t, \cdot)$.

5.2.2 Certified Trapdoor Permutations

The trapdoor permutation families used in sequential aggregation must be certified trapdoor permutation families [19]. A certified trapdoor permutation family is one such that, for any string s, it is easy to determine whether s can have been output by Gen, and thereby ensure that $\mathsf{Ev}(s, \cdot)$ is a permutation. This is important when permutation descriptions s can be generated by malicious parties.

Applying the definitions above to the RSA permutation family requires some care. RSA gives permutations over domains $\mathbb{Z}_N^*$, where each user has a distinct modulus N. Moreover, given just a public key (N, e), certifying that the key actually describes a permutation is nontrivial. We consider this further in Section 5.5.

5.2.3 Claw-Free Permutations, Homomorphic Trapdoor Permutations

We now describe two variants of trapdoor permutations: claw-free permutations and homomorphic trapdoor permutations. The features these variants provide are not needed in the description of the sequential aggregate signature scheme, but allow a more efficient security reduction in Theorem 5.4.3.

A *claw-free* permutation family $\mathcal{CFP}$ [66] is a trapdoor permutation family where each key (s, t) describes not only a permutation $\pi : D \to D$ as before but also an additional permutation $g : D \to D$, evaluated using the algorithm $\mathsf{EvG}(s, \cdot)$. (More generally, g can map any domain E onto D as long as the uniform distribution on E induces the uniform distribution on $g(E)$.) We assume that algorithm EvG runs in unit time and that choosing an element of E at random also takes unit time, just as above.

Definition 5.2.2. The advantage of an algorithm $\mathcal{A}$ in finding a claw in a claw-free permutation family is

$$\mathbf{Adv}^{\mathrm{claw}}_{\mathcal{CFP},\mathcal{A}} \overset{\mathrm{def}}{=} \Pr\left[\begin{array}{l} \mathsf{Ev}(s, x) = \mathsf{EvG}(s, y)\ : \\ \qquad\qquad (s, t) \overset{\mathrm{R}}{\leftarrow} \mathsf{Gen}, (x, y) \overset{\mathrm{R}}{\leftarrow} \mathcal{A}(s) \end{array}\right].$$

The probability is taken over the coin tosses of Gen and of $\mathcal{A}$. An algorithm $\mathcal{A}$ (t, ϵ)-breaks a claw-free permutation family if $\mathcal{A}$ runs in time at most t and $\mathbf{Adv}^{\mathrm{claw}}_{\mathcal{CFP},\mathcal{A}}$ is at least ϵ. A permutation family is (t, ϵ)-claw-free if no algorithm (t, ϵ)-breaks the claw-free permutation family.

Again, when it engenders no ambiguity, we abbreviate $\mathsf{EvG}(s, \cdot)$ as $g(\cdot)$, and write $(\pi, \pi^{-1}, g) \overset{\mathrm{R}}{\leftarrow} \Pi$. In this compact notation, a claw is a pair (x, y) such that $\pi(x) = g(y)$.

One can obtain from every claw-free permutation family a trapdoor permutation family, simply by ignoring EvG [66]. The proof is straightforward. Suppose there exists an algorithm $\mathcal{A}$ that inverts π with nonnegligible probability. One selects $y \overset{\mathrm{R}}{\leftarrow} D$, and provides $\mathcal{A}$ with $z = g(y)$, which is uniformly distributed in D. If $\mathcal{A}$ outputs x such that $x = \pi^{-1}(z)$, then it has uncovered a claw $\pi(x) = g(y)$.

A trapdoor permutation family is *homomorphic* if D is a group with some operation $*$ and if, for all (s, t) generated by Gen, the permutation $\pi : D \to D$ induced by $\mathsf{Ev}(s, \cdot)$ is an automorphism on D with $*$. That is, if $a = \pi(x)$ and $b = \pi(y)$, then $a * b = \pi(x * y)$. The group action $*$ is assumed to be computable in unit time. The operation $*$ can be different from the operation $\odot$ given above; nor do we require any particular relationship (e.g., distributivity) between $\odot$ and $*$.

One can obtain from every homomorphic trapdoor permutation family a claw-free permutation family [49]. Pick some $z \neq 1 \in D$, and define $g(x) = z * \pi(x)$. In this case, $E = D$. Then a claw $\pi(x) = g(y) = z * \pi(y)$ reveals $\pi^{-1}(z) = x * (1/y)$ (where the inverse is with respect to $*$).

5.2.4 Full-Domain Signatures

We review the full-domain hash signature scheme; this scheme underlies our sequential aggregate signature scheme, just as the BLS signature scheme of Section 3.3 underlies the BGLS aggregate signature scheme of Sect 4.4.2.

The full-domain hash signature scheme, introduced by Bellare and Rogaway [16], can be instantiated using any trapdoor one-way permutation family. The more efficient security reduction given by Coron [39] additionally requires that the permutation family be homomorphic. Dodis and Reyzin show that Coron's analysis can be applied for any claw-free permutation family [49]. The scheme makes use of a hash function $H : \{0,1\}^* \to D$, which is modeled as a random oracle. The signature scheme is a three-tuple of algorithms $\mathcal{FDHS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$. These behave as follows.

FDHS.Kg. For a particular user, pick random $(s,t) \xleftarrow{\mathrm{R}} \mathsf{TDP.Gen}$. The user's public key pk is s. The user's private key sk is (s,t).

FDHS.Sig(sk, M)**.** Parse the user's private key sk as (s,t). Compute $h \leftarrow H(M)$, where $h \in D$, and $\sigma \leftarrow \mathsf{TDP.Inv}(s,t,h)$. The signature is $\sigma \in D$.

FDHS.Vf(pk, M, σ)**.** Parse the user's public key pk as s. Compute $h \leftarrow H(M)$; accept if $h = \mathsf{TDP.Ev}(s, \sigma)$ holds.

The following theorem, due to Coron, shows the security of full-domain signatures under the adaptive chosen message attack in the random oracle model. The terms given in the exact analysis of ϵ and t have been adapted to agree with the accounting employed elsewhere in this thesis. A theorem with similar bounds can be given assuming claw-free permutations. (The comparable theorem for general trapdoor permutations has a security loss proportional to q_H rather than q_S.)

Theorem 5.2.3. *Let Π be a (t', ϵ')-one-way homomorphic trapdoor permutation family. Then the full-domain hash signature scheme on Π is (t, q_H, q_S, ϵ)-secure against existential forgery under an adaptive chosen-message attack (in the random oracle model) for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + 1) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - 2(q_H + 2q_S) \ .$$

Here e is the base of the natural logarithm.

5.3 Sequential Aggregate Signatures

In an aggregate signature scheme, as presented in Section 4.4, signatures are first individually generated and then combined into an aggregate. Sequential aggregate signatures are different. Each would-be signer transforms a sequential aggregate into another that includes a signature on a message of his choice. Signing and aggregation are a single operation; sequential aggregates are built in layers, like an onion; the first signature in the aggregate is the inmost. As with non-sequential aggregate signatures, the resulting sequential aggregate is the same length as an ordinary signature. This behavior closely mirrors the sequential nature of certificate chains in a PKI.

Let us restate the intuition given above more formally. A sequential aggregate signature scheme is a three-tuple of algorithms $\mathcal{SAS} = (\mathsf{Kg}, \mathsf{ASig}, \mathsf{AVf})$, which behave as follows.

SAS.Kg. As usual, a randomized algorithm that outputs a public-private keypair (pk, sk).

SAS.ASig$(sk, M, \sigma', \{pk_i, M_i\}_{i=1}^{k-1})$**.** Aggregation and signing is a combined operation. The sequential aggregate signing algorithm takes as input a private key sk, a message M in some message space to be signed, and a sequential aggregate σ' on messages $M_1, \dots, M_{k-1}$ under respective public keys $pk_1, \dots, pk_{k-1}$, where M_1 is the inmost message. If k is 1, the aggregate σ is taken to be empty. The algorithm adds an outmost signature on M under sk to the aggregate. outputting a sequential aggregate σ on all k messages under all k keys.

SAS.AVf$(\sigma, \{pk_i, M_i\}_{i=1}^{k})$**.** The input is a sequential aggregate σ on messages $M_1, \dots, M_k$ under public keys $pk_1, \dots, pk_k$. The algorithm verifies that σ is a valid sequential aggregate (with M_1 inmost) on the given messages under the given keys.

The notion of sequential aggregate signature security we consider is a generalization of existential unforgeability of ordinary signatures, which we recalled in Section 3.2, and a variant of the aggregate chosen-key model of Section 4.4.1. Specifically, the adversary attempts to forge a sequential aggregate signature, on messages of his choice, by some set of users; clearly, at least one of the users in the set must not be under the adversary's control.

We formalize this intuition as the sequential aggregate chosen-key security model. In this model, the adversary $\mathcal{A}$ is given a single public key. His goal is the existential forgery of a sequential aggregate signature. We give the adversary power to choose all public keys except the challenge public key. The adversary is also given access to a sequential aggregate signing oracle on the challenge key. His advantage, $\mathbf{Adv}^{\mathrm{seqagg}}_{\mathcal{SAS},\mathcal{A}}$, is defined to be his probability of success in the following game.

Setup. The challenger runs algorithm Kg to obtain a public key pk and private key sk. The aggregate forger $\mathcal{A}$ is given pk.

Queries. Proceeding adaptively, $\mathcal{A}$ requests sequential aggregate signatures with pk on messages of his choice. For each query, he supplies a sequential aggregate signature σ on some messages $M_1, \dots, M_{i-1}$ under distinct keys $pk_1, \dots, pk_{i-1}$, and an additional message M_i to be signed by the oracle under key pk (where i is at most n, a game parameter). The challenger responds to these queries by running ASig using private key sk. In the random oracle model, $\mathcal{A}$ can also make q_H queries to a hash oracle H.

Response. Finally, $\mathcal{A}$ outputs i distinct public keys $pk_1, \dots, pk_i$. Here i is at most n, and need not equal the lengths (also denoted i) of $\mathcal{A}$'s requests in the query phase above. One of these keys must equal pk, the challenge key. Algorithm $\mathcal{A}$ also outputs messages $M_1, \dots, M_i$, and a sequential aggregate signature σ by the i users, each on his corresponding message, with pk_1 inmost.

The forger wins if the sequential aggregate signature σ is a valid sequential aggregate signature on messages $M_1, \dots, M_i$ under keys $pk_1, \dots, pk_i$, and σ is nontrivial, i.e., $\mathcal{A}$ did not request a sequential aggregate signature on messages $M_1, \ldots, M_{i^*}$ under keys $pk_1, \ldots, pk_{i^*}$, where i^* is the index of the challenge key pk in the forgery. Note that i^* need not equal i: the forgery can be made in the middle of σ. The probability is over the coin tosses of the key-generation algorithm and of $\mathcal{A}$.

Definition 5.3.1. A sequential aggregate forger $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks an n-user aggregate signature scheme in the sequential aggregate chosen-key model if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the hash function and at most q_S queries to the aggregate signing oracle; $\mathbf{Adv}^{\mathrm{seqagg}}_{\mathcal{SAS},\mathcal{A}}$ is at least ϵ; and the forged sequential aggregate signature is by at most n users. A sequential aggregate signature scheme is $(t, q_H, q_S, n, \epsilon)$-secure against existential forgery in the sequential aggregate chosen-key model if no forger $(t, q_H, q_S, n, \epsilon)$-breaks it.

5.4 Sequential Aggregates from Trapdoor Permutations

We describe a sequential aggregate signature scheme arising from any family of trapdoor permutations, and prove the security of the scheme.

We must first introduce some notation for vectors. We write a vector as $\mathbf{x}$, its length as $|\mathbf{x}|$, and its elements as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\mathbf{x}|}$. We denote concatenating vectors as $\mathbf{x}\|\mathbf{y}$ and appending an element to a vector as $\mathbf{x}\|z$. For a vector $\mathbf{x}$, $\mathbf{x}|_a^b$ is the sub-vector containing elements $\mathbf{x}_a, \mathbf{x}_{a+1}, \ldots, \mathbf{x}_b$. (Obviously we have $1 \le a \le b \le |\mathbf{x}|$).

5.4.1 The Scheme

We now describe the three algorithms Kg, ASig, and AVf for our sequential aggregate signature scheme based on certified trapdoor permutations. The scheme employs a full-domain hash function $H : \{0,1\}^* \to D$, viewed as a random oracle, and resembles full-domain hash described in Section 5.2.4. The trick to aggregation is to incorporate the sequential aggregate signature of previous users by multiplying it (via the group operation $\odot$) together with the hash of the message. As it happens, the hash now needs to include not only the signer's message, but also her public key and the prior messages and keys.[2]

LMRS.Kg. For a particular user, pick random $(s, t) \overset{\mathrm{R}}{\leftarrow} \mathsf{TDP.Gen}$. The user's public key pk is s. The user's private key sk is (s, t).

LMRS.ASig$(sk, M, \sigma', \mathbf{M}, \mathbf{pk})$**.** The input is a private key sk, to be parsed as (s, t); a message $M \in \{0,1\}^*$ to be signed, and a sequential aggregate $\sigma' \in D$ on messages $\mathbf{M}$ under public keys $\mathbf{pk}$, to be parsed as a vector $\mathbf{s}$. Verify that σ' is a valid signature on $\mathbf{M}$ under $\mathbf{s}$ using the verification algorithm below; if not, output $\star$, indicating error. Otherwise, compute $h \leftarrow H(\mathbf{s}\|s, \mathbf{M}\|M)$, where $h \in D$, and $\sigma \leftarrow \mathsf{TDP.Inv}(s, t, h \odot \sigma')$. The sequential aggregate signature is $\sigma \in D$.

LMRS.AVf$(\sigma, \mathbf{M}, \mathbf{pk})$**.** The input is a sequential aggregate $\sigma \in D$ on messages $\mathbf{M}$ under public keys $\mathbf{pk}$, to be parsed as a vector $\mathbf{s}$. If any key appears twice in $\mathbf{s}$, if any element of $\mathbf{s}$ does not

[2]This is done not merely because we do not know how to prove the scheme secure otherwise. Micali et al. [87] pointed out that if the signature does not include the public key, then an adversary may attack the scheme by deciding on the public key after the signature is issued.

describe a valid permutation, or if $|\mathbf{M}|$ and $|\mathbf{s}|$ differ, reject. Otherwise, let i equal $|\mathbf{M}| = |\mathbf{s}|$. Set $\sigma_i \leftarrow \sigma$. Then, for $j = i, \ldots, 1$, set $\sigma_{j-1} \leftarrow \mathsf{TDP.Ev}(\mathbf{s}_j, \sigma_j) \odot H(\mathbf{s}|_1^j, \mathbf{M}|_1^j)^{-1}$. Accept if σ_0 equals 1, the unit of D with respect to $\odot$.

Written using π-notation, a sequential aggregate signature is of the form

$$\pi_i^{-1}(h_i \odot \pi_{i-1}^{-1}(h_{i-1} \odot \pi_{i-2}^{-1}(\cdots \pi_2^{-1}(h_2 \odot \pi_1^{-1}(h_1)) \cdots))) \ ,$$

where $h_j = H(\mathbf{s}|_1^j, \mathbf{M}|_1^j)$. Verification evaluates the permutations in the forward direction, peeling layers away until the center is reached.

5.4.2 Security

The following theorem demonstrates that our scheme is secure when instantiated on any certified trapdoor permutation family.

Theorem 5.4.1. *Let Π be a certified (t', ϵ')-trapdoor permutation family. Then our sequential aggregate signature scheme on Π is $(t, q_H, q_S, n, \epsilon)$-secure against existential forgery under an adaptive sequential aggregate chosen-message attack (in the random oracle model) for all t and ϵ satisfying*

$$\epsilon \geq (q_H + q_S + 1) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - (4nq_H + 4nq_S + 7n - 1) \ .$$

Following Coron's work [39], a better security reduction is obtained if the trapdoor permutations are, additionally, homomorphic under some operation $*$. (The operation $*$ need not be the same as the operation $\odot$ used in the description of the signature scheme in Section 5.4.)

Theorem 5.4.2. *Let Π be a certified homomorphic (t', ϵ')-trapdoor permutation family. Then our sequential aggregate signature scheme on Π is $(t, q_H, q_S, n, \epsilon)$-secure against existential forgery under an adaptive sequential aggregate chosen-message attack (in the random oracle model) for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + 1) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - ((4n+1)q_H + (4n+1)q_S + 7n + 3) \ .$$

Here e is the base of the natural logarithm.

Finally, following the work of Dodis and Reyzin [49], the homomorphic property is not really necessary, and can be replaced with the more general claw-free property:

Theorem 5.4.3. *Let Π be a certified (t', ϵ')-claw-free permutation family. Then the sequential aggregate signature scheme on Π is $(t, q_H, q_S, n, \epsilon)$-secure against existential forgery under an adaptive sequential aggregate chosen-message attack (in the random oracle model) for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + 1) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - (4nq_H + 4nq_S + 7n) \ .$$

Here e is the base of the natural logarithm.

The proofs of these theorems are very similar. In fact, Theorem 5.4.2 is just a corollary of Theorem 5.4.3, because, as we already saw, homomorphic trapdoor permutations are claw-free. We will prove all three at once.

Proofs. Suppose there exists a forger $\mathcal{A}$ that breaks the security of our sequential aggregate signature scheme. We describe three algorithms that use $\mathcal{A}$ to break one of the three possible security assumptions (trapdoor one-wayness, homomorphic one-wayness, and claw-freeness). In fact, the algorithms are quite similar regardless of the assumption. Therefore, we present only one of them: $\mathcal{B}$ that uses $\mathcal{A}$ to find a claw in a (supposedly) claw-free permutation family Π. We will point out later the changes needed to make the reduction to ordinary and homomorphic trapdoor permutations.

Suppose $\mathcal{A}$ is a forger algorithm that $(t, q_H, q_S, n, \epsilon)$-breaks the sequential aggregate signature scheme. We construct an algorithm $\mathcal{B}$ that finds a claw in Π.

Crucial in our construction is the following fact about our signature scheme: once the function H is fixed on i input values $(\mathbf{s}|_1^j, \mathbf{M}|_1^j)$, $1 \le j \le i$, there exists only one valid aggregate signature on $\mathbf{M}$ using keys $\mathbf{s}$. Thus, by answering hash queries properly, $\mathcal{B}$ can prepare for answering signature queries and for taking advantage of the eventual forgery.

Algorithm $\mathcal{B}$ is given the description s of an element of Π, and must find values $x \in D$ and $y \in E$ such that $\mathsf{Ev}(s, x) = \mathsf{EvG}(s, y)$. Algorithm $\mathcal{B}$ supplies $\mathcal{A}$ with the public key s. It then runs $\mathcal{A}$ and answers its oracle queries as follows.

Hash Queries. Algorithm $\mathcal{B}$ maintains a list of tuples $\langle \mathbf{s}^{(j)}, \mathbf{M}^{(j)}, w^{(j)}, r^{(j)}, c^{(j)} \rangle$, to which we refer as the H-list. The list is initially empty. When $\mathcal{A}$ queries the oracle H at a point $(\mathbf{s}, \mathbf{M})$, algorithm $\mathcal{B}$ responds as follows.

First we consider the easy cases.

- If some tuple $\langle \mathbf{s}, \mathbf{M}, w, r, c \rangle$ on the H-list already contains the query $(\mathbf{s}, \mathbf{M})$, then algorithm $\mathcal{B}$ answers the query as $H(\mathbf{s}, \mathbf{M}) = w \in D$.
- If $|\mathbf{M}|$ and $|\mathbf{s}|$ differ, if $|\mathbf{s}|$ exceeds n, if some key is repeated in $\mathbf{s}$, or if any key in $\mathbf{s}$ does not describe a valid permutation, then $(\mathbf{s}, \mathbf{M})$ can never be part of a sequential aggregate signature. Algorithm $\mathcal{B}$ picks $w \overset{\mathrm{R}}{\leftarrow} D$, and sets $r \leftarrow \star$ and $c \leftarrow \star$, both placeholder values. It adds $\langle \mathbf{s}, \mathbf{M}, w, r, c \rangle$ to the H-list and responds to the query as $H(\mathbf{s}, \mathbf{M}) = w \in D$.

Now for the more complicated cases. Set $i = |\mathbf{s}| = |\mathbf{M}|$. If i is greater than 1, $\mathcal{B}$ runs the hashing algorithm on input $(\mathbf{s}|_1^{i-1}, \mathbf{M}|_1^{i-1})$, obtaining the corresponding entry on the H-list, $\langle \mathbf{s}|_1^{i-1}, \mathbf{M}|_1^{i-1}, w', r', c' \rangle$. If i equals 1, $\mathcal{B}$ sets $r' \leftarrow 1$. Algorithm $\mathcal{B}$ must now choose elements r,w, and c to include, along with $\mathbf{s}$ and $\mathbf{M}$, in a new entry on the H-list. There are three cases to consider.

- If the challenge key s does not appear at any index of $\mathbf{s}$, $\mathcal{B}$ chooses $r \overset{\mathrm{R}}{\leftarrow} D$ at random, sets $c \leftarrow \star$, a placeholder value, and computes
$$w \leftarrow \mathsf{Ev}(\mathbf{s}_i, r) \odot (r')^{-1} \quad .$$
- If the challenge key s appears in $\mathbf{s}$ at index $i^* = i$, Algorithm $\mathcal{B}$ generates a random coin $c \in \{0, 1\}$ such that $\Pr[c = 0] = 1/(q_S + 1)$. If $c = 1$, $\mathcal{B}$ chooses $r \overset{\mathrm{R}}{\leftarrow} D$ at random and sets
$$w \leftarrow \mathsf{Ev}(s, r) \odot (r')^{-1} \quad .$$

  (In this case, w is uniform in D and independent of all other queries because r has been chosen uniformly and independently at random from D, and Ev and combining with

$(r')^{-1}$ are both permutations.) If $c = 0$, $\mathcal{B}$ chooses $r \xleftarrow{\text{R}} E$ at random and sets

$$w \leftarrow \mathsf{EvG}(s, r) \odot (r')^{-1} \ .$$

(In this case, w is uniform in D and independent of all other queries because r has been chosen uniformly and independently at random from E, EvG maps uniformly onto D, and combining with $(r')^{-1}$ is a permutation.)

- If the challenge key s appears in $\mathbf{s}$ at index $i^* \lneq i$, algorithm $\mathcal{B}$ picks $w \xleftarrow{\text{R}} D$ at random, and sets $r \leftarrow \star$ and $c \leftarrow \star$, both placeholder values.

Finally, $\mathcal{B}$ adds $\langle \mathbf{s}, \mathbf{M}, w, r, c \rangle$ to the H-list, and responds to the query as $H(\mathbf{s}, \mathbf{M}) = w$.

In all cases, $\mathcal{B}$'s response, w, is uniform in D and independent of $\mathcal{A}$'s current view, as required.

Aggregate Signature Queries. Algorithm $\mathcal{A}$ requests a sequential aggregate signature, under key s, on messages $\mathbf{M}$ under keys $\mathbf{s}$.

If $|\mathbf{s}|$ and $|\mathbf{M}|$ differ, if $|\mathbf{s}|$ exceeds n, if any key appears more than once in $\mathbf{s}$, or if any key in $\mathbf{s}$ does not describe a valid permutation, $(\mathbf{s}, \mathbf{M})$ is not a valid aggregate, and $\mathcal{B}$ responds to $\mathcal{A}$ with $\star$, indicating error. Let $i = |\mathbf{s}| = |\mathbf{M}|$. If $\mathbf{s}_i$ differs from s, $(\mathbf{s}, \mathbf{M})$ is not a valid query to the aggregate signing oracle, and $\mathcal{B}$ again responds with $\star$.

Algorithm $\mathcal{A}$ also supplies a purported sequential aggregate signature σ' on messages $\mathbf{M}|_1^{i-1}$ under keys $\mathbf{s}|_1^{i-1}$. If i equals 1, $\mathcal{B}$ verifies that σ' equals 1. Otherwise, $\mathcal{B}$ uses AVf to ensure that σ' is the correct sequential aggregate signature on $(\mathbf{s}|_1^{i-1}, \mathbf{M}|_1^{i-1})$. If σ' is incorrect, $\mathcal{B}$ again responds with $\star$.

Otherwise, $\mathcal{B}$ runs the hash algorithm on $(\mathbf{s}, \mathbf{M})$, obtaining the corresponding entry on the H-list, $\langle \mathbf{s}, \mathbf{M}, w, r, c \rangle$. Since $\mathbf{s}_i$ equals s, c must be 0 or 1. If $c = 0$ holds, $\mathcal{B}$ reports failure and terminates. Otherwise, $\mathcal{B}$ responds to the query with $\sigma \leftarrow r$.

Output. Eventually algorithm $\mathcal{A}$ halts, outputting a message vector $\mathbf{M}$, a public-key vector $\mathbf{s}$, and a corresponding sequential aggregate signature forgery σ. The forgery must be valid: No key may occur more than once in $\mathbf{s}$, each key in $\mathbf{s}$ must describe a valid permutation, the two vectors $\mathbf{s}$ and $\mathbf{M}$ must have the same length i, which is at most n. The forgery must also be nontrivial: The challenge key s must occur in $\mathbf{s}$, at some location i^*, and $\mathcal{A}$ must not have asked for a sequential aggregate signature on messages $\mathbf{M}|_1^{i^*}$ under keys $\mathbf{s}|_1^{i^*}$. If $\mathcal{A}$ fails to output a valid and nontrivial forgery, $\mathcal{B}$ reports failure and terminates.

Algorithm $\mathcal{B}$ begins by checking the hashes included in σ. For each j, $1 \le j \le i$, $\mathcal{B}$ runs its hash algorithm on $(\mathbf{s}|_1^j, \mathbf{M}|_1^j)$, obtaining a series of tuples $\langle \mathbf{s}|_1^j, \mathbf{M}|_1^j, w^{(j)}, r^{(j)}, c^{(j)} \rangle$. Note that $\mathcal{B}$ always returns w as the answer to a hash query so for each j we have $H(\mathbf{s}|_1^j, \mathbf{M}|_1^j) = w^{(j)}$.

Algorithm $\mathcal{B}$ then examines $c^{(i^*)}$. Since $s^{(i^*)}$ equals s, $c^{(i^*)}$ must be 0 or 1. If $c^{(i^*)} = 1$ holds, $\mathcal{B}$ reports failure and terminates. Then $\mathcal{B}$ applies the aggregate signature verification algorithm to σ. It sets $\sigma^{(i)} \leftarrow \sigma$. For $j = i, \ldots, 1$, it sets $\sigma^{(j-1)} \leftarrow \mathsf{Ev}(s^{(j)}, \sigma^{(j)}) \odot (w^{(j)})^{-1}$.

If $\sigma^{(0)}$ does not equal 0, σ is not a valid aggregate signature, and $\mathcal{B}$ reports failure and terminates. Otherwise, σ is valid and, moreover, each $\sigma^{(j)}$ computed by $\mathcal{B}$ is the (unique) valid aggregate signature on messages $\mathbf{M}|_1^j$ under keys $\mathbf{s}|_1^j$.

Finally, $\mathcal{B}$ sets $x \leftarrow \sigma^{(i^*)}$ and $y \leftarrow r^{(i^*)}$.

This completes the description of algorithm $\mathcal{B}$.

It is easy to modify this algorithm for homomorphic trapdoor permutations. Now the algorithm's goal is not to find a claw, but to invert the permutation given by s on a given input z. Simply replace, when answering hash queries for $c = 0$, invocation of $\mathsf{EvG}(s, r)$ with $z * \mathsf{Ev}(s, r)$. The a claw (x, y) allows $\mathcal{B}$ to recover the inverse of z under the permutation by computing $z = x * (1/y)$, where $1/y$ is the inverse of y under $*$.

Finally, it is also easy to modify this algorithm for ordinary trapdoor permutations:

- In answering hash queries where the challenge key s is outmost in $\mathbf{s}$, instead of letting $c = 0$ with probability $1/(q_S + 1)$, set $c = 0$ for exactly one query, chosen at random. There can be at most $q_H + q_S + 1$ such queries.
- For the $c = 0$ query, set $w \leftarrow z \odot (r')^{-1}$. Then w is random given $\mathcal{A}$'s view.
- If Algorithm $\mathcal{A}$'s forgery is such that $c^{(i^*)} = 0$, $\mathcal{B}''$ outputs $x \leftarrow \sigma^{(i^*)}$.

To complete the proof, we now show that $\mathcal{B}$ correctly simulates $\mathcal{A}$'s environment, and analyze its running time and success probability.

Recall that we suppose that $\mathcal{A}$ is a forger algorithm that $(t, q_H, q_S, n, \epsilon)$-breaks the sequential aggregate signature scheme. Our goal is to show that algorithm $\mathcal{B}$, described above, correctly simulates $\mathcal{A}$'s environment, runs in time t', and finds a claw in Π with probability at least ϵ', which will contradict the (t', ϵ')-claw-freeness of Π (almost identical arguments work for the modifications of $\mathcal{B}$ for ordinary and homomorphic trapdoor permutations).

We introduce some notation, which we will use to demonstrate that $\mathcal{B}$ correctly answers $\mathcal{A}$'s oracle queries. Consider public keys $\mathbf{s}$ and respective messages $\mathbf{M}$, where $i = |\mathbf{s}| = |\mathbf{M}|$, and the entries in $\mathbf{s}$ are all distinct. For each j, $1 \le j \le i$, $\mathcal{B}$'s hash algorithm associates with $(\mathbf{s}|_1^j, \mathbf{M}|_1^j)$ a tuple $\langle \mathbf{s}|_1^j, \mathbf{M}|_1^j, w^{(j)}, r^{(j)}, c^{(j)} \rangle$. The last three elements of these tuples we view as i-element vectors $\mathbf{w}$, $\mathbf{r}$, and $\mathbf{c}$. Algorithm $\mathcal{B}$ always returns w as the answer to a hash query, so, for each j, $H(\mathbf{s}|_1^j, \mathbf{M}|_1^j) = \mathbf{w}_j$. Using the compact notation suggested above, we also abbreviate the permutation evaluation $\mathsf{Ev}(\mathbf{s}_j, \cdot)$ as $\pi_j(\cdot)$. For each j there is a unique correct sequential aggregate signature σ_j on messages $\mathbf{M}|_1^j$ under keys $\mathbf{s}|_1^j$. Finally, for the challenge key s, we abbreviate the second function of the claw-free permutation pair, $\mathsf{EvG}(s, \cdot)$, as $g(\cdot)$.

Note that, for the keys $\mathbf{s}$ and messages $\mathbf{M}$ output by $\mathcal{A}$ as its forgery, $\mathcal{B}$, in its output phase, computes and makes use of the vectors $\mathbf{w}$, $\mathbf{r}$, and $\mathbf{c}$ as defined here, along with the correct sequential aggregate signatures σ_j. In our analysis, we will also consider these vectors for keys and messages other than those forged on by $\mathcal{A}$.

The proof proceeds in a series of claims. In particular, Claim 13 below shows that $\mathcal{B}$ answers $\mathcal{A}$'s signature queries with the correct sequential aggregate signature, and Claim 14 below shows that $\mathcal{B}$ outputs a claw $\pi(x) = g(y)$.

Claim 10. *If the challenge key s does not equal any of the elements of $\mathbf{s}$, then $\sigma_j = \mathbf{r}_j$ for each j, $1 \le j \le i$.*

Proof. We proceed by induction. Since $\mathbf{s}_1 \ne s$, $\mathbf{w}_1 = \pi_1(\mathbf{r}_1) \odot 1$, or, equivalently, $\mathbf{r}_1 = \pi_1^{-1}(\mathbf{w}_1) = \pi_1^{-1}(H(\mathbf{s}|_1^1, \mathbf{M}|_1^1)) = \sigma_1$. Thus the claim holds for $j = 1$. If the claim holds for $j - 1$, then, since $\mathbf{s}_j \ne s$, $\mathbf{w}_j = \pi_j(\mathbf{r}_j) \odot {\mathbf{r}_{j-1}}^{-1}$, or, equivalently, $\mathbf{r}_j = \pi_j^{-1}(\mathbf{w}_j \odot \mathbf{r}_{j-1}) = \pi_1^{-1}(H(\mathbf{s}|_1^j, \mathbf{M}|_1^j) \odot \sigma_{j-1}) = \sigma_j$, and the claim holds for j. □

Claim 11. *If the challenge key s appears at index i^* of $\mathbf{s}$, and $\mathbf{c}_{i^*} = 1$, then $\sigma_j = \mathbf{r}_j$ for each j, $1 \le j \le i^*$.*

Proof. If $\mathbf{c}_{i^*}$ equals 1, then $\mathcal{B}$ computes $\mathbf{w}_{i^*}$ precisely as it would have had $\mathbf{s}_{i^*}$ not been s. Thus the proof of Claim 10 applies still. □

Claim 12. *If the challenge key s appears at index i^* of $\mathbf{s}$, and $\mathbf{c}_{i^*} = 0$, then, for $j < i^*$, $\sigma_j = \mathbf{r}_j$, and, for $j = i^*$, $\sigma_j = \pi_{i^*}^{-1}(g(\mathbf{r}_{i^*}))$.*

Proof. For $j < i^*$, the result follows from Claim 10. We consider the case $j = i^*$. If i^* equals 1, $\mathcal{B}$ calculates the hash $\mathbf{w}_{i^*}$ as

$$\mathbf{w}_1 = g(\mathbf{r}_1) \odot 1^{-1} = g(\mathbf{r}_1) \ .$$

Thus the correct aggregate signature σ_1 is

$$\sigma_1 = \pi_1^{-1}(\mathbf{w}_1) = \pi_1^{-1}(g(\mathbf{r}_1)) \ .$$

If i^* is greater than 1, $\mathcal{B}$ calculates the hash $\mathbf{w}_{i^*}$ as

$$\mathbf{w}_{i^*} = g(\mathbf{r}_{i^*}) \odot (\mathbf{r}_{i^*-1})^{-1} \ ,$$

and thus

$$\sigma_{i^*} = \pi_{i^*}^{-1}(\mathbf{w}_{i^*} \odot \sigma_{i^*-1}) = \pi_{i^*}^{-1}(\mathbf{w}_{i^*} \odot \mathbf{r}_{i^*-1}) = \pi_{i^*}^{-1}(g(\mathbf{r}_{i^*})) \ ,$$

where the first substitution follows from the first half of this claim. Thus the claim also holds for $j = i^* > 1$. □

Using the claims above, we can demonstrate that $\mathcal{B}$ correctly answers $\mathcal{A}$'s aggregate signing queries, and that, except when it declares failure, $\mathcal{B}$ correctly computes a claw $\pi(x) = g(y)$, the solution to the challenge posed it.

Claim 13. *If $\mathcal{A}$ makes a valid sequential aggregate query, supplying messages $\mathbf{M}$, keys $\mathbf{s}$, and sequential aggregate signature σ' on all but the last message, then $\mathcal{B}$ either declares failure and halts or outputs the correct sequential aggregate signature σ on the messages.*

Proof. If the request is valid then no key appears twice in $\mathbf{s}$, $|\mathbf{s}| = |\mathbf{M}| = i \le n$, and $\mathbf{s}_i = s$. Algorithm $\mathcal{B}$ examines $\mathbf{c}_i$. If $\mathbf{c}_i$ equals 0, $\mathcal{B}$ declares failure and exits; if it equals 1, $\mathcal{B}$ outputs $\mathbf{r}_i$ as the answer to the signature query. In this case, the antecedent of Claim 11 is satisfied, and $\sigma = \sigma_i$ equals $\mathbf{r}_i$, as required. □

Claim 14. *If $\mathcal{A}$ outputs a valid and nontrivial aggregate signature forgery σ on messages $\mathbf{M}$ under keys $\mathbf{s}$ then $\mathcal{B}$ either declares failure and halts, or outputs the correct solution x to the given challenge.*

Proof. If the forgery is valid and nontrivial, then no key appears twice in $\mathbf{s}$, $|\mathbf{s}| = |\mathbf{M}| = i \le n$, and $\mathbf{s}_{i^*} = s$ for some i^*. Algorithm $\mathcal{B}$ examines $\mathbf{c}_{i^*}$. If $\mathbf{c}_{i^*}$ equals 1, $\mathcal{B}$ declares failure and exits. If $\mathbf{c}_{i^*}$ equals 0, the antecedent of Claim 12 is satisfied, and

$$\sigma_{i^*} = \pi_{i^*}^{-1}(g(\mathbf{r}_{i^*})) \ .$$

That is,

$$\pi(\sigma_{i^*}) = g(\mathbf{r}_{i^*}) \ ,$$

where we note that $\pi_{i^*}(\cdot) = \pi(\cdot)$, the challenge permutation. Algorithm $\mathcal{B}$ outputs (in our notation) $x = \sigma_{i^*}$ and $y = \mathbf{r}_{i^*}$; it therefore outputs a claw on $\pi(\cdot)$ and $g(\cdot)$, as required. □

It remains to show that $\mathcal{B}$ outputs the claw with probability at least ϵ'. To do so, we analyze the three events needed for $\mathcal{B}$ to succeed:

$\mathcal{E}_1$: $\mathcal{B}$ does not abort as a result of any of $\mathcal{A}$'s sequential aggregate signature queries.

$\mathcal{E}_2$: $\mathcal{A}$ generates a valid and nontrivial sequential aggregate forgery σ on messages $\mathbf{M}$ under keys $\mathbf{s}$.

$\mathcal{E}_3$: Event $\mathcal{E}_2$ holds, and $c = 0$ for the tuple containing $(\mathbf{s}|_1^{i^*}, \mathbf{M}|_1^{i^*})$ on the H-list, where i^* is the index of s in $\mathbf{s}$.

$\mathcal{B}$ succeeds if all of these events happen. The probability $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3]$ decomposes as

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \ . \tag{5.1}$$

The following claims give a lower bound for each of these terms.

Claim 15. *The probability that algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s aggregate signature queries is at least $1/e$. Hence, $\Pr[\mathcal{E}_1] \geq 1/e$.*

Proof. Without loss of generality we assume that $\mathcal{A}$ does not ask for the signature of the same message twice. We prove by induction that after $\mathcal{A}$ makes k signature queries the probability that $\mathcal{B}$ does not abort is at least $(1 - 1/(q_S + 1))^k$. The claim is trivially true for $k = 0$. Let $(\mathbf{s}^{(k)}, \mathbf{M}^{(k)})$ be $\mathcal{A}$'s k'th signature query and let $\langle \mathbf{s}^{(k)}, \mathbf{M}^{(k)}, w^{(k)}, r^{(k)}, c^{(k)} \rangle$ be the corresponding tuple on the H-list. Then, prior to $\mathcal{A}$'s issuing the query, the bit $c^{(k)}$ is independent of $\mathcal{A}$'s view— the only value that could be given to $\mathcal{A}$ that depends on $c^{(k)}$ is $H(\mathbf{s}^{(k)}, \mathbf{M}^{(k)})$, but the distribution of $H(\mathbf{s}^{(k)}, \mathbf{M}^{(k)})$ is the same whether $c^{(k)} = 0$ or $c^{(k)} = 1$. Therefore, the probability that this query causes $\mathcal{B}$ to abort is at most $1/(q_S + 1)$, the probability that $c^{(k)}$ equals 0. Using the inductive hypothesis and the independence of $c^{(k)}$, the probability that $\mathcal{B}$ does not abort after this query is at least $(1 - 1/(q_S + 1))^k$. This proves the inductive claim. Since $\mathcal{A}$ makes at most q_S signature queries, the probability that $\mathcal{B}$ does not abort as a result of all signature queries is at least $(1 - 1/(q_S + 1))^{q_S} \geq 1/e$. Hence $\Pr[\mathcal{E}_1] \geq 1/e$. □

Claim 16. *If algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s queries then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Hence, $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.*

Proof. The public key given to $\mathcal{A}$ is from the same distribution as public keys produced by algorithm Kg. Responses to hash queries are as in the real attack since each response is uniformly and independently distributed in D. All responses to sequential aggregate signature queries are valid. Therefore $\mathcal{A}$ will produce a valid and nontrivial aggregate signature forgery with probability at least ϵ. Hence $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$. □

Claim 17. *The probability that algorithm $\mathcal{B}$ does not abort after $\mathcal{A}$, outputs a valid and nontrivial forgery is at least $1/(q_S + 1)$. Hence, $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(q_S + 1)$.*

Proof. Given that events $\mathcal{E}_1$ and $\mathcal{E}_2$ occurred, $\mathcal{B}$ will abort only if $\mathcal{A}$ generates a forgery $(\mathbf{s}, \mathbf{M}, \sigma)$ for which the tuple $\langle \mathbf{s}|_1^{i^*}, \mathbf{M}|_1^{i^*}, w^{(i^*)}, r^{(i^*)}, c^{(i^*)} \rangle$ on the H-list has $c^{(i^*)} = 1$, where i^* is the index of s in $\mathbf{s}$. At the time $\mathcal{A}$ generates its output, it knows the value of c for those vector pairs $(\mathbf{s}', \mathbf{M}')$ on which it issued a sequential aggregate signature query (and in which s is necessarily the last key). All the remaining c's are independent of $\mathcal{A}$'s view. Indeed, if $\mathcal{A}$ did not issue a signature query for $(\mathbf{s}|_1^{i^*}, \mathbf{M}|_1^{i^*})$, then the only value given to $\mathcal{A}$ that depends on $c^{(i^*)}$ is $H(\mathbf{s}|_1^{i^*}, \mathbf{M}|_1^{i^*})$, but the distribution on $H(\mathbf{s}|_1^{i^*}, \mathbf{M}|_1^{i^*})$ is the same whether $c^{(i^*)} = 0$ or $c^{(i^*)} = 1$. Since the forgery is nontrivial, $\mathcal{A}$ could not have issued a signature query at $(\mathbf{s}|_1^{i^*}, \mathbf{M}|_1^{i^*})$, so $c^{(i^*)}$ is independent of $\mathcal{A}$'s current view and therefore $\Pr[c = 0 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(q_S + 1)$ as required. □

Using the bounds from the claims above in equation (5.1) shows that $\mathcal{B}$ produces the correct answer with probability at least $1/e \cdot \epsilon \cdot 1/(q_S + 1)$, as required.

Algorithm $\mathcal{B}$'s running time is the same as $\mathcal{A}$'s running time plus the time is takes to respond to up to q_H hash queries and q_S aggregate signature queries. Each hash query may require as many as n levels of recursion, and each level requires (at most) choosing a random value from D or E, a call to Ev or EvG, an inversion in D, and a evaluation of the group operation $\odot$ in D. Any of these operations is computable in unit time, so each hash query requires at most $4n$ time units to answer. Each signature query involves a corresponding hash computation, and so requires at most $4n$ time units to answer (σ' can be verified at no cost by comparing it to $\mathbf{r}_{i-1}$). Transforming a forgery into a claw (x, y) requires a hash query and a signature verification. As before, the hash query takes at most $4n$ time units to process. The signature verification requires at most n steps, each of which requires a call to Ev, an inversion in D, and a evaluation of the group operation $\odot$ in D, and thus takes at most n time units to process. The output step thus takes at most $7n$ time units in total. Hence $\mathcal{B}$'s total running time is at most $t + (4nq_H + 4nq_S + 7n) \leq t'$ as required.

In the case when case $\mathcal{B}$ is modified for homomorphic trapdoor permutations, the running-time accounting requires some care, since it needs now two time units to compute EvG, not one. Answering a hash oracle query $(\mathbf{s}, \mathbf{M})$ may involve up to n nested computations, but only one entry in $\mathbf{s}$ can contain the challenge key s and require a call to EvG. The same is true of the hashing required to answer signature oracle queries and in the output phase of $\mathcal{B}$. In addition, $\mathcal{B}'$ takes 2 time units to compute $\pi^{-1}(z)$. Hence the total running time of $\mathcal{B}'$ is at most $t + ((4n + 1)q_H + (4n + 1)q_S + 7n + 3) \leq t'$ as required.

Finally, when $\mathcal{B}$ is modified for plain trapdoor permutations, we analyze the running time and the success probability as follows. The challenge z is embedded in only one hash response $(\mathbf{s}, \mathbf{M})$. If $\mathcal{A}$ asks for a signature on $(\mathbf{s}, \mathbf{M})$, it cannot later forge on it — the forgery would be trivial — and so $\mathcal{B}$ can then never succeed in inverting z, and its not being able to answer $\mathcal{A}$'s query is of no consequence. Algorithm $\mathcal{B}''$ succeeds if $\mathcal{A}$ succeeds in creating a forgery, which happens with probability ϵ, and if that forgery includes the challenge $(\mathbf{s}, \mathbf{M})$, which happens with probability at least $1/(q_H + q_S + 1)$. These two probabilities are independent since the placement of the challenge is independent of $\mathcal{A}$'s view. The running time of $\mathcal{B}$ does not change. (The only difference is that, for the single hash query for which $c = 0$, $\mathcal{B}$ need not compute EvG, saving one time unit overall). □

5.5 Aggregating with RSA

Here we consider the details of instantiating the sequential aggregate signature scheme presented in Section 5.4 using the RSA permutation family.

The RSA function was introduced by Rivest, Shamir, and Adleman [105]. If $N = pq$ is the product of two large primes and $ed = 1 \bmod \phi(N)$, then $\pi(x) = x^e \bmod N$ is a permutation on $\mathbb{Z}_N^*$, and $\pi^{-1}(x) = x^d \bmod N$ is its inverse. Setting $s = (N, e)$ and $t = (d)$ gives a one-way trapdoor permutation that is multiplicatively homomorphic.

A few difficulties arise when we try to instantiate the above scheme with RSA. We tackle them individually.

The first problem is that RSA is not a *certified* trapdoor permutation. Raising to the power e may not be a permutation over $\mathbb{Z}_N^*$ if e is not relatively prime with $\phi(N)$. Moreover, even if it is a permutation of $\mathbb{Z}_N^*$, it may not be a permutation of all of $\mathbb{Z}_N$ if N is maliciously generated (in particular, if N is not square-free). Note that, for maliciously generated N, the difference between $\mathbb{Z}_N^*$ and $\mathbb{Z}_N$ may be considerable. The traditional argument used to dismiss this issue (that if one finds x outside $\mathbb{Z}_N^*$, one factors N) has no relevance here: N may be generated by the adversary, and our ability to factor it has no positive impact on the security of the scheme for the honest signer who is using a different modulus. Our security proof substantially relied on the fact that even the adversarial public keys define permutations for uniqueness of signatures and proper distribution of hash query answers. Indeed, this is not just a "proof problem," but a demonstrable security concern: If the adversary is able to precede the honest user's key (N_i, e_i) with multiple keys $(N_1, e_1), \ldots, (N_{i-1}, e_{i-1})$, each of which defines a collision-prone function rather than a permutation, then it is quite possible that no matter value one takes for σ_i, it will be likely to verify correctly: for example, there will be two valid σ_1 values, four valid σ_2 values, eight valid σ_3 values, ..., and 2^i valid σ_i values.

One way to resolve this problem is to make sure that *every* key participating in an aggregate signature has been verified to be of correct form. This could be accomplished by having a trusted certification authority check that N is a product of two large primes and e is relatively prime to $\phi(N)$ before issuing a certificate. This check, however, requires one to place more trust in the authority than usual: the authority must be trusted not just to verify the identity of a key's purported owner, but also to perform verification of some complicated properties of the key. Moreover, the security of an honest signer can be compromised *without the signer's knowledge or participation* by dishonest signers whose keys are of incorrect form, when the dishonest signers form an aggregate signature that verifies with the honest signer's public key. The only way to prevent this is to trust that the verifier of the aggregate signature only accepts certificates from certification authorities who verify the correctness of the key.

In the case when it is best to avoid assuming such complex trust relationships, we propose to tackle this problem in the same way as Micali et al. [86], though at the expense of longer verification time. First, we require e to be a prime larger than N (this idea also appeared in a paper by Cachin et al. [31]). Then e is guaranteed to be relatively prime with $\phi(N)$, and thus to provide a permutation over $\mathbb{Z}_N^*$. To extend to a permutation over $\mathbb{Z}_N$, we define $\mathsf{Ev}((N, e), x)$ as follows: if $\gcd(x, N) = 1$, output $x^e \bmod N$; else output x.

The second problem is that the natural choice for the group operation $\odot$, multiplication, is not actually a group operation over $\mathbb{Z}_N$. Thus, signature verification, which requires computation of an inverse under $\odot$, may be unable to proceed. Further, our security proof will no longer hold, since it relies on the fact that $\odot$ is a group operation for uniqueness of signatures and proper distribution of hash query answers. This difficulty is simple to overcome: Use addition modulo N as the group operation $\odot$. Recall that no properties were required of $\odot$ beyond being a group operation on the domain.

The third problem is that two users cannot share the same modulus N. Thus the domains of the one-way permutations belonging to the aggregating users differ, making it difficult to treat RSA as a family of trapdoor permutations. Below, we give three approaches that allow us to create sequential aggregates from RSA nonetheless.

Our first approach is to require the users' moduli to be arranged in increasing order: $N_1 < N_2 \cdots < N_n$. At verification, it is important to check that the i-th signature σ_i is actually less than N_i, to ensure that correct signatures are unique if H is fixed. As long as $\log N_1 - \log N_n$ is constant, and the range of H is a subset of $\mathbb{Z}_{N_1}$ whose size is a constant fraction of N_1, the scheme will be secure. The same security proof still goes through, with the following minor modification for answering hash queries. Whenever a hash query answer w is computed by first choosing a random r in $\mathbb{Z}_{N_i}$, there is a chance that w will be outside of the range of H. In this case, simply repeat with a fresh random r until w falls in the right range (the expected number of repetitions is constant). Note that because we insisted on Ev being a permutation and $\odot$ being a group operation, the resulting distribution of w is uniform on the range of H. Therefore, the distribution of answers to hash queries is uniform. Since signatures are uniquely determined by answers to hash queries, the adversary's whole view is correct, and the proof works without other modifications. (This technique is related to Coron's partial-domain hash analysis [41], though Coron deals with the more complicated case when the partial domain is exponentially smaller than the full domain.)

Our second approach allows for more general moduli: We do not require them to be in increasing order. However, we do require them to be of the same length l (constant differences in the lengths will also work, but we do not address them here for simplicity of exposition). The signature will expand by n bits $b_1 \dots b_n$, where n is the total number of users. Namely, during signing, if $\sigma_i \geq N_{i+1}$, let $b_i = 1$; else, let $b_i = 0$. During verification, if $b_i = 1$, add N_{i+1} to σ_i before proceeding with the verification of σ_i. Always check that σ_i is in the correct range $0 \leq \sigma_i < N_i$ (to ensure, again, uniqueness of signatures). The security proof requires no major modifications.[3]

A third approach is to construct from RSA a family of trapdoor permutations with a common domain. Hayashi et al. [67] describe one such construction. If all the moduli N_i are such that $2^{l-1} < N_i < 2^l$ then, under the transformation proposed by Hayashi et al., each modulus describes an "RSACD" trapdoor permutation on the range $[0, 2^l - 1]$. Using these RSACD permutations, we obtain a sequential aggregate signature scheme — based on the security of the RSA problem — in which moduli need not be placed in increasing order and in which the aggregates do not expand as more signatures are added. The downside is that each RSACD evaluation requires two evaluations of the underlying RSA permutation, so signing and verification take twice as long as in the other approaches. Note that, in the underlying RSA evaluations, one must still apply the rule laid out above in the case that $\gcd(x, N)$ is not 1.

5.5.1 Concrete Proposals for Sequential Aggregates with RSA

We now present the RSA-based aggregate signature schemes that are obtained using the suggestions above. In each case, we consider n users, all with moduli of length l bits. Let $H : \{0,1\}^* \to \{0,1\}^{l-1}$ be a hash function.

[3] We need to argue that correct signatures are unique given the hash answers. At first glance it may seem that the adversary may have choice on whether to use $b_i = 0$ or $b_i = 1$. However, this will result in two values σ_{i-1} that are guaranteed to be different: one will be less than N_i and the other at least N_i. Hence uniqueness of σ_{i-1} implies uniqueness of b_i and, therefore, σ_i. Thus, by induction, signatures are still unique. In particular, there is no need to include b_i in the hash function input.

When the moduli are ordered by size, the scheme works as follows.

Key Generation. Each user i generates an RSA public key (N_i, e_i) and secret key (N_i, d_i), ensuring that $2^{l-1}(1 + (i-1)/n) \le N_i < 2^{l-1}(1 + i/n)$ and that $e_i > N_i$ is a prime.

Signing. User i is given an aggregate signature σ', the messages $M_1, \ldots, M_{i-1}$, and the corresponding keys $(N_1, e_1), \ldots, (N_{i-1}, e_{i-1})$. User i first verifies σ', using the verification procedure below. If this succeeds, user i computes $h_i = H\big((M_1, \ldots, M_i), ((N_1, e_1), \ldots, (N_i, e_i))\big)$, $y = h_i + \sigma'$ and outputs $\sigma = y^{d_i} \bmod N_i$. The user may first check that $\gcd(y, N) = 1$ and, if not, output y; however, the chances that the check will fail are negligible, because the user is honest.

Verifying. The verifier is given as input an aggregate signature σ, the messages $M_1, \ldots, M_i$, and the corresponding keys $(N_1, e_1), \ldots, (N_i, e_i)$, and proceeds as follows. Check that no key appears twice, that $e_i > N_i$ is a prime and that N_i is of length l bits (this needs to be checked only once per key, and need not be done with every signature verification) and that $0 \le \sigma < N_i$. If $\gcd(\sigma, N_i) = 1$, let $y \leftarrow \sigma^{e_i} \bmod N_i$. Else let $y \leftarrow \sigma$ (this check is crucial, because we do not know if user i is honest). Compute $h_i \leftarrow H\big((M_1, \ldots, M_i), ((N_1, e_1), \ldots, (N_i, e_i))\big)$ and $\sigma' \leftarrow y - h_i \bmod N_i$. Verify σ' recursively. The base case for recursion is $i = 0$, in which case simply check that $\sigma = 0$.

In the second scheme, the moduli are unordered, but the aggregates are allowed to grow with additional signatures. The scheme functions like the previous one, but with the following modifications. First, the moduli N_i should now satisfy $2^{l-1} < N_i < 2^l$. Second, when signing, when verifying the aggregate-so-far σ', check if $\sigma' \ge N_i$. If so, replace σ' with $\sigma' - N_i$ and set $b_i = 1$; else, set $b_i = 0$. Finally, to verify, replace σ' with $\sigma' + b_i N_i$ before proceeding with the recursive step.

Finally, in the scheme based on RSACD, the moduli N_i must again satisfy $2^{l-1} < N_i < 2^l$. Aggregate signatures are integers in the range $[0, 2^l - 1]$. One must still check that each key describes a permutation and that the aggregate-so-far σ' is a valid aggregate signature. The RSACD trapdoor permutation can then be applied to $h \odot \sigma'$ without further preprocessing. Here $\odot$can be taken to be exclusive-or on l-bit strings.

5.5.2 Security

Because RSA over $\mathbb{Z}_N^*$ is homomorphic with respect to multiplication, it is claw-free (not just over $\mathbb{Z}_N^*$, but over entire $\mathbb{Z}_N$, because finding a claw outside of $\mathbb{Z}_N^*$ implies factoring N and hence being able to invert RSA). Therefore, the conclusions of Theorem 5.4.3 apply to the RSA-specific aggregate signature schemes described above.

Chapter 6

Group Signatures

6.1 Introduction

Group signatures, introduced by Chaum and van Heyst [36], provide anonymity for signers. Any member of the group can sign messages, but the resulting signature keeps the identity of the signer secret. Often there is a third party that can undo the signature anonymity (trace) using a special trapdoor [36, 6]. Some systems support revocation [32, 8, 116, 47], where group membership can be disabled without affecting the signing ability of unrevoked members. Currently, the most efficient constructions are based on the Strong-RSA assumption introduced by Baric and Pfitzman [10]. These signatures are usually much longer than RSA signatures of comparable security.

A number of recent projects require properties provided by group signatures. One such project is the Trusted Computing effort [115] that, among other things, enables a desktop PC to prove to a remote party what software it is running via a process called *attestation*. Group signatures are needed for privacy-preserving attestation [30] [58, Section 2.2]. To enable attestation, each computer ships with an embedded TCG tamper-resistant chip that signs certain system components using a secret key embedded in the chip. During attestation to a remote party (e.g., a bank) these signatures are sent to the remote party. To maintain user privacy it is desirable that the signatures not reveal the identity of the chip that issued them. To do so, each tamper resistant chip issues a group signature (rather than a standard signature) on system components that it signs. Here the group is the set of all TCG-enabled machines. The group signature proves that the attestation was issued by a valid tamper-resistant chip, but hides which machine it comes from.

Another is the Vehicle Safety Communications project (VSC), a collaboration of major car-makers and the U.S. Department of Transportation [34], which has attracted a great deal of other research [21]. The system embeds short-range transmitters in cars; these transmit status information to other cars in close proximity. For example, if a car executes an emergency brake, all cars in its vicinity are alerted. To prevent message spoofing, all messages are signed by a tamper-resistant chip in each car. (MACs were ruled out for this many-to-many broadcast environment.) Since VSC messages reveal the speed and location of the car, there is a strong desire to provide user privacy so that the full identity of the car sending each message is kept private. Using group signatures — where the group is the set of all cars — we can maintain privacy while still being able to revoke a signing key in case the tamper resistant chip in a car is compromised. Due to the number of cars transmitting concurrently there is a hard requirement that the length of each signature be under 250 bytes.

The two examples above illustrate the need for efficient group signatures. The second example also shows the need for short group signatures. Currently, group signatures based on Strong-RSA are too long for this application.

We construct short group signatures whose length is under 200 bytes that offer approximately the same level of security as a regular RSA signature of the same length. The security of our scheme is based on the SDH and Linear assumptions (Sects. 2.2.2 and 2.2.3). While SDH is similar to the Strong-RSA assumption, our results suggest that systems based on SDH are simpler and shorter than their Strong-RSA counterparts.

Our system is based on a new Zero-Knowledge Proof of Knowledge (ZKPK) of the solution to an SDH problem. We convert this ZKPK to a group signature via the Fiat-Shamir heuristic [51] and prove security in the random oracle model. Our security proofs use a variant of the security model for group signatures proposed by Bellare, Micciancio, and Warinschi [15].

Recently, Camenisch and Lysyanskaya [33] proposed a signature scheme with efficient protocols for obtaining and proving knowledge of signatures on committed values. They derive a group signature scheme using these protocols as building blocks. Their signature scheme is based on the LRSW assumption [81], which, like SDH, is a discrete-logarithm–type assumption. Their methodology can also be applied to the SDH assumption, yielding a different SDH-based group signature.

6.2 A Zero-Knowledge Protocol for SDH

We begin by presenting a protocol for proving possession of a solution to an SDH problem. The public values are $g_1, u, v, h \in G_1$ and $g_2, w \in G_2$. Here u, v, h are random in G_1, g_2 is a random generator of G_2, g_1 equals $\psi(g_2)$, and w equals g_2^γ for some (secret) $\gamma \in \mathbb{Z}_p$. The protocol proves possession of a pair $(A, x) \in G_1 \times \mathbb{Z}_p$ such that $A^{x+\gamma} = g_1$. Such a pair satisfies $e(A, wg_2^x) = e(g_1, g_2)$. We use a standard generalization of Schnorr's protocol for proving knowledge of discrete logarithm in a group of prime order [108].

Protocol 1. Alice, the prover, selects exponents $\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_p$, and computes a Linear encryption of A:

$$T_1 \leftarrow u^\alpha \qquad T_2 \leftarrow v^\beta \qquad T_3 \leftarrow Ah^{\alpha+\beta} \ . \tag{6.1}$$

She also computes two helper values $\delta_1 \leftarrow x\alpha$ and $\delta_2 \leftarrow x\beta \in \mathbb{Z}_p$.

Alice and Bob then undertake a proof of knowledge of values $(\alpha, \beta, x, \delta_1, \delta_2)$ satisfying the following five relations:

$$\begin{gathered} u^\alpha = T_1 \qquad\qquad v^\beta = T_2 \\ e(T_3, g_2)^x \cdot e(h, w)^{-\alpha-\beta} \cdot e(h, g_2)^{-\delta_1-\delta_2} = e(g_1, g_2)/e(T_3, w) \\ T_1^x u^{-\delta_1} = 1 \qquad\qquad T_2^x v^{-\delta_2} = 1 \ . \end{gathered}$$

This proof of knowledge of $(\alpha, \beta, x, \delta_1, \delta_2)$ proceeds as follows. Alice picks blinding values r_α, r_β, r_x, r_{δ_1}, and r_{δ_2} at random from $\mathbb{Z}_p$. She computes five values based on all these:

$$\begin{gathered} R_1 \leftarrow u^{r_\alpha} \qquad\qquad R_2 \leftarrow v^{r_\beta} \\ R_3 \leftarrow e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha-r_\beta} \cdot e(h, g_2)^{-r_{\delta_1}-r_{\delta_2}} \\ R_4 \leftarrow T_1^{r_x} \cdot u^{-r_{\delta_1}} \qquad\qquad R_5 \leftarrow T_2^{r_x} \cdot v^{-r_{\delta_2}} \ . \end{gathered} \tag{6.2}$$

She then sends $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ to the verifier. Bob, the verifier, sends a challenge value c chosen uniformly at random from $\mathbb{Z}_p$. Alice computes and sends back the values

$$s_\alpha \leftarrow r_\alpha + c\alpha \qquad s_\beta \leftarrow r_\beta + c\beta \qquad s_x \leftarrow r_x + cx \qquad s_{\delta_1} \leftarrow r_{\delta_1} + c\delta_1 \qquad s_{\delta_2} \leftarrow r_{\delta_2} + c\delta_2 \ . \tag{6.3}$$

Finally, Bob verifies the following five equations:

$$u^{s_\alpha} \stackrel{?}{=} T_1^c \cdot R_1 \tag{6.4}$$

$$v^{s_\beta} \stackrel{?}{=} T_2^c \cdot R_2 \tag{6.5}$$

$$e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \stackrel{?}{=} \big(e(g_1, g_2)/e(T_3, w)\big)^c \cdot R_3 \tag{6.6}$$

$$T_1^{s_x} \cdot u^{-s_{\delta_1}} \stackrel{?}{=} R_4 \tag{6.7}$$

$$T_2^{s_x} \cdot v^{-s_{\delta_2}} \stackrel{?}{=} R_5 \ . \tag{6.8}$$

Bob accepts if all five hold.

Theorem 6.2.1. *Protocol 1 is a public-coin honest-verifier zero-knowledge proof of knowledge of an SDH pair under the Decision Linear assumption.*

The proof of the theorem follows from the following lemmas that show that the protocol is (1) complete (the verifier always accepts an interaction with an honest prover), (2) zero-knowledge (can be simulated), and (3) a proof of knowledge (has an extractor).

Lemma 6.2.2. *Protocol 1 is complete.*

Proof. If Alice is an honest prover in possession of an SDH pair (A, x) she follows the computations specified for her in the protocol. In this case,

$$u^{s_\alpha} = u^{r_\alpha + c\alpha} = (u^\alpha)^c \cdot u^{r_\alpha} = T_1^c \cdot R_1 \ ,$$

so (6.4) holds. For analogous reasons (6.5) holds. Further,

$$T_1^{s_x} u^{-s_{\delta_1}} = (u^\alpha)^{r_x + cx} u^{-r_{\delta_1} - cx\alpha} = (u^\alpha)^{r_x} u^{-r_{\delta_1}} = T_1^{r_x} u^{-r_{\delta_1}} = R_4 \ ,$$

so (6.7) holds. For analogous reasons (6.8) holds. Finally,

$$\begin{aligned}
&e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \\
&\qquad = e(T_3, g_2)^{r_x + cx} \cdot e(h, w)^{-r_\alpha - r_\beta - c\alpha - c\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2} - cx\alpha - cx\beta} \\
&\qquad = e(T_3, g_2^x)^c \cdot e(h^{-\alpha-\beta}, w g_2^x)^c \cdot \big(e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}\big) \\
&\qquad = e(T_3 h^{-\alpha-\beta}, w g_2^x)^c \cdot e(T_3, w)^{-c} \cdot (R_3) \\
&\qquad = \big(e(A, w g_2^x)/e(T_3, w)\big)^c \cdot R_3 \\
&\qquad = \big(e(g_1, g_2)/e(T_3, w)\big)^c \cdot R_3 \ .
\end{aligned}$$

so (6.6) holds. □

Lemma 6.2.3. *For an honest verifier, transcripts of Protocol 1 can be simulated, under the Decision Linear assumption.*

Proof. We describe a simulator that outputs transcripts of Protocol 1. The simulator begins by picking $A \overset{\mathrm{R}}{\leftarrow} G_1$ and $\alpha, \beta \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$. It sets $T_1 \leftarrow u^\alpha$, $T_2 \leftarrow v^\beta$, and $T_3 \leftarrow Ah^{\alpha+\beta}$. Assuming the Decision Linear assumption holds on G_1, the tuples (T_1, T_2, T_3) generated by the simulator are drawn from a distribution that is indistinguishable from the distribution output by any particular prover.

The remainder of this simulation does not assume knowledge of A, x, α, or β, so it can also be used when T_1, T_2, and T_3 are pre-specified. When the pre-specified (T_1, T_2, T_3) are a random Linear encryption of some A, the remainder of the transcript is simulated perfectly, as in a standard simulation of a Schnorr proof of knowledge.

The simulator chooses a challenge $c \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ and values $s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$. It computes R_1, R_2, R_3, R_4, R_5 as:

$$R_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c} \qquad R_2 \leftarrow v^{s_\beta} \cdot T_2^{-c} \qquad R_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\delta_1}} \qquad R_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\delta_2}}$$
$$R_3 \leftarrow e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \big(e(T_3, w)/e(g_1, g_2)\big)^c \ .$$

It is easy to see that the resulting values R_1, R_2, R_3, R_4, R_5 satisfy Equations (6.4)–(6.8) and are distributed as in a real transcript.

The simulator outputs the transcript $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$. As discussed above, this transcript is indistinguishable from transcripts of Protocol 1, assuming the Decision Linear assumption holds. □

Lemma 6.2.4. *There exists an extractor for Protocol 1.*

Proof. Suppose that an extractor can rewind a prover in the protocol above to the point just before the prover is given a challenge c. At the first step of the protocol, the prover sends T_1, T_2, T_3 and R_1, R_2, R_3, R_4, R_5. Then, to challenge value c, the prover responds with s_α, s_β, s_x, s_{δ_1}, and s_{δ_2}. To challenge value $c' \neq c$, the prover responds with s'_α, s'_β, s'_x, s'_{δ_1}, and s'_{δ_2}. If the prover is convincing, all five verification equations (6.4)–(6.8) hold for each set of values.

For brevity, let $\Delta c = c - c'$, $\Delta s_\alpha = s_\alpha - s'_\alpha$, and similarly for Δs_β, Δs_x, Δs_{δ_1}, and Δs_{δ_2}.

Now consider (6.4) above. Dividing the two instances of this equation (one instance using c and the other using c'), we obtain $u^{\Delta s_\alpha} = T_1^{\Delta c}$. The exponents are in a group of known prime order, so we can take roots; let $\tilde{\alpha} = \Delta s_\alpha / \Delta c$. Then $u^{\tilde{\alpha}} = T_1$. Similarly, from (6.5), we obtain $\tilde{\beta} = \Delta s_\beta / \Delta c$ such that $v^{\tilde{\beta}} = T_2$.

Consider (6.7) above. Dividing the two instances gives $T_1^{\Delta s_x} = u^{\Delta s_{\delta_1}}$. Substituting $T_1 = u^{\tilde{\alpha}}$ gives $u^{\tilde{\alpha} \Delta s_x} = u^{\Delta s_{\delta_1}}$, or $\Delta s_{\delta_1} = \tilde{\alpha} \Delta s_x$. Similarly, from (6.8) we deduce that $\Delta s_{\delta_2} = \tilde{\beta} \Delta s_x$.

Finally, dividing the two instances of (6.6), we obtain

$$\begin{aligned}\big(e(g_1, g_2)/e(T_3, w)\big)^{\Delta c} &= e(T_3, g_2)^{\Delta s_x} \cdot e(h, w)^{-\Delta s_\alpha - \Delta s_\beta} \cdot e(h, g_2)^{-\Delta s_{\delta_1} - \Delta s_{\delta_2}} \\ &= e(T_3, g_2)^{\Delta s_x} \cdot e(h, w)^{-\Delta s_\alpha - \Delta s_\beta} \cdot e(h, g_2)^{-\tilde{\alpha} \Delta s_x - \tilde{\beta} \Delta s_x} \ .\end{aligned}$$

Taking Δc-th roots, and letting $\tilde{x} = \Delta s_x / \Delta c$, we obtain

$$e(g_1, g_2)/e(T_3, w) = e(T_3, g_2)^{\tilde{x}} \cdot e(h, w)^{-\tilde{\alpha} - \tilde{\beta}} \cdot e(h, g_2)^{-\tilde{x}(\tilde{\alpha} + \tilde{\beta})} \ .$$

This can be rearranged as

$$e(g_1, g_2) = e(T_3 h^{-\tilde{\alpha} - \tilde{\beta}}, w g_2^{\tilde{x}}) \ ,$$

or, letting $\tilde{A} = T_3 h^{-\tilde{\alpha}-\tilde{\beta}}$,

$$e(\tilde{A}, w g_2^{\tilde{x}}) = e(g_1, g_2) \ .$$

Thus the extractor obtains an SDH tuple $(\tilde{A}, \tilde{x})$. Moreover, the $\tilde{A}$ in this SDH tuple is, perforce, the same as that in the Linear encryption (T_1, T_2, T_3). □

6.3 Short Group Signatures from SDH

Armed with Theorem 6.2.1, we obtain from Protocol 1 a regular signature scheme secure in the random oracle model by applying the Fiat-Shamir heuristic [51, 1]. Signatures obtained from a proof of knowledge via the Fiat-Shamir heuristic are often called signatures of knowledge.

The resulting signature scheme is, in fact, also a group signature scheme, and we describe it as such. In our construction we use a variant of the Fiat-Shamir heuristic, used also by Ateniese et al. [6], where the challenge c rather than the values $R_1, \ldots, R_5$ is transmitted in the signature; the output of the random oracle acts as a checksum for those values not transmitted.

In describing the group signature, we use the terminology of Bellare et al. [15]. Consider a bilinear group pair (G_1, G_2) with a computable isomorphism ψ, as in Section 2.1. Suppose further that the SDH assumption holds on (G_1, G_2), and the Linear assumption holds on G_1. The scheme employs a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$, treated as a random oracle in the proof of security.

BBS.Kg(n)**.** This randomized algorithm takes as input a parameter n, the number of members of the group, and proceeds as follows. Select a generator g_2 in G_2 uniformly at random, and set $g_1 \leftarrow \psi(g_2)$. Select $h \overset{\text{R}}{\leftarrow} G_1 \setminus \{1_{G_1}\}$ and $\xi_1, \xi_2 \overset{\text{R}}{\leftarrow} \mathbb{Z}_p^*$, and set $u, v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$. Select $\gamma \overset{\text{R}}{\leftarrow} \mathbb{Z}_p^*$, and set $w \leftarrow g_2^{\gamma}$.

Using γ, generate for each user i, $1 \le i \le n$, an SDH tuple (A_i, x_i): select $x_i \overset{\text{R}}{\leftarrow} \mathbb{Z}_p^*$, and set $A_i \leftarrow g_1^{1/(\gamma+x_i)} \in G_1$.

The group public key is $gpk = (g_1, g_2, h, u, v, w)$. The private key of the group manager (the party able to trace signatures) is $gmsk = (\xi_1, \xi_2)$. Each user's private key is her tuple $\boldsymbol{gsk}[i] = (A_i, x_i)$. No party is allowed to possess γ; it is only known to the private-key issuer.

BBS.Sig$(gpk, \boldsymbol{gsk}[i], M)$**.** Given a group public key $gpk = (g_1, g_2, h, u, v, w)$, a user's private signing key $\boldsymbol{gsk}[i] = (A_i, x_i)$, and a message $M \in \{0,1\}^*$, compute the signature as follows:

1. Compute the values $T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5$ as specified in the first round of Protocol 1 (Equations (6.1) and (6.2)).
2. Compute a challenge c using the hash function as:

$$c \leftarrow H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5) \in \mathbb{Z}_p \ . \tag{6.9}$$

3. Using c construct the values $s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}$ as in the third round of Protocol 1 (Equation (6.3)).
4. Output the signature σ, computed as $\sigma \leftarrow (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$.

BBS.Vf(gpk, M, σ)**.** Given a group public key $gpk = (g_1, g_2, h, u, v, w)$, a message M, and a group signature σ, verify that σ is a valid signature as follows:

1. Use Equations (6.4)–(6.8) to re-derive R_1, R_2, R_3, R_4, and R_5 as follows:

$$\tilde{R_1} \leftarrow u^{s_\alpha} \cdot T_1^{-c} \qquad \tilde{R_2} \leftarrow v^{s_\beta} \cdot T_2^{-c} \qquad \tilde{R_4} \leftarrow T_1^{s_x} \cdot u^{-s_{\delta_1}} \qquad \tilde{R_5} \leftarrow T_2^{s_x} \cdot v^{-s_{\delta_2}} \tag{6.10}$$
$$\tilde{R_3} \leftarrow e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \big(e(T_3, w)/e(g_1, g_2)\big)^c \ .$$

2. Check that these, along with the other first-round values included in σ, give the challenge c, i.e., that

$$c \overset{?}{=} H(M, T_1, T_2, T_3, \tilde{R_1}, \tilde{R_2}, \tilde{R_3}, \tilde{R_4}, \tilde{R_5}) \ . \tag{6.11}$$

Accepts if this check succeeds and reject otherwise.

BBS.Open$(gpk, gmsk, M, \sigma)$**.** This algorithm is used for tracing a signature to a signer. It takes as input a group public key $gpk = (g_1, g_2, h, u, v, w)$ and the corresponding group manager's private key $gmsk = (\xi_1, \xi_2)$, together with a message M and a signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ to trace, and proceeds as follows. First, verify that σ is a valid signature on M. Second, consider the first three elements (T_1, T_2, T_3) as a Linear encryption, and recover the user's A as $A \leftarrow T_3/(T_1^{\xi_1} \cdot T_2^{\xi_2})$, following the decryption algorithm given at the end of Section 2.2.3. If the group manager is given the elements $\{A_i\}$ of the users' private keys, he can look up the user index corresponding to the identity A recovered from the signature.

Signature Length. A group signature in the system above comprises three elements of G_1 and six elements of $\mathbb{Z}_p$. Using either the supersingular or MNT family of curves described in Section 2.3.5, one can take p to be a 170-bit prime and use a group G_1 where each element is 171 bits. Thus, the total group signature length is 1533 bits or 192 bytes. With these parameters, security is approximately the same as a standard 1024-bit RSA signature, which is 128 bytes. Using the Barreto Naehrig curves of Section 2.3.5, we can instead take p to be a 160-bit prime. This gives 1443-bit group signatures with the same security level.

Performance. The pairings $e(h, w)$, $e(h, g_2)$, and $e(g_1, g_2)$ can be precomputed and their value cached by signers and by verifiers. The signer can cache $e(A, g_2)$, and, when signing, compute $e(T_3, g_2)$ without evaluating a pairing. Accordingly, creating a group signature requires eight exponentiations (or multi-exponentiations) and no pairing computations. The verifier can derive $\tilde{R_3}$ efficiently by collapsing the $e(T_3, g_2)^{s_x}$ and $e(T_3, w)^c$ pairings into a single $e(T_3, w^c g_2^{s_x})$ term. Thus verifying a group signature requires six multi-exponentiations and one pairing computation. With parameters selected as above, the exponents are in every case 170-bit numbers. For the signer, all bases for exponentiation are fixed, which allows substantial further speedups by precomputation.

6.4 BBS Group Signature Security

We now turn to proving security of the BBS scheme. Bellare, Micciancio, and Warinschi [15] give three properties that a group signature scheme must satisfy:

- correctness, which ensures that honestly-generated signatures verify and trace correctly;
- full-anonymity, which ensures that signatures do not reveal their signer's identity; and

- full-traceability, which ensures that all signatures, even those created by the collusion of multiple users and the group manager, trace to a member of the forging coalition.

For the details, see Bellare et al. [15]. A notable feature of the BMW definitions is that all keys are generated and distributed by a trusted dealer. In many earlier schemes (e.g., that of Ateniese et al. [6]), the key-generator is not fully trusted, and users engage in an interactive protocol with him to derive their keys. We consider this further in Section 7.2.

We prove the security of our scheme using a variation of these properties. Specifically, we relax the full-anonymity requirement. As presented [15, Section 2], the full-anonymity experiment allows the adversary to query the opening (tracing) oracle before and after receiving the challenge σ. In this respect, the experiment mirrors the indistinguishability experiment against an adaptive CCA2 adversary. If we term this experiment CCA2-full-anonymity, we can analogously define another experiment, CPA-full-anonymity, in which the adversary cannot query the opening oracle. We prove privacy in the weakened model that comprises the full-traceability and CPA-full-anonymity experiments.

Access to the tracing functionality will likely be carefully controlled when group signatures are deployed, so CPA-full-anonymity is a reasonable model to consider. In any case, anonymity and unlinkability, the two traditional group signature security requirements implied by full anonymity [15, Section 3], also follow from CPA-full-anonymity. Thus a fully-traceable and CPA-fully-anonymous group signature scheme is still secure in the traditional sense.

In the statements of the theorem, we use big-O notation to elide the specifics of additive terms in time bounds, noting that, for given groups G_1 and G_2, operations such as sampling, exponentiation, and bilinear map evaluation are all constant-time.[1]

Theorem 6.4.1. *The BBS group signature scheme is correct.*

Proof. For any group public key $gpk = (g_1, g_2, h, u, v, w)$, and for any user with key $\boldsymbol{gsk}[i] = (A_i, x_i)$, the key generation algorithm guarantees that $A_i^{\gamma + x_i} = g_1$, so (A_i, x_i) is an SDH tuple for $w = g_2^{\gamma}$. A correct group signature σ is a proof of knowledge, which is itself a transcript of the SDH protocol given in Section 6.2. Verifying the signature entails verifying that the transcript is correct; thus Lemma 6.2.2 shows that σ will always be accepted by the verifier.

Moreover, an honest signer outputs, as the first three components of any signature σ, values $(T_1, T_2, T_3) = (u^{\alpha}, v^{\beta}, A_i \cdot h^{\alpha+\beta})$ for some $\alpha, \beta \in \mathbb{Z}_p$. These values form a Linear encryption of A_i under public key (u, v, h), which the group manager, possessing the corresponding private key (ξ_1, ξ_2), can always recover. Therefore any valid signature will always be opened correctly. $\square$

Theorem 6.4.2. *If Linear encryption is (t', ϵ')-semantically secure on G_1 then the BBS group signature scheme is (t, q_H, ϵ)-CPA-fully-anonymous, where $\epsilon = \epsilon'$ and $t = t' - q_H\Theta(1)$. Here q_H is the number of hash function queries made by the adversary and n is the number of members of the group.*

Proof. Suppose $\mathcal{A}$ is an algorithm that (t, q_H, ϵ)-breaks the anonymity of the group signature scheme. We show how to construct a $t + q_H\Theta(1)$-time algorithm $\mathcal{B}$ that breaks the semantic security of Linear encryption from Section 2.2.3 with advantage at least ϵ.

[1]This big-O notation can be made more precise by considering a family of bilinear map groups indexed by a security parameter λ. The only complication that arises is that the basic operations are no longer all $\Theta(1)$ but, e.g., $O(\lambda^3)$ for pairing evaluation.

Algorithm $\mathcal{B}$ is given a Linear encryption public key (u, v, h). It generates the remaining components of the group signature public key by following the group signature's key generation algorithm. It then provides to $\mathcal{A}$ the group public key (g_1, g_2, h, u, v, w), and the users' private keys (A_i, x_i).

At any time, $\mathcal{A}$ can query the random oracle H. Algorithm $\mathcal{B}$ responds with elements selected uniformly at random from $\mathbb{Z}_p$, making sure to respond identically to repeated queries.

Algorithm $\mathcal{A}$ requests its full-anonymity challenge by providing two indices, i_0 and i_1, and a message M. Algorithm $\mathcal{B}$, in turn, requests its indistinguishability challenge by providing the two user private keys A_{i_0} and A_{i_1} as the messages whose Linear encryption it must distinguish. It is given a Linear encryption (T_1, T_2, T_3) of A_{i_b}, where bit b is chosen by the Linear encryption challenger.

Algorithm $\mathcal{B}$ generates from this Linear encryption a protocol transcript $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ by means of the simulator of Lemma 6.2.3. This simulator can generate a trace given (T_1, T_2, T_3), even though $\mathcal{B}$ does not know α, β, or x. Since (T_1, T_2, T_3) is a random Linear encryption of A_{i_b}, the remainder of the transcript is distributed exactly as in a real protocol with a prover whose secret A is A_{i_b}.

Algorithm $\mathcal{B}$ then patches H at $(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ to equal c. It encounters a collision only with negligible probability. In case of a collision, $\mathcal{B}$ declares failure and exits. Otherwise, it returns the valid group signature $\sigma \leftarrow (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a bit b'. Algorithm $\mathcal{B}$ returns b' as the answer to its own challenge. Since the encryption of A_{i_b} is turned by $\mathcal{B}$ into a group signature by user i_b, $\mathcal{B}$ answers its challenge correctly whenever $\mathcal{A}$ does.

The keys given to $\mathcal{A}$, and the answers to $\mathcal{A}$'s queries, are all valid and properly distributed. Therefore $\mathcal{A}$ succeeds in breaking the anonymity of the group signature σ with advantage ϵ, and $\mathcal{B}$ succeeds in distinguishing the Linear encryption (T_1, T_2, T_3) with the same advantage.

Algorithm $\mathcal{B}$'s running time exceeds $\mathcal{A}$'s by the amount it takes to answer $\mathcal{A}$'s queries. Each hash query can be answered in constant time, and there are at most q_H of them. Algorithm $\mathcal{B}$ can also create the challenge group signature σ in constant time. If $\mathcal{A}$ runs in time t, $\mathcal{B}$ runs in time $t + q_H \Theta(1)$. □

The following theorem proves full traceability of our system. The proof is based on the Forking Lemma [103].

Theorem 6.4.3. *If SDH is (q, t', ϵ')-hard on (G_1, G_2), then the BBS group signature scheme is $(t, q_H, q_S, n, \epsilon)$-fully-traceable, where $n = q - 1$, $\epsilon = 4n\sqrt{2\epsilon' q_H} + n/p$, and $t = \Theta(1) \cdot t'$. Here q_H is the number of hash function queries made by the adversary, q_S is the number of signing queries made by the adversary, and n is the number of members of the group.*

Proof. Our proof proceeds in three parts. First, we describe a framework for interacting with an algorithm that wins a full-traceability game. Second, we show how to instantiate this framework appropriately for different types of such breaker algorithms. Third, we show how to apply the Forking Lemma [103] to the framework instances, obtaining SDH solutions.

Suppose we are given an algorithm $\mathcal{A}$ that breaks the full-traceability of the group signature scheme. We describe a framework for interacting with $\mathcal{A}$.

Setup. We are given groups (G_1, G_2) as above. We are given generators g_1 and g_2 such that $g_1 = \psi(g_2)$. We are also given $w = g_2^\gamma \in G_2$, and a list of pairs (A_i, x_i) for $i = 1, \ldots, n$. For each i, either $x_i = \star$, indicating that the x_i corresponding to A_i is not known, or else (A_i, x_i)

is an SDH pair, and $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$. We pick a generator $h \overset{\text{R}}{\leftarrow} G_1 \setminus \{1_{G_1}\}$ and values $\xi_1, \xi_2 \overset{\text{R}}{\leftarrow} \mathbb{Z}_p^*$, and compute $u, v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$. We then run $\mathcal{A}$, giving it the group public key (g_1, g_2, h, u, v, w) and the group manager's private key (ξ_1, ξ_2). We answer its oracle queries as follows.

Hash Queries. When $\mathcal{A}$ asks for the hash of $(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$, we respond with a random element of G_1, memoizing the answer in case the same query is made again.

Signature Queries. Algorithm $\mathcal{A}$ asks for a signature on message M by a key at index i. If $x_i \neq \star$, we follow the group signing procedure with key (A_i, x_i) to obtain a signature σ on M, and return σ to $\mathcal{A}$. If $x_i = \star$, we pick $\alpha, \beta \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$, set $T_1 \leftarrow u^\alpha$, $T_2 = v^\beta$, and $T_3 \leftarrow Ag_1^{\alpha+\beta}$ and run the Protocol 1 simulator with values T_1, T_2, T_3. The simulator returns a transcript $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$, from which we derive a group signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$. In addition, we must patch the hash oracle at $(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ to equal c. If this causes a collision, i.e., if we previously set the oracle at this point to some other c', we declare failure and exit. Otherwise, we return σ to $\mathcal{A}$. A signature query can trigger a hash query, which we charge against $\mathcal{A}$'s hash query limit to simplify the accounting.

Private Key Queries. Algorithm $\mathcal{A}$ asks for the private key of the user at some index i. If $x_i \neq \star$, we return (A_i, x_i) to $\mathcal{A}$. Otherwise, we declare failure and exit.

Output. Finally, if algorithm $\mathcal{A}$ is successful, it outputs a forged group signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ on a message M. We use the group manager's key (ξ_1, ξ_2) to trace σ, obtaining some A^*. If $A^* \neq A_i$ for all i, we output σ. Otherwise, $A^* = A_{i^*}$ for some i^*. If $s_{i^*} = \star$, we output σ. If, however, $s_{i^*} \neq \star$, we declare failure and exit.

As implied by the output phase of the framework above, there are two types of forger algorithm. Type I forgers output a forgery σ on a message M that traces to some identity $A^* \notin \{A_1, \dots, A_n\}$. Type II forgers output a forgery that traces to an identity A^* such that $A^* = A_{i^*}$ for some i^*, and the forger did not make a private-key oracle query at i^*. We treat these two types of forger differently.

Given a q-SDH instance $(g_1', g_2', (g_2')^\gamma, (g_2')^{\gamma^2}, \dots, (g_2')^{\gamma^q})$, we apply the technique of Boneh and Boyen's Lemma 3.2 [23] that we recalled in Section 3.4.1, obtaining generators $g_1 \in G_1$, $g_2 \in G_2$, $w = g_2^\gamma$, and $q-1$ SDH pairs (A_i, x_i) such that $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$ for each i. Any SDH pair (A, x) besides these $q-1$ pairs can be transformed into a solution to the original q-SDH instance, again using Boneh and Boyen's Lemma 3.2.

Type I Forger. Against a $(t, q_H, q_S, n, \epsilon)$-Type I forger $\mathcal{A}$, we turn an instance of $(n+1)$-SDH into values (g_1, g_2, w), and n SDH pairs (A_i, x_i). We then apply the framework to $\mathcal{A}$ with these values. Algorithm $\mathcal{A}$'s environment is perfectly simulated, and the framework succeeds whenever $\mathcal{A}$ succeeds, so we obtain a Type I forgery with probability ϵ.

Type II Forger. Against a $(t, q_H, q_S, n, \epsilon)$-Type II forger $\mathcal{A}$, we turn an instance of n-SDH into values (g_1, g_2, w), and $n-1$ SDH pairs. These pairs we distribute amongst n pairs (A_i, x_i). The unfilled entry at random index i^* we fill as follows. Pick $A_{i^*} \overset{\text{R}}{\leftarrow} G_1$, and set $x_{i^*} \leftarrow \star$, a placeholder value. Now we run $\mathcal{A}$ under the framework. The framework declares success only if $\mathcal{A}$ never queries the private key oracle at i^*, but forges a group signature that traces to A_{i^*}. It is easy to see that the framework simulation is perfect unless $\mathcal{A}$ queries the private key oracle at i^*. Because the protocol simulator invoked by the signing oracle produces group signatures that are indistinguishable from those of a user whose SDH tuple includes A_{i^*}, the value of i^* is independent of $\mathcal{A}$'s view unless and until it queries the private key oracle at i^*. (Since the hash oracle takes as input nine elements of G_1 or G_2 besides the message M, the probability of collision in simulated signing queries is bounded above by $(q_H q_S + q_S^2)/p^9$. Assuming $q_S \ll q_H \ll p = |G_1|$, this probability is negligible, and we ignore it in the analysis.) Finally, when $\mathcal{A}$ outputs its forgery σ, implicating some user i whose private key $\mathcal{A}$ has not requested, the value of i^* (amongst the users whose keys it has not requested) remains independent of $\mathcal{A}$'s view. It is easy to see, then, that $\mathcal{A}$ outputs a forged group signature that traces to user i^* with probability at least ϵ/n.

Now we show how to use the application of our framework to a Type I or Type II adversary $\mathcal{A}$ to obtain another SDH pair, contradicting the SDH assumption. The remainder of this proof follows closely the methodology and notation of the Forking Lemma [103].

Let $\mathcal{A}$ be a forger (of either type) for which the framework succeeds with probability ϵ'. From here on, we abbreviate signatures as $(M, \sigma_0, c, \sigma_1)$, where $\sigma_0 = (T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$, the values given, along with M, to the random oracle H, and from which c is derived, and where $\sigma_1 = (s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$. Those values normally omitted from the signature can be recovered as in Equation (6.10).

A run of the framework on $\mathcal{A}$ is completely described by the randomness string ω used by the framework and $\mathcal{A}$, and by the vector f of responses made by the hash oracle. Let S be the set of pairs (ω, h) such that the framework, invoked on $\mathcal{A}$, completes successfully with forgery $(M, \sigma_0, c, \sigma_1)$, and $\mathcal{A}$ queried the hash oracle on (M, σ_0). In this case, let $\mathrm{Ind}(\omega, f)$ be the index of f at which $\mathcal{A}$ queried (M, σ_0). We define $\nu = \Pr[S] = \epsilon' - 1/p$, where the $1/p$ term accounts for the possibility that $\mathcal{A}$ guessed the hash of (M, σ_0) without the hash oracle's help. For each j, $1 \le j \le q_H$, let S_j be the set of pairs (ω, h) as above, and such that $\mathrm{Ind}(\omega, f) = j$. Let J be the set of auspicious indices j such that $\Pr[S_j \mid S] \ge 1/(2q_H)$. Then $\Pr[\mathrm{Ind}(\omega, f) \in J \mid S] \ge 1/2$.

Let $f|_a^b$ be the restriction of f to its elements at indices $a, a+1, \ldots, b$. For each $j \in J$, we consider the heavy-rows lemma [103, Lemma 1] with rows $X = (\omega, f|_1^{j-1})$ and columns $Y = (f|_j^{q_H})$. Clearly $\Pr_{(x,y)}[(x, y) \in S_j] \ge \nu/(2q_H)$. Let the heavy rows Ω_j be those rows such that, $\forall (x, y) \in \Omega_j : \Pr_{y'}[(x, y') \in S_j] \ge \nu/(4q_H)$. Then, by the heavy-rows lemma, $\Pr[\Omega_j \mid S_j] \ge 1/2$. A simple argument then shows that $\Pr[\exists j \in J : \Omega_j \cap S_j \mid S] \ge 1/4$.

Thus, with probability $\nu/4$, the framework, invoked on $\mathcal{A}$, succeeds and obtains a forgery $(M, \sigma_0, c, \sigma_1)$ that derives from a heavy row $(x, y) \in \Omega_j$ for some $j \in J$, i.e., an execution (ω, f) such $\Pr_{f'}\left[(\omega, f') \in S_j \;\middle|\; f'|_1^{j-1} = f|_1^{j-1}\right] \ge \nu/(4q_H)$.

If we now rewind the framework and $\mathcal{A}$ to the jth query, and proceed with an oracle vector f' that differs from f from the jth entry on, we obtain, with probability at least $\nu/(4q_H)$, a successful framework completion and a second forgery $(M, \sigma_0, c', \sigma_1')$, with (M, σ_0) still queried at $\mathcal{A}$'s jth hash query.

By using the extractor of Lemma 6.2.4, we obtain from (σ_0, c, σ_1) and $(\sigma_0, c', \sigma_1')$ an SDH tuple (A, x). The extracted A is the same as the A in the Linear encryption (T_1, T_2, T_3) in σ_0. The framework declares success only when the A encrypted in (T_1, T_2, T_3) is not amongst those whose x it knows. Therefore, the extracted SDH tuple (A, x) is not amongst those that we ourselves created, and can be transformed, again following the technique of Boneh and Boyen's Lemma 3.2 [23], to an answer to the posed q-SDH problem.

Putting everything together, we have proved the following claims.

Claim 18. *Using a* $(t, q_H, q_S, n, \epsilon)$*-Type I forger* $\mathcal{A}$*, we solve an instance of* $(n+1)$*-SDH with probability* $(\epsilon - 1/p)^2/(16q_H)$ *in time* $\Theta(1) \cdot t$*.*

Claim 19. *Using a* $(t, q_H, q_S, n, \epsilon)$*-Type II forger* $\mathcal{A}$*, we solve an instance of* n*-SDH with probability* $(\epsilon/n - 1/p)^2/(16q_H)$ *in time* $\Theta(1) \cdot t$*.*

We can guess which of the two forger types a particular forger is with probability 1/2; then assuming the more pessimistic scenario of Claim 2 proves the theorem. □

6.5 Conclusions

We presented a group signature scheme based on the Strong Diffie-Hellman (SDH) and Linear assumptions. The signature makes use of a bilinear map $e : G_1 \times G_2 \to G_T$. Instantiated over appropriate curves are used, the group G_1 has a short representation and consequently we get a group signature whose length is under 200 bytes — less than twice the length of an ordinary RSA signature (128 bytes) with comparable security. Signature generation requires no bilinear pairing computations, and verification requires a single pairing; both also require a few exponentiations with short exponents.

Chapter 7

Group Signature Variants and Extensions

7.1 Introduction

The BBS group signature scheme presented in Chapter 6 is very flexible. In this chapter, we show how to add a number of features to it.

We first consider a modification to key issuing intended to achieve a stronger exculpability guarantee. In many group signature schemes, users obtain their private keys by engaging in a JOIN protocol with a key issuer. At the conclusion of this protocol, the user obtains the private key she will use to sign, but the key issuer does not. As a result, the key issuer cannot frame the user using her key, a property known as strong exculpability. (In the Bellare-Micciancio-Warinschi model [15], the key issuer is fully trusted.)

For the remainder of the chapter, we consider on revocation mechanisms for group signatures. Group signature research has focused more on key issuing than revocation. However, in the recent applications described in Section 6.1, whereas it seems reasonable to trust the device manufacturers that issue the signing keys, revocation is critical: If, for example, the private key in a TCG chip is exposed, all signatures from that chip must be invalidated since otherwise attestation becomes meaningless.

Consider three natural communication models for revoking a user's signing capabilities, without affecting other group members:

1. The simplest method revokes user i by issuing a new signature verification key and giving each signer, except user i, a new signing key. This requires an individual secret message to each signer (e.g., TCG chip) and a public broadcast message to all verifiers.

2. A better revocation mechanism sends a single short public broadcast message to all signers and verifiers. A recent system by Camenisch and Lysyanskaya [32], based on dynamic accumulators, provides such a mechanism.

3. Brickell [30] proposes a simpler mechanism where revocation messages are only sent to signature verifiers, so that there is no need ever to communicate with an end-user machine. A similar mechanism was considered by Ateniese et al. [8] and Kiayias et al. [73]. We refer to this as Verifier-Local Revocation (VLR) group signatures.

Of these models, the first can trivially be applied to any group signature scheme.

The second model can also be applied to BBS group signatures, as we show in Section 7.3. In particular, we adapt the accumulator-based revocation mechanism of Camenisch and Lysyanskaya to the BBS scheme and its underlying SDH problem.

Finally, we construct VLR group signatures based on BBS. Signatures in our VLR system are about the same length as standard RSA signatures of comparable security. For our proof of security, we give a precise security definition, which is modeled on the Bellare-Micciancio-Warinschi framework [15].

7.2 Strong Exculpability for BBS

Exculpability, a group signature security property introduced by Ateniese and Tsudik [7], is informally defined by Bellare, Micciancio, and Warinschi [15] as follows: No member of the group and not even the group manager — the entity that is given the tracing key — can produce signatures on behalf of other users. Thus, no user can be framed for producing a signature he did not produce. Bellare et al. argue that a group signature secure in the sense of full-traceability also has the exculpability property. Thus, in the terminology of Bellare et al. [15], our BBS group signature scheme has the exculpability property.

A stronger notion of exculpability is considered in much of the group signature literature — e.g., in Ateniese et al. [6]. In this stronger version, it is required that even the entity that issues user keys cannot forge signatures under users' keys. Formalizations of strong exculpability have recently been proposed by Kiayias and Yung [74, 75] and by Bellare, Shi, and Zhang [18].

To achieve this stronger property the system of Ateniese et al. [6] uses a protocol (called JOIN) to issue a key to a new user. At the end of the protocol, the key issuer does not know the full private key given to the user and therefore cannot forge signatures under the user's key.

The BBS group signature scheme can be extended to provide strong exculpability using a similar mechanism. Instead of simply giving user i the private key $(g_1^{1/(\gamma+x_i)}, x_i)$, the user and key issuer engage in a JOIN protocol where at the end of the protocol user i has a triple (A_i, x_i, y_i) such that $A_i^{\gamma+x_i} h_1^{y_i} = g_1$ for some public parameter h_1. The value y_i is chosen by the user and is kept secret from the key issuer. The ZKPK of Section 6.2 can be modified to prove knowledge of such a triple. The resulting system is a short group signature with strong exculpability.

The JOIN protocol envisaged above would mean that the key generator no longer knows users' private keys, though it could still generate different keys and ascribe them to the users. In the Girault hierarchy [64], this is a move from level 1 to level 2. To ensure that the authority will be caught if it attempts to frame users with self-generated keys — level 3 in the hierarchy — requires that the users sign their (public) group membership certificate, and that the the tracing authority produce a proof of correct tracing.

7.3 Revocation for BBS using Accumulators

We now discuss how to revoke users in the BBS group signature scheme of Section 6.3. A number of revocation mechanisms for group signatures have been proposed [8, 32]. In this section, we describe a revocation mechanism along the lines of Camenisch and Lysyanskaya's [32], based on dynamic accumulators.

Recall that the group's public key in our scheme is (g_1, g_2, h, u, v, w) where $w = g_2^\gamma \in G_2$ for random $\gamma \in \mathbb{Z}_p^*$ and random $h, u, v \in G_1$. User i's private key is a pair (A_i, x_i) where $A_i = g_1^{1/(\gamma+x_i)} \in G_1$.

Now, suppose we wish to revoke users $1, \ldots, r$ without affecting the signing capability of other users. To do so, the revocation authority publishes a revocation list containing the private keys of all revoked users. More precisely, $\boldsymbol{rl} = \{(A_1^*, x_1), \ldots, (A_r^*, x_r)\}$, where $A_i^* = g_2^{1/(\gamma+x_i)} \in G_2$. Note that $A_i = \psi(A_i^*)$. Here the SDH secret γ is needed to compute the A_i^*'s. In the case that G_1 and G_2 are the same group we have $A_i = A_i^*$ and consequently the revocation list can be derived directly from the private keys of revoked users without having to use γ.

The list $\boldsymbol{rl}$ is given to all signers and verifiers in the system. It is used to update the group public key used to verify signatures. Let $y = \prod_{i=1}^{r}(\gamma + x_i) \in \mathbb{Z}_p^*$. The new public key is $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ where $\bar{g}_1 = g_1^{1/y}$, $\bar{g}_2 = g_2^{1/y}$, and $\bar{w} = (\bar{g}_2)^\gamma$. We show that, given $\boldsymbol{rl}$, anyone can compute this new public key, and any unrevoked user can update her private key locally so that it is well formed with respect to this new public key. Revoked users are unable to do so.

We show how to revoke one private key at a time. By repeating the process r times (as the revocation list grows over time) we can revoke all private keys on the Revocation List. We first show how given the public key (g_1, g_2, h, u, v, w) and one revoked private key $(A_1^*, x_1) \in \boldsymbol{rl}$ anyone can construct the new public key $(\hat{g}_1, \hat{g}_2, h, u, v, \hat{w})$ where $\hat{g}_1 = g_1^{1/(\gamma+x_1)}$, $\hat{g}_2 = g_2^{1/(\gamma+x_1)}$, and $\hat{w} = (\hat{g}_2)^\gamma$. This new public key is constructed simply as:

$$\hat{g}_1 \leftarrow \psi(A_1^*) \qquad \hat{g}_2 \leftarrow A_1^* \qquad \text{and} \qquad \hat{w} \leftarrow g_2 \cdot (A_1^*)^{-x_1} \enspace ;$$

then $\hat{g}_1 = \psi(A_1)^* = g_1^{1/(\gamma+x_1)}$ and $\hat{w} = g_2 \cdot (A_1^*)^{-x_1} = g_2^{1-\frac{x_1}{\gamma+x_1}} = (A_1^*)^\gamma = (\hat{g}_2)^\gamma$, as required.

Next, we show how unrevoked users update their own private keys. Consider an unrevoked user whose private key is (A, x). Given a revoked private key, (A_1^*, x_1) the user computes $\hat{A} \leftarrow \psi(A_1^*)^{1/(x-x_1)}/A^{1/(x-x_1)}$ and sets his new private key to be $(\hat{A}, x)$. Then, indeed,

$$(\hat{A})^{\gamma+x} = \psi(A_1^*)^{\frac{\gamma+x}{x-x_1}}/A^{\frac{\gamma+x}{x-x_1}} = \psi(A_1^*)^{\frac{(\gamma+x_1)+(x-x_1)}{x-x_1}}/g_1^{\frac{1}{x-x_1}} = \psi(A_1^*) = \hat{g}_1 \enspace ,$$

as required. Hence, $(\hat{A}, x)$ is a valid private key with respect to $(\hat{g}_1, \hat{g}_2, h, u, v, \hat{w})$.

By repeating this process r times (once for each revoked key in $\boldsymbol{rl}$) anyone can compute the updated public key $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ defined above. Similarly, an unrevoked user with private key (A, x) can compute his updated private key $(\bar{A}, x)$ where $\bar{A} = (\bar{g}_1)^{1/(\gamma+x)}$. We note that it is possible to process the entire $\boldsymbol{rl}$ at once (as opposed to one element at a time) and compute $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ directly; however this is less efficient when keys are added to $\boldsymbol{rl}$ incrementally.

A revoked user cannot construct a private key for the new public key $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$. In fact, the proof of Theorem 6.4.3 shows that, if a revoked user can generate signatures for the new public key $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$, then that user can be used to break the SDH assumption. Very briefly, the reason is that given an SDH challenge one can easily generate a public key tuple $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ along with the private key for a revoked user $(g_1^{1/(x+\gamma)}, x)$. Then an algorithm that can forge signatures given these two tuples can be used to solve the SDH challenge.

In the revocation mechanism above a user is revoked by the publication of a value that exposes that user's private key. Consequently, it is crucial that updates to the revocation list be sent simultaneously to all verifiers. Otherwise, someone who obtains a new entry on the revocation list can fool a verifier who has not yet updated his copy of the revocation list. The verifier-local revocation method described in the next section addresses this limitation.

7.4 Verifier-Local Revocation

In a group signature with verifier-local revocation, signers are stateless, and revocation messages are processed by the verifiers alone [30, 8, 73]. Distributing revocation information only to the signers simplifies revocation when verifiers are fewer than signers and, when signing functionality is implemented in a tamper-resistant module, allowing signers to be stateless gives added robustness and security. Therefore, verifier-local revocation is advantageous for privacy-preserving attestation in the trusted computing environment.

We implement verifier-Local group signatures by providing to the signature verification algorithm the Revocation List ($\boldsymbol{rl}$) as additional argument. The $\boldsymbol{rl}$ contains a token for each revoked user. The verification algorithm accepts all signatures issued by unrevoked users and reveals no information about which unrevoked user issued the signature. However, if a user is ever revoked (by having his revocation token added to the $\boldsymbol{rl}$), signatures from that user are no longer accepted. It follows that signatures from a revoked user become linkable: To test that two signatures were issued by the same revoked user, verify the signatures once using the $\boldsymbol{rl}$ before the user is revoked and once using the $\boldsymbol{rl}$ after. In the case of trusted computing, for example, users who break the tamper resistance of their TCG chip and are revoked would lose their privacy by design.

Our specific VLR group signatures have an additional useful property: given a user's private key it is easy to derive that user's revocation token — the revocation token is the left half of the private key. Hence, any private key that is published on the web can be trivially added to the $\boldsymbol{rl}$ and revoked. This potentially eliminates the need for a trusted revocation authority. Instead, revocation could be done by just scanning the web and newsgroups for exposed private keys and telling all signature verifiers to add these keys to their $\boldsymbol{rl}$. We discuss this in more detail in the next section.

7.4.1 Definitions

Formally, a VLR group signature scheme comprises three algorithms, Kg, Sig, and Vf, which behave as follows:

VLR.Kg(n)**.** This randomized algorithm takes as input a parameter n, the number of members of the group. It outputs a group public key gpk, an n-element vector of user keys $\boldsymbol{gsk} = (\boldsymbol{gsk}[1], \boldsymbol{gsk}[2], \ldots, \boldsymbol{gsk}[n])$, and an n-element vector of user revocation tokens $\boldsymbol{grt}$, similarly indexed.

VLR.Sig$(gpk, \boldsymbol{gsk}[i], M)$**.** The (randomized) signing algorithm takes as input the group public key gpk, a private key $\boldsymbol{gsk}[i]$, and a message $M \in \{0,1\}^*$, and returns a signature σ.

VLR.Vf$(gpk, \boldsymbol{rl}, \sigma, M)$**.** The verification algorithm takes as input the group public key gpk, a set of revocation tokens $\boldsymbol{rl}$ (whose elements form a subset of the elements of $\boldsymbol{grt}$), and a purported signature σ on a message M. It returns either `valid` or `invalid`. The latter response can mean either that σ is not a valid signature, or that the user who generated it has been revoked.

Implicit Tracing Algorithm Any VLR group signature scheme has an associated implicit tracing algorithm that, using a secret tracing key, can trace a signature to at least one group member who generated it. The vector of revocation tokens, $\boldsymbol{grt}$, functions as this secret tracing key. Given a valid message-signature pair (M, σ), a party possessing all the revocation tokens $\boldsymbol{grt}$ can determine which user issued the signature using the following algorithm:

1. For each $i = 1, \ldots, n$ run the verification algorithm on M, σ with revocation list $\boldsymbol{rl} = \{\boldsymbol{grt}[i]\}$.
2. Output the index of the first user for which the verification algorithm says `invalid`. Output `fail` if the signature verifies properly for all n users.

Our security definitions below explain why this is a correct tracing algorithm. The algorithm above demonstrates that the $\boldsymbol{grt}$ vector can function as a secret tracing key, if so desired. Note that $\boldsymbol{grt}$ in the BS scheme can be derived from just one value so that there is no need to store a large vector as a tracing key.

In the constructions we have in mind, a user can derive her revocation token from her private key, and can therefore determine whether her key was used to generate a particular signature. We refer to this as *selfless-anonymity*: a group member can tell whether she generated a particular signature σ, but if she didn't she learns nothing else about the origin of σ. We describe a new security model that captures this notion. We use the framework of Bellare et al. [15].

A secure VLR group signature scheme must satisfy three requirements: correctness, traceability, and selfless-anonymity. We describe each in turn.

Correctness This requires that, for all $(gpk, \boldsymbol{gsk}, \boldsymbol{grt})$ generated by the generation algorithm, every signature generated by a user verify as valid, except when the user is revoked; or, formally, that

$$\mathsf{Vf}(gpk, \boldsymbol{rl}, \mathsf{Sig}(gpk, \boldsymbol{gsk}[i], M), M) = \texttt{valid} \quad \Longleftrightarrow \quad \boldsymbol{grt}[i] \notin \boldsymbol{rl} \ .$$

Traceability We say that a VLR group signature scheme is traceable if no adversary can win the traceability game. In the traceability game, the adversary's goal is to forge a signature that cannot be traced to one of the users in his coalition using the implicit tracing algorithm above. Let n be a given group size. The traceability game, between a challenger and an adversary $\mathcal{A}$, is defined as follows.

Setup. The challenger runs algorithm $\mathsf{Kg}(n)$, obtaining group parameters gpk, $\boldsymbol{gsk}$, and $\boldsymbol{grt}$. He provides the adversary $\mathcal{A}$ with gpk and $\boldsymbol{grt}$, and sets $U \leftarrow \emptyset$.

Queries Algorithm $\mathcal{A}$ can make queries of the challenger, as follows.

Signing. Algorithm $\mathcal{A}$ requests a signature on an arbitrary message M for the user at index i, where $1 \leq i \leq n$. The challenger computes $\sigma \leftarrow \mathsf{Sig}(gpk, \boldsymbol{gsk}[i], M)$ and returns the signature σ to $\mathcal{A}$.

Corruption. Algorithm $\mathcal{A}$ requests the private key of the user at index i, $1 \leq i \leq n$. The challenger appends i to U, the adversary's coalition, and responds with $\boldsymbol{gsk}[i]$.

Response. Finally, forger $\mathcal{A}$ outputs a message M^*, a set $\boldsymbol{rl}^*$ of revocation tokens, and a signature σ^*.

The forger wins if: (1) σ^* is accepted by the verification algorithm as a valid signature on M^* with revocation-token set $\boldsymbol{rl}^*$; (2) σ^* traces (using the implicit tracing algorithm above) to some user outside of the coalition $U \backslash \boldsymbol{rl}^*$, or the tracing algorithm fails; and (3) σ^* is nontrivial, i.e., $\mathcal{A}$ did not obtain σ^* by making a signing query at M^*.

We denote by $\mathbf{Adv}^{\text{vlr-trace}}_{\mathcal{VLR},\mathcal{A}}$ the probability that $\mathcal{A}$ wins the game. The probability is taken over the coin tosses of $\mathcal{A}$ and the randomized key generation and signing algorithms.

The security proof for our system is set in the random oracle model [16] and therefore we include in our security definitions an extra parameter q_H denoting the number of random oracle queries that the adversary issues.

Definition 7.4.1. An aggregate forger $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks traceability in an n-user VLR group signature scheme if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H hash oracle queries and at most q_S signing queries; and $\mathbf{Adv}^{\text{vlr-trace}}_{\mathcal{VLR},\mathcal{A}}$ is at least ϵ.

Selfless-anonymity In the selfless-anonymity game, the adversary's goal is to determine which of two keys generated a signature. He is not given access to either key. The game is defined as follows.

Setup. The challenger runs the Kg algorithm, obtaining group parameters gpk, $\boldsymbol{gsk}$, and $\boldsymbol{grt}$. It provides the adversary $\mathcal{A}$ with gpk.

Queries. Algorithm $\mathcal{A}$ can make queries of the challenger, as follows.

- **Signing.** Algorithm $\mathcal{A}$ requests a signature on an arbitrary message M for the user at index i, where $1 \le i \le n$. The challenger computes $\sigma \leftarrow \mathsf{Sig}(gpk, \boldsymbol{gsk}[i], M)$ and returns the signature σ to $\mathcal{A}$.
- **Corruption.** Algorithm $\mathcal{A}$ request the private key of the user at index i, $1 \le i \le n$. The challenger responds with $\boldsymbol{gsk}[i]$.
- **Revocation.** Algorithm $\mathcal{A}$ can request the revocation token of the user at index i, $1 \le i \le n$. The challenger responds with $\boldsymbol{grt}[i]$.

Challenge. Algorithm $\mathcal{A}$ outputs a message M and two indices i_0 and i_1. It must have made neither a corruption nor a revocation query at either index. The challenger chooses a bit $b \xleftarrow{\text{R}} \{0,1\}$ uniformly at random, computes a signature on M by user i_b as $\sigma^* \leftarrow \mathsf{Sig}(gpk, \boldsymbol{gsk}[i_b], M)$, and provides σ^* to $\mathcal{A}$.

Restricted Queries. After obtaining the challenge, algorithm $\mathcal{A}$ is allowed to make additional queries of the challenger, restricted as follows.

- **Signing.** Algorithm $\mathcal{A}$ can make signing queries as before.
- **Corruption.** As before, but $\mathcal{A}$ cannot make corruption queries at i_0 and i_1.
- **Revocation.** As before, but $\mathcal{A}$ cannot make revocation queries at i_0 and i_1.

Output. Finally, $\mathcal{A}$ outputs a bit b', its guess of b. The adversary wins if $b' = b$.

We define $\mathcal{A}$'s advantage in winning the game as $\mathbf{Adv}^{\text{vlr-sanon}}_{\mathcal{VLR},\mathcal{A}}$ as $|\Pr[b = b'] - 1/2|$. The probability is taken over the coin tosses of $\mathcal{A}$, of the randomized key generation and signing algorithms, and the choice of b. Note that $\mathcal{A}$ can make no more than $n-2$ corruption and revocation queries.

Definition 7.4.2. An aggregate adversary $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks selfless-anonymity in an n-user VLR group signature scheme if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the hash function and at most q_S signing queries; and $\mathbf{Adv}^{\text{vlr-sanon}}_{\mathcal{VLR},\mathcal{A}}$ is at least ϵ.

Definition 7.4.3. A group signature scheme with verifier-local revocation is $(t, q_H, q_S, n, \epsilon)$ secure in the VLR security model if: it is correct; no algorithm $(t, q_H, q_S, n, \epsilon)$-breaks its traceability; and no algorithm $(t, q_H, q_S, n, \epsilon)$-breaks its selfless-anonymity.

We note that a signature scheme that satisfies the VLR security model above is existentially unforgeable under a chosen message attack. This follows immediately from the traceability game.

7.4.2 Short VLR Group Signatures from SDH

In this subsection, we describe in detail the BS VLR group signature scheme. (In the next subsection, we give intuition for how the scheme is derived.) As with the BBS group signature scheme of Section 6.3, we base security on the SDH and Linear assumptions.

Consider bilinear groups (G_1, G_2) with isomorphism ψ and respective generators g_1 and g_2, as in Section 2.1. The scheme employs hash functions H_0 and H, with respective ranges G_2^2 and $\mathbb{Z}_p$, treated as random oracles.

BS.Kg(n)**.** The key generation algorithm takes as input n, the number of user keys to generate. It proceeds as follows:

1. Select a generator g_2 in G_2 uniformly at random, and set $g_1 \leftarrow \psi(g_2)$. (In the unlikely case that $e(\psi(g_2), g_2) = 1$, repeat this step; see Section 2.1.2.)
2. Select $\gamma \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p^*$ and set $w = g_2^{\gamma}$.
3. Using γ, generate for each user an SDH tuple (A_i, x_i) by selecting $x_i \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p^*$ such that $\gamma + x_i \neq 0$, and setting $A_i \leftarrow g_1^{1/(\gamma + x_i)}$.

The group public key is $gpk = (g_1, g_2, w)$. Each user's private key is her tuple $\boldsymbol{gsk}[i] = (A_i, x_i)$. The revocation token corresponding to a user's key (A_i, x_i) is $\boldsymbol{grt}[i] = A_i$. The algorithm outputs $(gpk, \boldsymbol{gsk}, \boldsymbol{grt})$. No party is allowed to possess γ; it is only known to the private-key issuer.

BS.Sig$(gpk, \boldsymbol{gsk}[i], M)$**.** The signing algorithm takes as input a group public key $gpk = (g_1, g_2, w)$, a user private key $\boldsymbol{gsk}[i] = (A_i, x_i)$, and a message $M \in \{0,1\}^*$, and proceeds as follows.

1. Pick a random nonce $r \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$. Obtain generators $(\hat{u}, \hat{v})$ in G_2 from H_0 as
$$(\hat{u}, \hat{v}) \leftarrow H_0(gpk, M, r) \in G_2^2 \ , \tag{7.1}$$
and compute their images in G_1:
$$u \leftarrow \psi(\hat{u}) \ , \qquad v \leftarrow \psi(\hat{v}) \ .$$
2. Select an exponent $\alpha \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ and compute:
$$T_1 \leftarrow u^{\alpha} \qquad \text{and} \qquad T_2 \leftarrow A_i v^{\alpha} \ . \tag{7.2}$$
3. Set $\delta \leftarrow x_i \alpha \in \mathbb{Z}_p$. Pick blinding values r_α, r_x, and $r_\delta \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$.
4. Compute helper values R_1, R_2, and R_3:
$$\begin{aligned} R_1 \leftarrow u^{r_\alpha} \qquad\qquad & R_3 \leftarrow T_1^{r_x} \cdot u^{-r_\delta} \\ R_2 \leftarrow e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} & \cdot e(v, g_2)^{-r_\delta} \ . \end{aligned} \tag{7.3}$$

5. Compute a challenge value $c \in \mathbb{Z}_p$ using H:

$$c \leftarrow H(gpk, M, r, T_1, T_2, R_1, R_2, R_3) \in \mathbb{Z}_p \ . \tag{7.4}$$

6. Compute $s_\alpha = r_\alpha + c\alpha$, $s_x = r_x + cx_i$, and $s_\delta = r_\delta + c\delta \in \mathbb{Z}_p$.

Output the signature $\sigma \leftarrow (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$.

BS.Vf$(gpk, \boldsymbol{rl}, \sigma, M)$**.** The verification algorithm takes as input a group public key $gpk = (g_1, g_2, w)$, a set $\boldsymbol{rl}$ of revocation tokens (each an element of G_1), a purported signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$, and a message $M \in \{0,1\}^*$, and proceeds in two phases. First, it ensures that the signature σ is valid; then it ensures that σ was not generated by a revoked user. It accepts only if both conditions hold.

1. Signature Check. Check that σ is a valid signature, as follows.

1. Compute $\hat{u}$ and $\hat{v}$ using equation (7.1), and their images u and v in G_1:

$$u \leftarrow \psi(\hat{u}) \ , \qquad v \leftarrow \psi(\hat{v}) \ .$$

2. Re-derive R_1, R_2, and R_3 as:

$$\begin{gathered} \tilde{R_1} \leftarrow u^{s_\alpha}/T_1^c \qquad \tilde{R_3} \leftarrow T_1^{s_x} u^{-s_\delta} \\ \tilde{R_2} \leftarrow e(T_2, g_2)^{s_x} e(v, w)^{-s_\alpha} e(v, g_2)^{-s_\delta} \cdot \big(e(T_2, w)/e(g_1, g_2)\big)^c \ . \end{gathered} \tag{7.5}$$

3. Check that the challenge c is correct:

$$c \stackrel{?}{=} H(gpk, M, r, T_1, T_2, \tilde{R_1}, \tilde{R_2}, \tilde{R_3}) \ . \tag{7.6}$$

If it is, accept. Otherwise, reject.

2. Revocation Check. For each element $A \in \boldsymbol{rl}$, check whether A is encoded in (T_1, T_2) by checking if

$$e(T_2/A, \hat{u}) \stackrel{?}{=} e(T_1, \hat{v}) \ .$$

If no element of $\boldsymbol{rl}$ is encoded in (T_1, T_2), the signer of σ has not been revoked.

The algorithm outputs `valid` if both phases accept, `invalid` otherwise.

Signature Length A group signature in the system above comprises two elements of G_1 and five elements of $\mathbb{Z}_p$. Using either the supersingular or MNT family of curves described in Section 2.3.5, one can take p to be a 170-bit prime and use a group G_1 where each element is 171 bits. Thus, the total group signature length is 1192 bits or 149 bytes. With these parameters, security is approximately the same as a standard 1024-bit RSA signature, which is 128 bytes. Using the Barreto Naehrig curves of Section 2.3.5, we can instead take p to be a 160-bit prime. This gives 1122-bit group signatures with the same security level.

Performance Signature generation requires two applications of the isomorphism ψ. Computing the isomorphism takes roughly the same time as an exponentiation in G_1 (using fast computations of the trace map). Thus, signature generation requires about 8 exponentiations (or multi-exponentiations) and 2 bilinear map computations. Signature verification takes 6 exponentiations and $3 + 2|\boldsymbol{rl}|$ computations of the bilinear map. A far more efficient revocation check algorithm, whose running time is independent of $|\boldsymbol{rl}|$, is described in Section 7.4.5.

We now prove the correctness of the BS scheme. The proofs of the selfless-anonymity and traceability are given in Section 7.4.4.

Theorem 7.4.4. *The BS VLR signature scheme is correct, as defined in equation (7.4.1).*

Proof. Consider public parameters $gpk = (g_1, g_2, w)$; secret-key vector $\boldsymbol{gsk}$ where, for each i, $\boldsymbol{gsk}[i] = (A_i, x_i)$, an SDH tuple, i.e., a tuple satisfying $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$; and revocation-token list $\boldsymbol{grt}$ where $\boldsymbol{grt}[i] = A_i$, as output by the key generation algorithm.

An honest signer with private key (A_i, x_i) generates a signature $(r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$ by following the signing algorithm described above. In particular, the signer computes the generators $\hat{u}$ and $\hat{v}$ according to equation (7.1), so the verifier uses the same generators. Now, the first phase of the signature verification algorithm accepts a signature if the output of H equals the challenge c. This will only be true (except with negligible probability) when all inputs to H are exactly the same for the verifier as for the signer. An honest signer's signature includes all these inputs except R_1, R_2, and R_3, which are re-derived by the verifier. We must therefore show that the values re-derived by the verifier using equations (7.5) equal those derived by the signer using equations (7.3). First,

$$\tilde{R}_1 = u^{s_\alpha}/T_1^c = u^{r_\alpha + c\alpha}/(u^\alpha)^c = u^{r_\alpha} = R_1 \ ,$$

so $\tilde{R}_1 = R_1$. Further,

$$\tilde{R}_3 = T_1^{s_x} u^{-s_\delta} = (u^\alpha)^{r_x + cx_i} \cdot u^{-r_\delta - cx_i\alpha} = (u^\alpha)^{r_x} \cdot u^{-r_\delta} = T_1^{r_x} \cdot u^{-r_\delta} = R_3 \ ,$$

so $\tilde{R}_3 = R_3$. Finally,

$$\begin{aligned}\tilde{R}_2 &= e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \cdot \left(\frac{e(T_2, w)}{e(g_1, g_2)}\right)^c \\ &= \Big(e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} \cdot e(v, g_2)^{-r_\delta}\Big) \\ &\qquad \times \Big(e(T_2, g_2)^{x_i} \cdot e(v, w)^{-\alpha} \cdot e(v, g_2)^{-x_i\alpha} \cdot \frac{e(T_2, w)}{e(g_1, g_2)}\Big)^c \\ &= R_2 \cdot \Big(\frac{e(T_2 v^{-\alpha}, wg_2^{x_i})}{e(g_1, g_2)}\Big)^c = R_2 \cdot \Big(\frac{e(A_i, wg_2^{x_i})}{e(g_1, g_2)}\Big)^c = R_2 \ ,\end{aligned}$$

so $\tilde{R}_2 = R_2$. The last equality follows from the SDH equation. Thus (7.6) will be satisfied.

In a signature generated by the signing algorithm, we have $T_1 = \psi(\hat{u})^\alpha$ and $T_2 = A_i\psi(\hat{v})^\alpha$ for some α. The revocation check algorithm will reject a signature as originating from a revoked user with token A exactly when $(\hat{u}, \hat{v}, T_1, T_2/A)$ is a co-Diffie-Hellman tuple, i.e., when A equals A_i. Thus the group signature verification algorithm will accept a signature as valid exactly when A_i is not included in its input $\boldsymbol{rl}$, as required. □

7.4.3 Intuition

The BS scheme presented in Section 7.4.2 above is derived, via a variant of the Fiat-Shamir heuristic [51], from a new protocol for proving possession of an SDH tuple. We present this protocol below to give intuition into the construction of the BS scheme.

The protocol is a proof of knowledge, which means that by rewinding a prover it is possible to extract an SDH pair. The protocol is intentionally *not* zero-knowledge; a verifier in possession of a revocation token can determine whether he is interacting with a revoked prover.

The public values are $g_1 \in G_1$ and $g_2, w \in G_2$. Here g_2 is a random generator of G_2, g_1 equals $\psi(g_2)$, and w equals g_2^γ for some (secret) $\gamma \in \mathbb{Z}_p$. The prover wishes to demonstrate possession of a pair (A, x), where $A \in G_1$ and $x \in \mathbb{Z}_p$, such that $A^{x+\gamma} = g_1$. Such a pair satisfies $e(A, wg_2^x) = e(g_1, g_2)$. We use a generalization of Schnorr's protocol for proving knowledge of discrete logarithm [108] in a group of prime order.

Protocol 2. Bob, the verifier, selects elements $\hat{u}$ and $\hat{v}$ uniformly at random from G_2 and sends them to Alice, the prover. Alice sets $u \leftarrow \psi(\hat{u})$ and $v \leftarrow \psi(\hat{v})$. She selects exponent $\alpha \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, and computes

$$T_1 \leftarrow u^\alpha \qquad \text{and} \qquad T_2 \leftarrow Av^\alpha \ .$$

Alice and Bob then undertake a proof of knowledge of values (α, x, δ) satisfying the following three relations:

$$u^\alpha = T_1 \ , \qquad T_1^x = u^\delta \ , \qquad e(T_2 v^{-\alpha}, wg_2^x) = e(g_1, g_2) \ .$$

This proof of knowledge proceeds as follows. Alice computes a helper value $\delta = x\alpha$. She then picks blinding values r_α, r_x, and r_δ at random from $\mathbb{Z}_p$. She computes three values based on all these:

$$R_1 \leftarrow u^{r_\alpha} \qquad\qquad R_3 \leftarrow T_1^{r_x} \cdot u^{-r_\delta}$$
$$R_2 \leftarrow e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} \cdot e(v, g_2)^{-r_\delta} \ .$$

She then sends $(T_1, T_2, R_1, R_2, R_3)$ to Bob. Bob sends a challenge value c chosen uniformly at random from $\mathbb{Z}_p$. Alice computes and sends back $s_\alpha = r_\alpha + c\alpha$, $s_x = r_x + cx$, and $s_\delta = r_\delta + c\delta$. Finally, Bob verifies the following three equations:

$$u^{s_\alpha} \overset{?}{=} T_1^c \cdot R_1 \tag{7.7}$$
$$e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \overset{?}{=} \big(e(g_1, g_2)/e(T_2, w)\big)^c \cdot R_2 \tag{7.8}$$
$$T_1^{s_x} u^{-s_\delta} \overset{?}{=} R_3 \ . \tag{7.9}$$

Bob accepts if all three hold. Applying a standard variant of the Fiat-Shamir heuristic to this protocol produces the signature scheme of the previous section.

The protocol above is (by design) not a zero-knowledge protocol. Given (T_1, T_2) and a candidate A, anyone can check whether A is ElGamal-encrypted in (T_1, T_2) by checking whether $e(T_2/A, \hat{u}) \overset{?}{=} e(T_1, \hat{v})$ holds. Below, however, we show that the protocol has an extractor and, given a (T_1, T_2) pair, can be simulated. The correctness of the protocol follows from Theorem 7.4.4.

Lemma 7.4.5. *For any $(\hat{u}, \hat{v}, T_1, T_2)$, Transcripts of Protocol 2 can be simulated.*

Proof. Choose challenge $c \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$. Select $s_\alpha \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, and set $R_1 \leftarrow u^{s_\alpha}/T_1^c$. Then equation (7.7) is satisfied. With α and c fixed, a choice for either of r_α or s_α determines the other, and a uniform random choice of one gives a uniform random choice of the other. Therefore s_α and R_1 are distributed as in a real transcript.

Select $s_x \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$. Now, A and α are fixed by T_1 and T_2, x is implicitly fixed by the SDH equation for A, r_x is fixed by x and s_x, and δ is fixed as $x\alpha$. Select $s_\delta \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$; a uniform distribution on this gives a uniform distribution on r_δ. Set $R_3 \leftarrow T_1^{s_x} u^{-s_\delta}$. Again, all the computed values are distributed as in a real transcript. Finally, set

$$R_2 \leftarrow e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \cdot \Big(\frac{e(T_2, w)}{e(g_1, g_2)}\Big)^c \ .$$

This R_2 satisfies (7.8), so it, too, is properly distributed.

Finally, the simulator outputs the transcript $(\hat{u}, \hat{v}, T_1, T_2, R_1, R_2, R_3, c, s_\alpha, s_\beta, s_x, s_\delta)$. As argued above, this transcript is distributed identically to transcripts of actual Protocol 2 interactions for the given $(\hat{u}, \hat{v}, T_1, T_2)$. □

Lemma 7.4.6. *There exists an extractor for Protocol 2 that extracts an SDH pair from a convincing prover.*

Proof. Suppose that an extractor can rewind a prover in the protocol above. The verifier sends $\hat{u}, \hat{v}$ to the prover. Let $u = \psi(\hat{u})$ and $v = \psi(\hat{v})$. The prover then sends T_1, T_2 and R_1, R_2, R_3. To challenge value c, the prover responds with s_α, s_x, and s_δ. To challenge value $c' \neq c$, the prover responds with s'_α, s'_x, and s'_δ. If the prover is convincing, all three verification equations hold for each set of values.

For brevity, let $\Delta c = c - c'$, $\Delta s_\alpha = s_\alpha - s'_\alpha$, and similarly for Δs_x, and Δs_δ.

Consider (7.7) above. Dividing the two instances of this equation, we obtain $u^{\Delta s_\alpha} = T_1^{\Delta c}$. The exponents are in a group of known prime order, so we can take roots; let $\tilde{\alpha} = \Delta s_\alpha / \Delta c$. Then $u^{\tilde{\alpha}} = T_1$.

Now consider (7.9) above. Dividing the two instances gives $T_1^{\Delta s_x} = u^{\Delta s_\delta}$. Substituting $T_1 = u^{\tilde{\alpha}}$ gives $u^{\tilde{\alpha}\Delta s_x} = u^{\Delta s_\delta}$, or $\Delta s_\delta = \tilde{\alpha}\Delta s_x$.

Finally, dividing the two instances of (7.8), we obtain

$$\big(e(g_1, g_2)/e(T_2, w)\big)^{\Delta c} = e(T_2, g_2)^{\Delta s_x} \cdot e(v, w)^{-\Delta s_\alpha} \cdot e(v, g_2)^{-\tilde{\alpha}\Delta s_x} \ .$$

Taking Δc-th roots, and letting $\tilde{x} = \Delta s_x / \Delta c$, we obtain

$$e(g_1, g_2)/e(T_2, w) = e(T_2, g_2)^{\tilde{x}} \cdot e(v, w)^{-\tilde{\alpha}} \cdot e(v, g_2)^{-\tilde{x}\tilde{\alpha}} \ .$$

This can be rearranged as

$$e(g_1, g_2) = e(T_2 v^{-\tilde{\alpha}}, w g_2^{\tilde{x}}) \ ,$$

or, letting $\tilde{A} = T_2 v^{-\tilde{\alpha}}$,

$$e(\tilde{A}, w g_2^{\tilde{x}}) = e(g_1, g_2) \ .$$

Thus the extractor obtains an SDH tuple $(\tilde{A}, \tilde{x})$. Moreover, the $\tilde{A}$ in this SDH tuple is, perforce, the same as that in the ElGamal encryption (T_1, T_2). In other words, the extractor recovers the same A that a revocation-checker matches. □

7.4.4 Proof of Security

We show that the BS scheme described in Section 7.4.2 is a VLR group signature scheme. Correctness was demonstrated in Theorem 7.4.4, above. Below we give proofs of selfless-anonymity and traceability, as defined in Section 7.4.1.

Selfless-Anonymity

Lemma 7.4.7. *The BS VLR group signature scheme in (G_1, G_2) has $(t, q_H, q_S, n, \epsilon)$ selfless anonymity in the random oracle model assuming the (t, ϵ') Decision Linear assumption holds in the group G_2 for $\epsilon' = \frac{\epsilon}{2}\left(\frac{1}{n^2} - \frac{q_S q_H}{p}\right) \approx \epsilon/2n^2$.*

Proof. Suppose algorithm $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks the selfless anonymity of the BS VLR group signature scheme. We build an algorithm $\mathcal{B}$ that breaks the Decision Linear assumption in G_2. Algorithm $\mathcal{B}$ is given as input a 6-tuple $(u_0,\, u_1,\, v,\, h_0 = u_0^a,\, h_1 = u_1^b,\, Z) \in G_2^6$ where $u_0, u_1, v \overset{\text{R}}{\leftarrow} G_2,\ a, b \overset{\text{R}}{\leftarrow} \mathbb{Z}_p^*$ and either $Z = v^{a+b} \in G_2$ or Z is random in G_2. Algorithm $\mathcal{B}$ decides which Z it was given by interacting with $\mathcal{A}$ as follows:

Setup. Recall that g_1, g_2 are the fixed generators of G_1, G_2 respectively. Algorithm $\mathcal{B}$ does the following:

1. Algorithm $\mathcal{B}$ picks a random $\gamma \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$ and sets $w = g_2^\gamma$. It gives $\mathcal{A}$ the $gpk = (g_1, g_2, w)$.
2. $\mathcal{B}$ picks two random users $i_0, i_1 \overset{\text{R}}{\leftarrow} \{1, \ldots, n\}$ and keeps i_0, i_1 secret. For all users $j \neq i_0, i_1$ it generates private keys $\boldsymbol{gsk}[j] = (A_j, x_j)$ using γ and the standard key generation algorithm.
3. $\mathcal{B}$ picks a random $W \overset{\text{R}}{\leftarrow} G_2$.

To give some intuition for the rest of the simulation we define $A_{i_0} = ZW/v^a$ and $A_{i_1} = Wv^b$. In what follows, $\mathcal{B}$ will behave as if user i_0's private key is (A_{i_0}, x_{i_0}) and user i_1's private key is (A_{i_1}, x_{i_1}) for some $x_{i_0}, x_{i_1} \in \mathbb{Z}_p$. We emphasize that $\mathcal{B}$ does not know either A_{i_0} or A_{i_1} since it doesn't know a or b. Observe that if $Z = v^{a+b}$ then

$$A_{i_0} = ZW/v^a = v^{a+b}W/v^a = Wv^b = A_{i_1}$$

Hence, if $Z = v^{a+b}$ users i_0 and i_1 have the same private key. But if Z is random in G_2 then i_0, i_1 have independent private keys.

Hash queries. At any time, $\mathcal{A}$ can query the hash functions H and H_0. Algorithm $\mathcal{B}$ responds with random values while ensuring consistency.

Phase 1. Algorithm $\mathcal{A}$ can issue signing queries, corruption queries, and revocation queries. If a query is for user $i \neq i_0, i_1$ then $\mathcal{B}$ uses the private key $\boldsymbol{gsk}[i]$ to respond to the query. For queries for users i_0 or i_1 algorithm $\mathcal{B}$ responds as follows:

- Signing queries: given a message $M \in \{0,1\}^*$ and a user $i \in \{i_0, i_1\}$ algorithm $\mathcal{B}$ must generate a signature for M using user i's private key.

– To generate a signature for user $i = i_0$, $\mathcal{B}$ picks random $s, t, l \xleftarrow{\text{R}} \mathbb{Z}_p^*$ and makes the following assignments:

$$T_1 \leftarrow h_0 u_0^s \qquad T_2 \leftarrow ZWv^s h_0^t u_0^{st} \qquad \hat{u} \leftarrow u_0^l \qquad \hat{v} \leftarrow (vu_0^t)^l \ .$$

Let $\alpha = (a+s)/l \in \mathbb{Z}_p$. Then $T_1 = \hat{u}^\alpha$ and $T_2 = A_{i_0} \cdot \hat{v}^\alpha$.

– To generate a signature for user $i = i_1$, $\mathcal{B}$ picks random $s, t, l \xleftarrow{\text{R}} \mathbb{Z}_p^*$ and makes the following assignments:

$$T_1 \leftarrow h_1 u_1^s \qquad T_2 \leftarrow Wh_1^t u_1^{st}/v^s \qquad \hat{u} \leftarrow u_1^l \qquad \hat{v} \leftarrow (u_1^t/v)^l \ .$$

Let $\alpha = (b+s)/l \in \mathbb{Z}_p$. Then $T_1 = \hat{u}^\alpha$ and $T_2 = A_{i_1} \cdot \hat{v}^\alpha$.

Either way, $T_1 = \hat{u}^\alpha$ and $T_2 = A_i \hat{v}^\alpha$ for some random $\alpha \in \mathbb{Z}_p$ and random and independent $\hat{u}, \hat{v} \in G_2$. Algorithm $\mathcal{B}$ next picks random $r, c, s_\alpha, s_x, s_\delta \xleftarrow{\text{R}} \mathbb{Z}_p$ and computes the corresponding R_1, R_2, R_3 using equations (7.5). In the unlikely event that $\mathcal{A}$ has already issued a hash query either for $H(gpk, M, r, \psi(T_1), \psi(T_2), R_1, R_2, R_3)$ or for $H_0(gpk, M, r)$, $\mathcal{B}$ reports failure and terminates. Since r is random in $\mathbb{Z}_p$ this happens with probability at most q_H/p. Otherwise, $\mathcal{B}$ defines

$$H(gpk, M, r, \psi(T_1), \psi(T_2), R_1, R_2, R_3) = c$$
$$H_0(gpk, M, r) = (\hat{u}, \hat{v}) \ .$$

Algorithm $\mathcal{B}$ then computes the signature σ as $\sigma \leftarrow (r, \psi(T_1), \psi(T_2), c, s_\alpha, s_x, s_\delta)$, and gives σ to $\mathcal{A}$. Note that by Lemma 7.4.5, σ is a properly distributed signature under user i's private key.

- Corruption queries and revocation queries: if $\mathcal{A}$ ever issues a corruption of revocation query for users i_0 or i_1 then $\mathcal{B}$ reports failure and aborts.

Challenge Algorithm $\mathcal{A}$ outputs a message M and two users i_0^* and i_1^* where it wishes to be challenged. if $\{i_0^*, i_1^*\} \neq \{i_0, i_1\}$ then $\mathcal{B}$ reports failure and aborts. Otherwise, we assume $i_0^* = i_0$ and $i_1^* = i_1$. Algorithm $\mathcal{B}$ picks a random $b \xleftarrow{\text{R}} \{0,1\}$ and generates a signature σ^* under user i_b's key for M using the same method used to respond to signing queries in Phase 1. It gives σ^* as the challenge to $\mathcal{A}$.

Phase 2. Algorithm $\mathcal{A}$ issues restricted queries. $\mathcal{B}$ responds as in Phase 1.

Output. Eventually, $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b. If $b = b'$ then $\mathcal{B}$ outputs 0 (indicating that Z is random in G_2); otherwise $\mathcal{B}$ outputs 1 (indicating that $Z = v^{a+b}$).

Suppose $\mathcal{B}$ does not abort during the simulation. Then, when Z is random in G_2 algorithm $\mathcal{B}$ emulates the selfless-anonymity game perfectly. Hence, $\Pr[b = b'] > \frac{1}{2} + \epsilon$. When $Z = v^{a+b}$ then the private keys for users i_0 and i_1 are identical and therefore the challenge signature σ^* is independent of b. It follows that $\Pr[b = b'] = 1/2$. Therefore, assuming $\mathcal{B}$ does not abort, it has advantage at least $\epsilon/2$ in solving the given linear challenge $(u_0, u_1, v, h_0, h_1, Z) \in G_2^6$.

Algorithm $\mathcal{B}$ does not abort if it correctly guesses the values i_0^* and i_1^* during the setup phase and none of the signature queries cause it to abort. The probability that a given signature query causes $\mathcal{B}$ to abort is at most q_H/p and therefore the probability that $\mathcal{B}$ aborts as a result of $\mathcal{A}$'s signature queries is at most $q_S q_H/p$. As long as $\mathcal{B}$ does not abort during phase 1, algorithm $\mathcal{A}$ gets no information about the choice of i_0, i_1. Therefore, the probability that the query pattern during phase 1 and the choice of challenge do not cause $\mathcal{B}$ to abort is at least $1/n^2$. It now follows that $\mathcal{B}$ solves the given linear challenge with advantage at least $\frac{\epsilon}{2}\left(\frac{1}{n^2} - \frac{q_S q_H}{p}\right)$, as required. $\square$

Traceability

Theorem 7.4.8. *If SDH is (q, t', ϵ')-hard on (G_1, G_2), then the BS VLR group signature scheme is $(t, q_H, q_S, n, \epsilon)$-traceable, where $n = q - 1$, $\epsilon = 4n\sqrt{2\epsilon' q_H} + n/p$, and $t = \Theta(1) \cdot t'$.*

Proof. Our proof proceeds in three parts. First, we describe a framework for interacting with an algorithm that wins a traceability game. Second, we show how to instantiate this framework appropriately for different types of such breaker algorithms. Third, we show how to apply the forking lemma [103] to the framework instances, obtaining SDH solutions.

Interaction Framework Suppose we are given an algorithm $\mathcal{A}$ that breaks the traceability of the BS scheme. We describe a framework for interacting with $\mathcal{A}$.

Setup. We are given groups (G_1, G_2) as above, with respective generators g_1 and g_2. We are also given $w = g_2^{\gamma} \in G_2$, and a list of pairs (A_i, x_i) for $i = 1, \ldots, n$. For each i, either $x_i = \star$, indicating that the x_i corresponding to A_i is not known, or else (A_i, x_i) is an SDH pair, and $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$. We then run $\mathcal{A}$, giving it the group public key (g_1, g_2, w) and the users' revocation tokens $\{A_i\}$. We answer its oracle queries as follows.

Hash Queries. At any time, $\mathcal{A}$ can query the hash functions to obtain generators $(\hat{u}, \hat{v})$ or challenge c. We respond with random values while maintaining consistency. made again.

Signature Queries. Algorithm $\mathcal{A}$ asks for a signature on message M by a key at index i. If $s_i \neq \star$, we follow the group signing procedure with key (A_i, x_i) to obtain a signature σ on M, and return σ to $\mathcal{A}$.

Otherwise $s_i = \star$. We pick a nonce $r \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$, obtain generators $(\hat{u}, \hat{v}) \leftarrow H_0(gpk, M, r)$, and set $u \leftarrow \psi(\hat{u})$ and $v \leftarrow \psi(\hat{v})$. We then pick $\alpha \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$, set $T_1 \leftarrow u^{\alpha}$, and $T_2 \leftarrow Ag_1^{\alpha}$ and run the Protocol 2 simulator with values $(\hat{u}, \hat{v}, T_1, T_2)$. The simulator returns a transcript $(\hat{u}, \hat{v}, T_1, T_2, R_1, R_2, R_3, c, s_\alpha, s_x, s_\delta)$, from which we derive a VLR group signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$. In addition, we must patch the hash oracle at $(M, r, T_1, T_2, R_1, R_2, R_3)$ to equal c. If this causes a collision, i.e., if we previously set the oracle at this point to some other c', we declare failure and exit. Otherwise, we return σ to $\mathcal{A}$. A signature query can trigger a hash query, which we charge against $\mathcal{A}$'s hash query limit to simplify the accounting.

Private Key Queries. Algorithm $\mathcal{A}$ asks for the private key of the user at some index i. If $x_i \neq \star$, we return (A_i, x_i) to $\mathcal{A}$. Otherwise, we declare failure and exit.

Output. Finally, if algorithm $\mathcal{A}$ is successful, it outputs a forged VLR group signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$ on a message M with nonce r, along with a revocation list $\boldsymbol{rl}^*$. We apply the implicit revocation algorithm, with revocation tokens $\{A_i\}$ to determine which A^* is encoded in (T_1, T_2). This A^* cannot be on $\boldsymbol{rl}^*$; if it were, the signature would have been rejected as invalid. Thus for the forgery to be nontrivial, A^* must also be outside the adversary's coalition U. If A^* does not equal A_i for any i, we output σ. Otherwise, $A^* = A_{i^*}$ for some i^*. If $s_{i^*} = \star$, we output σ. If, however, $s_{i^*} \neq \star$, we declare failure and exit.

As implied by the output phase of the framework above, there are two types of forger algorithm. Type I forgers output a forgery σ on a message M that encodes some identity $A^* \notin \{A_1, \ldots, A_n\}$. Type II forgers output a forgery that encodes an identity A^* such that $A^* = A_{i^*}$ for some i^*, and the forger did not make a private-key oracle query at i^*. We treat these two types of forger differently.

Given a q-SDH instance $(g_1', g_2', (g_2')^{\gamma}, (g_2')^{\gamma^2}, \ldots, (g_2')^{\gamma^q})$, we apply the technique of Boneh and Boyen's Lemma 3.2 [23] that we recalled in Section 3.4.1, obtaining generators $g_1 \in G_1$, $g_2 \in G_2$, $w = g_2^{\gamma}$, along with $q-1$ SDH pairs (A_i, x_i) such that $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$ for each i. Any SDH pair (A, x) besides these $q-1$ pairs can be transformed into a solution to the original q-SDH instance, again using Boneh and Boyen's Lemma 3.2.

Type I Forger Against a $(t, q_H, q_S, n, \epsilon)$-Type I forger $\mathcal{A}$, we turn an instance of $(n+1)$-SDH into values (g_1, g_2, w), and n SDH pairs (A_i, x_i). We then apply the framework to $\mathcal{A}$ with these values. Algorithm $\mathcal{A}$'s environment is perfectly simulated, and the framework succeeds whenever $\mathcal{A}$ succeeds, so we obtain a Type I forgery with probability ϵ.

Type II Forger Against a $(t, q_H, q_S, n, \epsilon)$-Type II forger $\mathcal{A}$, we turn an instance of n-SDH into values (g_1, g_2, w), and $n-1$ SDH pairs. These pairs we distribute amongst n pairs (A_i, x_i). The unfilled entry at random index i^* we fill as follows. Pick $A_{i^*} \overset{\text{R}}{\leftarrow} G_1$, and set $x_{i^*} \leftarrow \star$, a placeholder value. Now we run $\mathcal{A}$ under the framework. The framework declares success only if $\mathcal{A}$ never queries the private key oracle at i^*, but forges a group signature that traces to A_{i^*}. It is easy to see that the framework simulation is perfect unless $\mathcal{A}$ queries the private key oracle at i^*. Because the protocol simulator invoked by the signing oracle produces group signatures that are indistinguishable from those of a user whose SDH tuple includes A_{i^*}, the value of i^* is independent of $\mathcal{A}$'s view unless and until it queries the private key oracle at i^*. (Since the hash oracle takes as input five elements of G_1 or G_2 besides the message M, the probability of collision in simulated signing queries is bounded above by $(q_H q_S + q_S^2)/p^9$. Assuming $q_S \ll q_H \ll p = |G_1|$, this probability is negligible, and we ignore it in the analysis.) Finally, when $\mathcal{A}$ outputs its forgery σ, implicating some user i whose private key $\mathcal{A}$ has not requested, the value of i^* (amongst the users whose keys it has not requested) remains independent of $\mathcal{A}$'s view. It is easy to see, then, that $\mathcal{A}$ outputs a forged group signature that traces to user i^* with probability at least ϵ/n.

Application of Forger Now we show how to use the application of our framework to a Type I or Type II adversary $\mathcal{A}$ to obtain another SDH pair, contradicting the SDH assumption. The remainder of this proof follows closely the methodology and notation of the forking lemma [103].

Let $\mathcal{A}$ be a forger (of either type) for which the framework succeeds with probability ϵ'. From here on, we abbreviate signatures as $(M, \sigma_0, c, \sigma_1)$, where $\sigma_0 = (r, \hat{u}, \hat{v}, T_1, T_2, R_1, R_2, R_3)$, the values given, along with M, to the random oracle H, and from which c is derived, and where $\sigma_1 = (s_\alpha, s_x, s_\delta)$. (Those values normally omitted from the signature can be recovered as the verification algorithm in Section 7.4.2 does.)

We require that $\mathcal{A}$ always query H_0 at (M, r) before querying H at $(M, r, \ldots)$. Any adversary can be modified mechanically into satisfying this condition. This technical requirement means that, even if in rewinding we change the value of $H(M, r, \ldots)$, the value of $H_0(M, r)$, and therefore of the u and v used implicitly in the arguments of the H call, remains unchanged.

For any fixed f_0 vector of H_0 responses, a run of the framework on $\mathcal{A}$ is completely described by the randomness string ω used by the framework and $\mathcal{A}$, by the vector f_0 of responses made by the H_0 hash oracle, and by the vector f of responses made by the H hash oracle. Let S be the set of tuples (ω, f_0, f) such that the framework, invoked on $\mathcal{A}$, completes successfully with forgery $(M, \sigma_0, c, \sigma_1)$, and $\mathcal{A}$ queried the hash oracle H on (M, σ_0). In this case, let $\mathrm{Ind}(\omega, f_0, f)$ be the index of f at which $\mathcal{A}$ queried (M, σ_0). We define $\nu = \Pr[S] = \epsilon' - 1/p$, where the $1/p$ term accounts for the possibility that $\mathcal{A}$ guessed the hash of (M, σ_0) without the hash oracle's help. For each j, $1 \le j \le q_H$, let S_j be the set of tuples (ω, f_0, f) as above and such that $\mathrm{Ind}(\omega, f_0, f) = j$. Let J be the set of auspicious indices j such that $\Pr[S_j \mid S] \ge 1/(2q_H)$. Then

$$\Pr\big[\mathrm{Ind}(\omega, f) \in J \mid S\big] \ge 1/2 \ .$$

Let $f|_a^b$ be the restriction of f to its elements at indices $a, a+1, \ldots, b$. For each $j \in J$, we consider the heavy-rows lemma [103, Lemma 1] with rows $X = (\omega, f_0, f|_1^{j-1})$ and columns $Y = (f|_j^{q_H})$. Clearly $\Pr_{(x,y)}[(x, y) \in S_j] \ge \nu/(2q_H)$. Let the heavy rows Ω_j be those rows such that, $\forall (x, y) \in \Omega_j : \Pr_{y'}[(x, y') \in S_j] \ge \nu/(4q_H)$. Then, by the heavy-rows lemma, $\Pr[\Omega_j \mid S_j] \ge 1/2$. A simple argument then shows that $\Pr[\exists j \in J : \Omega_j \cap S_j \mid S] \ge 1/4$.

Thus, with probability $\nu/4$, the framework, invoked on $\mathcal{A}$, succeeds and obtains a forgery $(M, \sigma_0, c, \sigma_1)$ that derives from a heavy row $(x, y) \in \Omega_j$ for some $j \in J$, i.e., an execution (ω, f_0, f) such that

$$\Pr_{f'}\big[(\omega, f_0, f') \in S_j \mid f'|_1^{j-1} = f|_1^{j-1}\big] \ge \nu/(4q_H) \ .$$

If we now rewind the framework and $\mathcal{A}$ to the jth query and proceed with an oracle vector f' that differs from f from the jth entry on, we obtain, with probability at least $\nu/(4q_H)$, a successful framework completion and a second forgery $(M, \sigma_0, c', \sigma_1')$, with (M, σ_0) still queried at $\mathcal{A}$'s jth hash query. Since the adversary queried H_0 at (M, r) (where r is the first element of σ_0) before he made his jth H oracle query, the values of $\hat{u}$ and $\hat{v}$ in these two forgeries will be the same.

By using the extractor of Lemma 7.4.6, we obtain from the forgeries (σ_0, c, σ_1) and $(\sigma_0, c', \sigma_1')$ an SDH tuple (A, x). The extracted A is the same as the A encoded in (T_1, T_2) in σ_0. The framework declares success only when the A encoded in (T_1, T_2) is not amongst those whose x it knows. Therefore, the extracted SDH tuple (A, x) is not amongst those that we ourselves created, and can be transformed, again following the technique of Boneh and Boyen's Lemma 3.2 [23], to an answer to the posed q-SDH problem.

Putting everything together, we have proved the following claims.

Claim 20. *Using a $(t, q_H, q_S, n, \epsilon)$ Type I forger, we solve an instance of $(n+1)$-SDH with probability $(\epsilon - 1/p)^2/(16q_H)$ in time $\Theta(1) \cdot t$.*

Claim 21. *Using a $(t, q_H, q_S, n, \epsilon)$ Type II forger, we solve an instance of n-SDH with probability $(\epsilon/n - 1/p)^2/(16q_H)$ in time $\Theta(1) \cdot t$.*

We can guess which of the two forger types a particular forger is with probability 1/2; then assuming the more pessimistic scenario of Claim 2 proves the theorem. □

7.4.5 Efficient Revocation for BS Signatures

In the BS scheme (Section 7.4.2), signature verification time grows linearly in the number of revoked users. It is desirable to have a Verifier-Local Revocation system where verification time is constant. In this section we describe a simple modification to the signing and verification algorithms that achieves this at the cost of slightly reduced anonymity.

Consider how our system is used for privacy-preserving attestation: Users connect to various web sites and at each site they perform a private attestation using the group signature issued by the tamper-resistant chip in their machine. For an efficient revocation check, when the chip issues a signature for attesting to a site S it uses the signing algorithm from Section 7.4.2 with the small modification that the parameters u and v are generated as:

$$(u, v) \overset{\text{R}}{\leftarrow} H_0(gpk, S, r)$$

where r is random in the range $\{1, \ldots, k\}$ and k is a security parameter (e.g., $k = 128$). Note that, unlike Section 7.4.2, here (u, v) do not depend on the message being signed. Hence, at a given site S there are only k possible values for the pair (u, v).

Now, suppose site S has been supplied with a revocation list $\boldsymbol{rl} = \{A_1, \ldots, A_b\}$. To verify that a signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$ was not issued by a revoked user the site uses the same procedure as in Section 7.4.2:

1. Compute $(u, v) \overset{\text{R}}{\leftarrow} H_0(gpk, S, r)$, and
2. For $i = 1, \ldots, b$ check that $e(T_1, v)e(A_i, u) \neq e(T_2, u)$.

Since at site S there are only k possible values for u, the value $e(A_i, u)$ can be precomputed for the entire $\boldsymbol{rl}$ for all possible u's. Thus, site S stores a $|\boldsymbol{rl}| \times k$ table of values, $e(A_i, u_j)$. To check revocation, it simply does a table-lookup to see if the value $e(T_2, u)/e(T_1, v)$ is in the r'th row of the table. If not, then the signature was not issued by a revoked user. Hence, the revocation check takes time independent of the size of $\boldsymbol{rl}$.

The downside is that the scheme is now only partially anonymous. If the user issues two signatures at site S using the same random value $r \in \{1, \ldots, k\}$ then the site can test that these two signatures came from the same user. However, signatures issued at different sites are still completely unlinkable. Similarly, signatures issued at the same site using different r's are unlinkable (e.g., with $k = 100$ only 1% of signatures at S are linkable). For some applications, this trade-off between partial linkability and efficient revocation might be acceptable.

7.4.6 Backward Unlinkability

When a user is revoked in the BS scheme, all her signatures can be traced, including any issued before her key was compromised. It might be preferable keep these older signatures unlinkable, a property called backward unlinkability [7, 113]. This property is typically achieved by dividing time into intervals. signatures are tied to the interval in which they were issued, and revocation is effective only from the current interval onwards.

Nakanishi and Funabiki have proposed a variant of BS signatures with backward unlinkability using time intervals [94]. They prove the security of their construction under security definitions adapted from those in Section 7.4.1.

7.4.7 Strong Exculpability for BS

In the BS VLR scheme, as in the BBS scheme of Section 6.3, keys are issued by a trusted key generator. This is in keeping with the security definitions given in Section 7.4.1, which themselves are modeled after the Bellare-Micciancio-Warinschi definitions for ordinary group signatures [15].

It is possible to achieve strong exculpability — where even the entity that issues user keys cannot forge signatures under users' keys — for BS signatures as well. The necessary modifications are essentially those suggested for BBS in Section 7.2.

An appropriate model for proving the modified BS signatures secure would closely resemble the definitions for traceable signatures proposed by Kiayias and Yung [73]. ("Claiming" a signature, in the Kiayias-Yung terminology, would simply require proving knowledge of a value A_i such that $(\hat{u}, \hat{v}, T_1, T_2/A_i)$ is a co-Diffie-Hellman tuple.)

7.5 Conclusions and Open Problems

We have described extensions to the BBS group signature scheme to achieve strong exculpability and (traditional) revocation. In addition, we have described a short group signature scheme where user revocation only requires sending revocation information to signature verifiers, a setup we call verifier-local revocation. Our signatures are short: only 141 bytes for a standard security level. They are shorter than group signatures built from the Strong-RSA assumption and are shorter even than BBS short group signatures.

There are still a number of open problems related to VLR signatures. Most importantly, is there an efficient VLR group signature scheme where signature verification time is sub-linear in the number of revoked users, without compromising user privacy?

Bibliography

[1] M. Abdalla, J. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In L. Knudsen, editor, *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 418–33. Springer-Verlag, May 2002.

[2] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In L. Knudsen, editor, *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 83–107. Springer-Verlag, May 2002.

[3] ANSI X9 Committee. DSTU X9.59-2000: Electronic commerce for the financial services industry: Account-based secure payment objects. Online: `http://www.x9.org/`.

[4] ANSI X9.62 and FIPS 186-2. Elliptic curve digital signature algorithm, 1998.

[5] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE J. Selected Areas in Comm.*, 18(4):593–610, Apr. 2000.

[6] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 255–70. Springer-Verlag, Aug. 2000.

[7] G. Ateniese and G. Tsudik. Some open issues and directions in group signatures. In M. Franklin, editor, *Proceedings of Financial Cryptography 1999*, volume 1648 of *LNCS*, pages 196–211. Springer-Verlag, Feb. 1999.

[8] G. Ateniese, G. Tsudik, and D. Song. Quasi-efficient revocation of group signatures. In M. Blaze, editor, *Proceedings of Financial Cryptography 2002*, volume 2357 of *LNCS*, pages 183–97. Springer-Verlag, 2003.

[9] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with offline TTP. In P. Karger and L. Gong, editors, *Proceedings of IEEE Security & Privacy*, pages 77–85, May 1998.

[10] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In W. Fumy, editor, *Proceedings of Eurocrypt 1997*, volume 1233 of *LNCS*, pages 480–494. Springer-Verlag, May 1997.

[11] P. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. `http://eprint.iacr.org/`.

[12] P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptology*, 17(4):321–34, Sept. 2004.

[13] P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Proceedings of SAC 2005*, volume 3897 of *LNCS*, pages 319–31. Springer-Verlag, 2006.

[14] M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Proceedings of Eurocrypt 1998*, volume 1403 of *LNCS*, pages 236–50. Springer-Verlag, May–Jun. 1998.

[15] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 614–29. Springer-Verlag, May 2003.

[16] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. Denning, R. Pyle, R. Ganesan, R. Sandhu, and V. Ashby, editors, *Proceedings of CCS 1993*, pages 62–73. ACM Press, Nov. 1993.

[17] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In U. Maurer, editor, *Proceedings of Eurocrypt 1996*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, May 1996.

[18] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In A. J. Menezes, editor, *Proceedings of CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–53. Springer-Verlag, Feb. 2005.

[19] M. Bellare and M. Yung. Certifying permutations: Non-interactive zero-knowledge based on any trapdoor permutation. *J. Cryptology*, 9(1):149–66, 1996.

[20] I. F. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Notes.* Cambridge University Press, 1999.

[21] J. Blum. *CARAVAN: A Communication Architecture for Reliable Adaptive Vehicular Ad hoc Networks.* PhD thesis, George Washington University, May 2005.

[22] A. Boldyreva. Threshold signature, multisignature and blind signature schemes based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer-Verlag, Jan. 2003.

[23] D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 56–73. Springer-Verlag, May 2004.

[24] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, Aug. 2004.

[25] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Computing*, 32(3):586–615, 2003. Extended abstract in *Proceedings of Crypto 2001.*

[26] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 416–32. Springer-Verlag, May 2003.

[27] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–32. Springer-Verlag, Dec. 2001.

[28] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, Sept. 2004. Extended abstract in *Proceedings of Asiacrypt 2001*.

[29] D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In B. Pfitzmann and P. Liu, editors, *Proceedings of CCS 2004*, pages 168–77. ACM Press, Oct. 2004.

[30] E. Brickell. An efficient protocol for anonymously providing assurance of the container of a private key, Apr. 2003. Submitted to the Trusted Computing Group.

[31] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Proceedings of Eurocrypt 1999*, volume 1592 of *LNCS*, pages 402–14. Springer-Verlag, May 1999.

[32] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 61–76. Springer-Verlag, Aug. 2002.

[33] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 56–72. Springer-Verlag, Aug. 2004.

[34] CAMP Vehicle Safety Communications Consortium. Vehicle safety communications project task 3 final report, Mar. 2005. Online: `http://www-nrd.nhtsa.dot.gov/pdf/nrd-12/1665CAMP3web/`.

[35] D. Chaum and T. Pedersen. Wallet databases with observers. In E. Brickell, editor, *Proceedings of Crypto 1992*, volume 740 of *LNCS*, pages 89–105. Springer-Verlag, Aug. 1992.

[36] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *Proceedings of Eurocrypt 1991*, volume 547 of *LNCS*, pages 257–65. Springer-Verlag, Apr. 1991.

[37] B. Chevallier-Mames. An efficient CDH-based signature scheme with a tight security reduction. In V. Shoup, editor, *Proceedings of Crypto 2005*, volume 3621 of *LNCS*, pages 511–26. Springer-Verlag, Aug. 2005.

[38] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Tran. Info. Th.*, 30(4):587–94, 1984.

[39] J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 229–35. Springer-Verlag, Aug. 2000.

[40] J.-S. Coron. Optimal security proofs for PSS and other signature schemes. In L. Knudsen, editor, *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 272–87. Springer-Verlag, May 2002.

[41] J.-S. Coron. Security proof for partial-domain hash signature schemes. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 613–26. Springer-Verlag, Aug. 2002.

[42] J.-S. Coron and D. Naccache. Boneh et al.'s k-element aggregate extraction assumption is equivalent to the Diffie-Hellman assumption. In C. S. Laih, editor, *Proceedings of Asiacrypt 2003*, volume 2894 of *LNCS*, pages 392–7. Springer-Verlag, Dec. 2003.

[43] N. Courtois, M. Daum, and P. Felke. On the security of HFE, HFEv- and Quartz. In Y. Desmedt, editor, *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 337–50. Springer-Verlag, Jan. 2003.

[44] R. Cramer and V. Shoup. A practical public key encryption system provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Proceedings of Crypto 1998*, volume 1642 of *LNCS*, pages 13–25. Springer-Verlag, Aug. 1998.

[45] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Trans. Info. & System Security*, 3(3):161–85, 2000.

[46] C. Diem. The GHS attack in odd characteristic. *J. Ramanujan Math. Soc.*, 18(1):1–32, 2003.

[47] X. Ding, G. Tsudik, and S. Xu. Leak-free group signatures with immediate revocation. In T. Lai and K. Okada, editors, *Proceedings of ICDCS 2004*, Mar. 2004.

[48] Y. Dodis. Efficient construction of (distributed) verifiable random functions. In Y. Desmedt, editor, *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 1–17. Springer-Verlag, Jan. 2003.

[49] Y. Dodis and L. Reyzin. On the power of claw-free permutations. In S. Cimato, C. Galdi, and G. Persiano, editors, *Proceedings of SCN 2002*, volume 2576 of *LNCS*, pages 55–73. Springer-Verlag, Sept. 2002.

[50] A. Fiat. Batch RSA. In G. Brassard, editor, *Proceedings of Crypto 1989*, volume 435 of *LNCS*, pages 175–85. Springer-Verlag, Aug. 1989.

[51] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Proceedings of Crypto 1986*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, Aug. 1986.

[52] G. Frey, M. Muller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Trans. Info. Th.*, 45(5):1717–9, 1999.

[53] S. Galbraith. Pairings. In I. F. Blake, G. Seroussi, and N. Smart, editors, *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Notes*, chapter IX, pages 183–213. Cambridge University Press, 2005.

[54] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D. Kohel, editors, *Proceedings of ANTS V*, volume 2369 of *LNCS*, pages 324–37. Springer-Verlag, July 2002.

[55] S. Galbraith and V. Rotger. Easy decision-Diffie-Hellman groups. *LMS J. Comput. Math*, 7:201–18, aug 2004.

[56] S. Galbraith and N. Smart. A cryptographic application of Weil descent. In M. Walker, editor, *Proceedings of Cryptography and Coding 1999*, volume 1746 of *LNCS*, pages 191–200. Springer-Verlag, Dec. 1999.

[57] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In M. Wiener, editor, *Proceedings of Crypto 1999*, volume 1666 of *LNCS*, pages 449–66. Springer-Verlag, Aug. 1999.

[58] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In L. Peterson, editor, *Proceedings of SOSP 2003*, pages 193–206. ACM Press, Oct. 2003.

[59] P. Gaudry, F. Hess, and N. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *J. Cryptology*, 15(1):19–46, 2002.

[60] P. Gemmel. An introduction to threshold cryptography. *RSA CryptoBytes*, 2(3):7–12, Winter 1997.

[61] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *Proceedings of Eurocrypt 1999*, volume 1592 of *LNCS*, pages 295–310. Springer-Verlag, May 1999.

[62] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. Robust and efficient sharing of RSA functions. *J. Cryptology*, 13(2):273–300, 2000.

[63] C. Gentry. How to compress Rabin ciphertexts and signatures (and more). In M. Franklin, editor, *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 179–200. Springer-Verlag, Aug. 2004.

[64] M. Girault. Self-certified public keys. In D. W. Davies, editor, *Proceedings of Eurocrypt 1991*, volume 547 of *LNCS*, pages 490–7. Springer-Verlag, Apr. 1991.

[65] E.-J. Goh and S. Jarecki. A signature scheme as secure as the Diffie-Hellman problem. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 401–15. Springer-Verlag, May 2003.

[66] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, 1988.

[67] R. Hayashi, T. Okamoto, and K. Tanaka. An RSA family of trap-door permutations with a common domain and its applications. In F. Bao, R. H. Deng, and J. Zhou, editors, *Proceedings of PKC 2004*, volume 2947 of *LNCS*, pages 291–304. Springer-Verlag, Mar. 2004.

[68] F. Hess. On the security of the verifiably encrypted signature scheme of Boneh, Gentry, Lynn and Shacham. *Info. Processing Letters*, 89(3):111–4, Feb. 2004.

[69] ISO TC86 Committee. ISO 8583: Financial transaction card originated messages — interchange message specifications. Online: `http://www.tc68.org/`.

[70] A. Joux and K. Nguyen. Separating decision Diffie-Hellman from computational Diffie-Hellman in cryptographic groups. *J. Cryptology*, 16(4):239–47, Sept. 2003.

[71] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In V. Atluri and T. Jaeger, editors, *Proceedings of CCS 2003*, pages 155–64. ACM Press, Oct. 2003.

[72] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (Secure-BGP). *IEEE J. Selected Areas in Comm.*, 18(4):582–92, April 2000.

[73] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In C. Cachin and J. Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 571–89. Springer-Verlag, May 2004.

[74] A. Kiayias and M. Yung. Efficient secure group signatures with dynamic joins and keeping anonymity against group managers. In E. Dawson and S. Vaudenay, editors, *Proceedings of Mycrypt 2005*, volume 3715 of *LNCS*, pages 151–70. Springer-Verlag, Sept. 2005.

[75] A. Kiayias and M. Yung. Group signatures with efficient concurrent join. In R. Cramer, editor, *Proceedings of Eurocrypt 2005*, volume 3494 of *LNCS*, pages 198–214. Springer-Verlag, May 2005.

[76] N. Koblitz. An elliptic curve implementation of the finite field digital signature algorithm. In H. Krawczyk, editor, *Proceedings of Crypto 1998*, volume 1462 of *LNCS*, pages 327–33. Springer-Verlag, Aug. 1998.

[77] S. Lang. *Elliptic Functions.* Addison-Wesley, Reading, MA, 1973.

[78] B. Laurie and N. Bohm. Signatures: an interface between law and technology, Jan. 2003. Online: `http://www.apache-ssl.org/tech-legal.pdf`.

[79] A. Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 597–612. Springer-Verlag, Aug. 2002.

[80] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In C. Cachin and J. Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 74–90. Springer-Verlag, May 2004.

[81] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In H. Heys and C. Adams, editors, *Proceedings of SAC 1999*, volume 1758 of *LNCS*, pages 184–99. Springer-Verlag, Aug. 1999.

[82] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In Y. Desmedt, editor, *Proceedings of Crypto 1994*, volume 839 of *LNCS*, pages 271–81. Springer-Verlag, Aug. 1994.

[83] A. Menezes, T. Okamoto, and P. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Info. Th.*, 39(5):1639–46, 1993.

[84] A. J. Menezes, editor. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[85] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[86] S. Micali, K. Ohta, and L. Reyzin. Provable-subgroup signatures. Unpublished manuscript, 1999.

[87] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures (extended abstract). In P. Samarati, editor, *Proceedings of CCS 2001*, pages 245–54. ACM Press, Nov. 2001.

[88] S. Micali and R. Rivest. Transitive signature schemes. In B. Preneel, editor, *Proceedings of CT-RSA 2002*, volume 2271 of *LNCS*, pages 236–43. Springer-Verlag, Feb. 2002.

[89] V. Miller. The Weil pairing, and its efficient calculation. *J. Cryptology*, 17(4):235–61, Sept. 2004.

[90] I. Mironov. A short signature as secure as DSA. Unpublished manuscript, 2001.

[91] S. Mitsunari, R. Sakai, and M. Kasahara. A new traitor tracing. *IEICE Trans. Fundamentals*, E85-A(2):481–4, Feb. 2002.

[92] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84-A(5):1234–43, May 2001.

[93] D. Naccache and J. Stern. Signing on a postcard. In Y. Frankel, editor, *Proceedings of Financial Cryptography 2000*, volume 1962 of *LNCS*, pages 121–35. Springer-Verlag, 2001.

[94] T. Nakanishi and N. Funabiki. Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps. In B. Roy, editor, *Proceedings of Asiacrypt 2005*, volume 3788 of *LNCS*, pages 533–48. Springer-Verlag, Dec. 2005.

[95] K. Nyberg and R. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. *Designs, Codes and Cryptography*, 7(1–2):61–81, 1996.

[96] K. Ohta and T. Okamoto. Multisignature schemes secure against active insider attacks. *IEICE Trans. Fundamentals*, E82-A(1):21–31, 1999.

[97] T. Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Computer Systems*, 6(4):432–41, November 1988.

[98] T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic primitives. In K. Kim, editor, *Proceedings of PKC 2001*, volume 1992 of *LNCS*, pages 104–18. Springer-Verlag, Feb. 2001.

[99] J. Patarin, N. Courtois, and L. Goubin. QUARTZ, 128-bit long digital signatures. In D. Naccache, editor, *Proceedings of CT-RSA 2001*, volume 2020 of *LNCS*, pages 282–97. Springer-Verlag, Apr. 2001.

[100] K. Paterson. Cryptography from pairings. In I. F. Blake, G. Seroussi, and N. Smart, editors, *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Notes*, chapter X, pages 215–51. Cambridge University Press, 2005.

[101] T. Pedersen. A threshold cryptosystem without a trusted third party. In D. W. Davies, editor, *Proceedings of Eurocrypt 1991*, volume 547 of *LNCS*, pages 522–6. Springer-Verlag, Apr. 1991.

[102] L. Pintsov and S. Vanstone. Postal revenue collection in the digital age. In Y. Frankel, editor, *Proceedings of Financial Cryptography 2000*, volume 1962 of *LNCS*, pages 105–20. Springer-Verlag, 2001.

[103] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–96, 2000.

[104] G. Poupard and J. Stern. Fair encryption of RSA keys. In B. Preneel, editor, *Proceedings of Eurocrypt 2000*, volume 1807 of *LNCS*, pages 172–89. Springer-Verlag, May 2000.

[105] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Commun. ACM*, 21(2):120–6, Feb. 1978.

[106] K. Rubin and A. Silverberg. Supersingular Abelian varieties in cryptology. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 336–53. Springer-Verlag, Aug. 2002.

[107] O. Schirokauer, D. Weber, and T. Denny. Discrete logarithms: The effectiveness of the index calculus method. In H. Cohen, editor, *Proceedings of ANTS II*, volume 1122 of *LNCS*, pages 337–61. Springer-Verlag, May 1996.

[108] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[109] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Proceedings of Eurocrypt 1997*, volume 1233 of *LNCS*, pages 256–66. Springer-Verlag, May 1997.

[110] V. Shoup. Practical threshold signatures. In B. Preneel, editor, *Proceedings of Eurocrypt 2000*, volume 1807 of *LNCS*, pages 207–20. Springer Verlag, May 2000.

[111] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, 1986.

[112] N. Smart and F. Vercauteren. On computable isomorphisms in efficient pairing based systems. Cryptology ePrint Archive, Report 2005/116, 2005. `http://eprint.iacr.org/`.

[113] D. Song. Practical forward secure group signature schemes. In P. Samarati, editor, *Proceedings of CCS 2001*, pages 225–34. ACM Press, Nov. 2001.

[114] R. Steinfeld, L. Bull, H. Wang, , and J. Pieprzyk. Universal designated-verifier signatures. In C. S. Laih, editor, *Proceedings of Asiacrypt 2003*, volume 2894 of *LNCS*, pages 523–42. Springer-Verlag, Dec. 2003.

[115] Trusted Computing Group. Trusted Computing Platform Alliance (TCPA) Main Specification, 2003. Online: `www.trustedcomputinggroup.org`.

[116] G. Tsudik and S. Xu. Accumulating composites and improved group signing. In C. S. Laih, editor, *Proceedings of Asiacrypt 2003*, volume 2894 of *LNCS*, pages 269–86. Springer-Verlag, Dec. 2003.

[117] E. R. Verheul. Self-blindable credential certificates from the Weil pairing. In C. Boyd, editor, *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 533–51. Springer-Verlag, Dec. 2001.

[118] W. C. Waterhouse. Abelian varieties over finite fields. *Ann. Sci. École Norm. Sup.*, 2:521–60, 1969.

[119] B. Waters. Efficient identity-based encryption without random oracles. In R. Cramer, editor, *Proceedings of Eurocrypt 2005*, volume 3494 of *LNCS*, pages 114–27. Springer-Verlag, May 2005.

[120] F. Zhang, R. Safavi-Naini, and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In F. Bao, R. H. Deng, and J. Zhou, editors, *Proceedings of PKC 2004*, volume 2947 of *LNCS*, pages 277–90. Springer-Verlag, Mar. 2004.