On Pairing-Based Signature and Aggregate Signature Schemes

by

Edward Knapp

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics & Optimization

Waterloo, Ontario, Canada, 2008

Abstract

In 2001, Boneh, Lynn, and Shacham presented a pairing-based signature scheme known as the BLS signature scheme. In 2003, Boneh, Gentry, Lynn, and Shacham presented the first aggregate signature scheme called the BGLS aggregate signature scheme. The BGLS scheme allows for N users with N signatures to combine their signatures into a single signature. The size of the resulting signature is independent of N. The BGLS signature scheme enjoys roughly the same level of security as the BLS scheme.

In 2005, Waters presented a pairing-based signature scheme which does not assume the existence of random oracles. In 2007, Lu, Ostrovsky, Sahai, Shacham, and Waters presented the LOSSW aggregate signature scheme which does not assume the existence of random oracles.

The BLS, BGLS, Waters, and LOSSW authors each chose to work with a restricted class of pairings. In each scheme, it is clear that the scheme extend to arbitrary pairings. We present the schemes in their full generality, explore variations of the schemes, and discuss optimizations that can be made when using specific pairings.

Each of the schemes we discuss is secure assuming that the computational Diffie-Hellman (CDH) assumption holds. We improve on the security reduction for a variation of the BGLS signature scheme which allows for some restrictions of the BGLS signature scheme can be dropped and provides a stronger guarantee of security. We show that the BGLS scheme can be modified to reduce public-key size in presence of a certifying authority, when a certain type of pairing is used. We show that patient-free bit-compression can be applied to each of the scheme with a few modifications.

Acknowledgements

I would like to thank my supervisor Alfred Menezes and Sanjit Chatterjee for their help and support throughout my thesis work.

I wish to thank my readers David Jao and Edlyn Teske for their comments and suggestions.

Contents

CHAPTER 1

Introduction

1.1. Signature schemes

Two parties Alice and Bob wish to communicate over an unsecured channel. Alice wishes to send messages to Bob. They are not concerned with hiding their messages but they wish to ensure that messages have not been modified in-transit by some malicious adversary, Oscar.

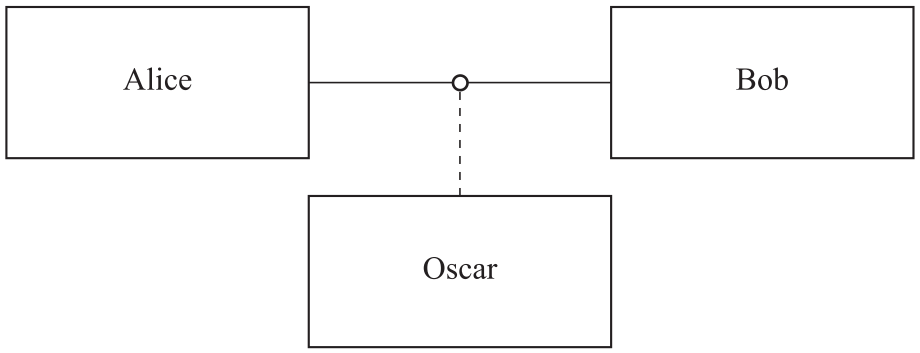

FIGURE 1. Alice sends a message to Bob. Oscar may modify the message before the message reaches Bob.

In the world of pen-and-paper, this problem is addressed using hand-written signatures. Alice signs a paper document with her signature, writes Bob's address on an envelope, seals the document inside, and drops it in a mailbox. If the document is intercepted at Bob's mailbox, it is a non-trivial task to place an identical signature on a different document. When Bob receives the paper document with Alice's signature, Bob has some assurance that the document is *authentic*.

Of course, in the digital world, paper documents and signatures are replaced with binary strings. The binary string representing the hand-written signature could easily be duplicated. *Signature schemes* are meant to address authentication in the digital world.

Public-key signature schemes require at least three algorithms, `Generate`, `Sign`, and `Verify`. The `Generate` algorithm produces private and public keys. The private key is used to produce signatures. The public key is used to verify a signature. Of course, the verifier must obtain an authentic copy of the public key. The `Sign` algorithm produces a *signature* on a given message using the private key. The `Verify` algorithm determines if a signature on some message is valid. Any signature produced by the `Sign` algorithm on a message m with the private key SK is accepted by the `Verify` algorithm on the message m with the public key PK corresponding to SK.

The most commonly used signature scheme is Full Domain Hash (FDH) [**3**]. Its security relies on the difficulty of the RSA problem and assumes the existence of a random oracle.

Full Domain Hash (FDH). *We define the functions* `Generate`*,* `Sign`*, and* `Verify`*. Fix some integer ℓ. Let $H\colon \{0,1\}^* \to \mathbb{Z}_N$ be a random oracle (where N is defined later).*

`Generate` *selects two primes p, q of bit-length ℓ uniformly at random and computes $N = pq$. The algorithm selects some $e \in \mathbb{Z}_N^*$ (usually $e = 3$ for fast verification) and computes d such that $ed \equiv 1 \pmod{\varphi(N)}$. The public key $\langle N, e\rangle$ and the private key d are returned. When we are referring to FDH, all multiplication using the keys $\langle N, e\rangle$, d are done modulo N.*

`Sign` *takes as input a message $m \in \{0,1\}^*$ and a private key d. The algorithm returns $\sigma = H(m)^d$.*

`Verify` *takes as input a message m, a public key $\langle N, e\rangle$, and a signature σ. The algorithm accepts the signature iff $H(m) = \sigma^e$.*

We will formalize the notion of security for signature schemes in Section 1.3. Intuitively, only the party that holds the private key should be able to create signatures. FDH has the property that it is infeasible to create signatures without the private key, provided that the hash function employed is assumed to be a random function and the following problem is computationally infeasible.

The *RSA problem* is the problem of determining h^d, given only e, N, and $h \in \mathbb{Z}_N$. The best method known for solving this problem (and the best attack known on FDH) is to factor N using the number field sieve algorithm [**22**] which runs in time

$$\mathrm{O}\left(e^{(1.923+\mathrm{o}(1))(\log n)^{1/3}(\log\log n)^{2/3}}\right).$$

Consequently, to obtain 80 bits of security, we require that N be a 1024 bit integer.

The assumption that the hash function is a random function is known as the *random oracle assumption*.

1.2. Bilinear pairings

Let G and G_T be groups of prime order p. Let g be a generator of G. A function $e\colon G \times G \to G_T$ is said to be a *non-degenerate, symmetric, bilinear pairing*, if e satisfies the following properties

(i) $e(g, g) \neq 1$,
(ii) $e(g^x, g) = e(g, g)^x = e(g, g^x)$, for all integers x.

In the remainder of this chapter, we fix e, G, and a generator g of G. In Chapter 2, we will fix more general entities for the remainder of the thesis.

The signature schemes we discuss assume that a problem known as the *computational Diffie-Hellman problem* (*CDH*) is difficult to solve for a sufficiently large group G.

The computational Diffie-Hellman problem (CDH). *Let G be a group of prime order with generator g. Given elements g^x, g^y, find g^{xy}.*

There exists a decisional version of the CDH problem, called the *decisional Diffie-Hellman problem (DDH)*: given g^x, g^y, and g^z, determine if

$$g^z = g^{xy}. \tag{1}$$

Using a pairing $e\colon G \times G \to G_T$, we have that Equation (1) holds iff

$$e(g^z, g) = e(g^x, g^y).$$

Therefore, the DDH problem can be solved using two pairing evaluations.

1.3. The Boneh-Lynn-Shacham signature scheme

We present a simplified version of the BLS signature scheme [**7**]. In Chapter 3, we generalize the BLS scheme for a more general class of pairings.

BLS. *Let $H\colon \{0,1\}^* \to G$ be a hash function. We define the algorithms* `Generate`*,* `Sign`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and returns the private key x and the public key $X = g^x$.*

`Sign` *takes as input a private key x, a message m, and returns a* signature *$\sigma = H(m)^x$.*

`Verify` *takes as input a public key X, a message m, and a signature σ. The algorithm accepts the signature iff*

$$e(\sigma, g) = e(H(m), X).$$

As we have mentioned, it should be impossible or at least computationally infeasible for an adversary to create signatures on messages without holding the private key. A strong security model is one in which the adversary is given many abilities and has a relatively weak goal. We fix some public key PK for the adversary and its goal is to produce a signature on any message of its choosing. We refer to such a signature as a *forgery*.

Intuitively, it is reasonable to assume that the adversary has seen many signatures of messages signed by a legitimate party. Of course, we wish the adversary to return the signature of a message that was not already signed. It is also conceivable that the adversary can force a legitimate party to sign as many "benign" messages as the adversary wishes. The adversary could then proceed to forge a signature on a potentially "damaging" message. We formalize these ideas in our attack model. We emphasize that the model we will describe has no notion of "benign" and "damaging" messages; the adversary is permitted to obtain signatures for arbitrary messages and to succeed must produce a signature for a message not queried.

Consider an arbitrary signature scheme. We fix an arbitrary user with some public key, referred to as the *challenge public key.* We assume that an adversary has access to the `Sign` algorithm as a black box. That is, the adversary is presented with an interface that behaves exactly like the `Sign` algorithm except that the private key need not be given as an input. We refer to this black box as a *signing oracle.* The goal of the adversary is to produce a signature for some message not given as input to the oracle. We refer to such an adversary as a *forger.* If no computationally-bounded adversary can succeed with a non-neglidgeable probability, the signature scheme is said to be *existentially unforgeable under chosen-message attacks.*

Assuming that the CDH problem in G is computationally infeasible and H is a random oracle, the BLS signature scheme is existentially unforgeable under chosen-message attacks [**7**].

1.4. Aggregate signature schemes

Given a set of N messages each corresponding to one of N signers, the signers wish to construct a signature which will convince a verifier that the messages are authentic. Using the signature schemes we previously described, we could generate N signatures and verify them one at a time. The total bit-length of the signatures grows linearly in the number of signers. Aggregate signature schemes are designed to fix the bit-length of the *aggregate signature*. That is, regardless of the number of signers and messages, the bit-length of the aggregate signature produced is fixed. Aggregate signatures schemes have applications in secure routing protocols and certificate chains [**5**].

1.4.1. The BGLS scheme. The first aggregate signature scheme, proposed by Boneh, Gentry, Lynn, and Shacham, is referred to as the BGLS aggregate signature scheme [**5**]. BGLS is inspired by the BLS signature scheme. Each user creates their own signature using the BLS algorithm and using a new algorithm we refer to as `Aggregate`, the signatures are combined into a single aggregate signature of a fixed bit-length. We will discuss a notion of security for aggregate signature schemes later in this section. For now, we present the BGLS signature scheme.

The Boneh-Gentry-Lynn-Shacham aggregate signature scheme (BGLS). *Let $H\colon \{0,1\}^* \to G$ be a hash function. We define the algorithms* `Generate`*,* `Sign`*,* `Aggregate`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and returns the private key x and the public key $X = g^x$.*

`Sign` *takes as input a private key x, a message m, and returns a* signature *$\sigma = H(m)^x$.*

`Aggregate` *takes as input signatures $\sigma_1, \ldots, \sigma_N$ and returns an* aggregate signature

$$\sigma = \prod_{i=1}^{N} \sigma_i$$

`Verify` *takes as input public keys $X_1, \ldots, X_N$, messages $m_1, \ldots, m_N$, and an aggregate signature σ. The algorithm accepts the signature iff*

$$e(\sigma, g) = \prod_{i=1}^{N} e(H(m_i), X_i).$$

Intuitively, given a particular user that contributes to an aggregate signature, the security of this user's contribution should not rely on the security of other users. That is, if we are given a set of $N-1$ compromised users and a single uncompromised user, it should be infeasible to generate an aggregate signature of these N users on any set of messages. To formalize this notion of security, we fix a particular user with some challenge public key. As with signature schemes, we grant an adversary access to a signing oracle with respect to this user's public key.

An adversary is successful if the adversary produces an aggregate signature on N messages corresponding to N public keys, one of which is the challenge public key. The adversary must produce all messages and all public keys corresponding to the signature. The message corresponding to the challenge public key cannot be one that was given as input to the signing oracle. We are careful to note that the adversary need only produce the public keys of the other users. The adversary, in general, does not need to possess the private keys of compromised users.

Since the invention of BGLS, other aggregate signature schemes have been proposed. In Chapter 6, we present one such aggregate signature scheme, called LOSSW [**24**]. The LOSSW aggregate signature scheme is a *sequential aggregate signature scheme* wherein each user must add their contribution to an *aggregate-so-far signature* in sequence. This differs from the BGLS signature scheme where individual signatures can be aggregated in sequence or at any time.

1.5. Outline and contributions

In Chapter 2, we define a more general class of pairings and define computational assumptions more general than the CDH assumption. Throughout this thesis, we present schemes in their most general form. Each scheme may be optimized for specific pairings.

The BLS signature scheme and the BGLS aggregate signature schemes are stated in a general setting in Chapters 3 and 4. Previous definitions and proofs of security of BLS and BGLS have been in terms of a specific pairing. Bellare, Namprempre, and Neven presented a variation of the BGLS scheme which offers a better proof of security [**2**]. We present an improvement to their proof of security. Although they claim to have proven their Theorem 3.2, the bounds they obtain in their probabilistic reduction do not correspond to those of the theorem. We give a direct proof of their Theorem 3.2. We show that if a proof-of-possession is implemented, then using certain pairings, BGLS public keys can be made smaller and verification can be done more efficiently. We describe a method for compressing points in BGLS that avoids certain patents related to bit-compression.

Waters presented a signature scheme using symmetric pairings which is secure without making the random oracle assumption [**32**]. In Chapter 5, we generalize the Waters signature scheme to arbitrary pairings. We show how the scheme can be modified to reduce signature size at the cost of dramatically increasing the public key size. We apply the same technique we used in BGLS to obtain patent-avoiding bit-compression. In Chapter 6, we present the LOSSW aggregate signature scheme and apply the same modifications we outlined for the Waters signature scheme.

In Chapter 7, we state results due to Coron [**10**] which show that security reductions for deterministic signature schemes cannot be improved without modification to the underlying scheme. We abstract some of the concepts used in Coron's proof and show that it can be extended to a class of non-deterministic signature schemes, which includes the Waters signature scheme.

CHAPTER 2

Baretto-Naehrig Curves

2.1. Terminology

An elliptic curve E defined over a field $\mathbb{F}$ of characteristic neither 2 nor 3 is defined by an equation

$$y^2 = x^3 + ax + b, \tag{2}$$

where a, $b \in \mathbb{F}$. The set of $\mathbb{F}$-rational points on E is

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + ax + b\} \cup \{\infty\}.$$

We denote by $\mathbb{F}_n$ the finite field of order n.

Let $p \neq 2$, 3 be prime and let E be an elliptic curve such that the order $n = \#E(\mathbb{F}_p)$ is prime. The *embedding degree* of E is the smallest integer k such that n divides $p^k - 1$. We shall assume throughout that $k > 1$. The *trace map* $\mathrm{Tr}\colon E(\mathbb{F}_{p^k}) \to \mathbb{E}(\mathbb{F}_p)$ of E is defined by

$$\mathrm{Tr}(Z) = \sum_{i=0}^{k-1} \pi^i(Z),$$

where $\pi(x, y) = (x^p, y^p)$ is the pth-power Frobenius map.

Let $E[n]$ be the set of points P in $E(\mathbb{F}_{p^k})$ such that the order of P divides n. We refer to $E[n]$ as the set of *n-torsion points*. It is well known that $E[n] \cong \mathbb{Z}_n \times \mathbb{Z}_n$, which implies that $E[n]$ has $n + 1$ subgroups of order n.

Consider groups G_1, G_2, G_T of prime order n and generators g_1, g_2 of G_1 and G_2 respectively. A function $e\colon G_1 \times G_2 \to G_T$ is referred to as a *non-degenerate bilinear pairing* if

(i) $e(g_1, g_2) \neq 1$;
(ii) $e(g_1^x, g_2) = e(g_1, g_2)^x = e(g_1, g_2^x)$, for all integers x.

The pairings we discuss in Chapter 1 are of *Type 1*, since $G_1 = G_2$. If $G_1 \neq G_2$ and an efficiently computable isomorphism $\psi\colon G_2 \to G_1$ is known, we refer to the pairing as a *Type 2* pairing. If $G_1 \neq G_2$ and no such map is known, the pairing is a *Type 3* pairing [**13**].

2.2. Baretto-Naehrig curves

A Baretto-Naehrig (BN) elliptic curve [**1**] is constructed by selecting an integer z such that $p = 36z^4 + 36z^3 + 24z^2 + 6z + 1$ and $n = 36z^4 + 36z^3 + 18z^2 + 6z + 1$ are prime. Then there is an elliptic curve $E\colon y^2 = x^3 + b$ defined over $\mathbb{F}_p$ such that $n = \#E(\mathbb{F}_p)$. Each such elliptic curve has embedding degree $k = 12$.

2.2.1. Type 3 pairings. Let $G_1 = E(\mathbb{F}_p$. Let G_T denote the unique order-n subgroup of $\mathbb{F}_{p^{12}}^*$. The curve E has a *degree*-6 *twist* over $\mathbb{F}_{p^2}$; that is, there exists an elliptic curve $\tilde{E}$ over $\mathbb{F}_{p^2}$ such that $d = 6$ is the smallest positive integer for which E and $\tilde{E}$ are isomorphic over $\mathbb{F}_{p^{2d}}$ [**18**]. The curve $\tilde{E}$ has the property that $n \mid \#\tilde{E}(\mathbb{F}_{p^2})$. Let $\tilde{T}$ be a point of order n and set $\tilde{G}_2 = \langle \tilde{T} \rangle$. Then there exists an efficiently computable monomorphism $\phi\colon \tilde{G}_2 \to E(\mathbb{F}_{p^{12}})$. For $T = \phi(\tilde{T})$, the group $G_2 = \langle T \rangle$ is such that $G_2 \neq G_1$ and we have an efficiently-computable isomorphism $\phi\colon \tilde{G}_2 \to G_2$. The group G_2 is called the *trace*-0 *subgroup* of $E[n]$, since it has the property that for each $Z \in G_2$, the trace of Z is the identity element. Next, we define three asymmetric pairings on (G_1, G_2, G_T). Since no efficiently-computable isomorphism from G_2 to G_1 is known, these pairings are of Type 3.

First, we describe the *Tate pairing*. The *Miller function* $f_{n,P}$ [**25**] is a function whose only zeroes and poles in E are a zero of order n at P and a pole of order n at ∞. The full Tate pairing $\hat{e}\colon E[n] \times E[n] \to G_T$ is defined for P, $Q \in E[n]$ and $R \in E(\mathbb{F}_{p^{12}})$ by

$$\hat{e}(P, Q) = \left(\frac{f_{n,P}(Q + R)}{f_{n,P}(R)} \right)^{(p^{12}-1)/n}.$$

If we restrict the domain of the Tate pairing to $G_1 \times G_2$, the Tate pairing $t_n\colon G_1 \times G_2 \to G_T$ is defined by

$$t_n(P, Q) = \big(f_{n,P}(Q)\big)^{(p^{12}-1)/n}$$

and can be efficiently computed using Algorithm 1 [**6**].

Algorithm 1 (Computing the Tate pairing)

```
INPUT: $P \in G_1$ and $Q \in G_2$.
OUTPUT: $t_n(P, Q)$.
    1. Write $n$ in binary: $n = \sum_{i=0}^{L-1} n_i 2^i$.
    2. $T \longleftarrow P$,   $f \longleftarrow 1$.
    3. For $i$ from $L - 2$ downto to 0 do:        {Miller operation}
       3.1 Let $\ell$ be the tangent line at $T$.
       3.2 $T \longleftarrow 2T$,   $f \longleftarrow f^2 \cdot \ell(Q)$.
       3.3 If $n_i = 1$ and $i \neq 0$ then
                 Let $\ell$ be the line through $T$ and $P$.
                 $T \longleftarrow T + P$,   $f \longleftarrow f \cdot \ell(Q)$.
    4. Return($f^{(p^{12}-1)/n}$).        {Final exponentiation}
```

The *ate pairing* $a_n\colon G_1 \times G_2 \to G_T$ [**18**] is defined by

$$a_n(P, Q) = \big(f_{t-1,Q}(P)\big)^{(p^{12}-1)/n}$$

where $t - 1 = p - n = 6z^2$. The ate pairing is generally faster to compute than the Tate pairing, since the number of iterations in the Miller operation is determined by the bit-length of $t - 1 \approx \sqrt{n}$.

The *R-ate pairing* $R_n\colon G_1 \times G_2 \to G_T$ [**23**] further reduces the number of iterations in the Miller operation. Set $a = 6z + 2$, $f = f_{a,Q}(P)$, and let $\ell_{A,B}$ denote the line through A and B. We define R_n by

$$R_n(P, Q) = \Big(f \cdot \big(f \cdot \ell_{aQ,Q}(P)\big)^p \cdot \ell_{\pi(aQ+Q),aQ}(P)\Big)^{(p^{12}-1)/n}.$$

There is an integer N such that $R_n(P, Q) = \hat{e}(Q, P)^N$ for all $P \in G_1$ and $Q \in G_2$ [**23**]. The R-ate pairing can be computed using Algorithm 2. The number of iterations in the Miller operation is now determined by the bit-length of $a \approx \sqrt{t} \approx n^{1/4}$.

Algorithm 2 (Computing the R-ate pairing)

```
INPUT: $P \in G_1$ and $Q \in G_2$.
OUTPUT: $R_n(Q, P)$.
    1. Write $a = 6z + 2$ in binary: $a = \sum_{i=0}^{L-1} a_i 2^i$.
    2. $T \longleftarrow Q$,    $f \longleftarrow 1$.
    3. For $i$ from $L - 2$ downto 0 do
       3.1 Let $\ell$ be the tangent line at $T$.
       3.2 $T \longleftarrow 2T$,    $f \longleftarrow f^2 \cdot \ell(P)$.
       3.3 If $a_i = 1$ then
                Let $\ell$ be the line through $T$ and $Q$.
                $T \longleftarrow T + Q$,    $f \longleftarrow f \cdot \ell(P)$.
    4. $f \longleftarrow f \cdot (f \cdot \ell_{T,Q}(P))^p \cdot \ell_{\pi(T+Q),T}(P)$.
    5. Return($f^{(p^{12}-1)/n}$).
```

Table 1, due to Hankerson, Menezes, and Scott [**16**], lists the expected costs of computing the Tate, ate, and R-ate pairings for a particular BN curve described in Section 2.3, demonstrating the superiority of the R-ate pairing. The cost estimates have been validated by experiments. For example, [**16**] reports timings of 81 million and 54 million clock cycles for computing the ate and R-ate pairings on a 2.8 GHz Pentium 4 machine using general purpose registers.

Pairing	Miller operation	Final exponentiation	Total
Tate	27,934m	7,246m+i	35,180m+i
ate	15,801m	7,246m+i	23,047m+i
R-ate	7,847m+i	7,246m+i	15,093m+2i

TABLE 1. Expected costs of the Tate, ate and R-ate pairings for the BN curves described in Section 2.3. Here, m and i denote multiplication and inversion in $\mathbb{F}_p$.

2.2.2. Type 2 pairings. Let $R \in E[n]$ where $R \notin G_1$ and $R \notin G_2$ and define $G_2' = \langle R \rangle$. The map $e_n\colon G_1 \times G_2' \to G_T$ defined by $e_n(P, Q) = \hat{e}(Q, P)^{2N}$ is an asymmetric pairing on (G_1, G_2', G_T). Since the trace map is an efficiently computable isomorphism from G_2' to G_1, the map e_n is a Type 2 pairing.

Lemma 1. [**19**] *Let $P \in G_1$ and $Q \in G_2'$. Then $e_n(P,Q) = R_n(P,\hat{Q})$, where $\hat{Q} = Q - \pi^6(Q)$.*

Proof. First note that $\hat{Q} \neq \infty$ since $Q \notin E(\mathbb{F}_{p^6})$. Moreover,

$$\mathrm{Tr}(\hat{Q}) = \mathrm{Tr}(Q) - \mathrm{Tr}(\pi^6(Q)) = \infty,$$

and hence $\hat{Q} \in G_2$. Finally,

$$\begin{aligned} e_n(P,Q) &= \hat{e}(Q,P)^{2N} \\ &= \hat{e}(2Q,P)^N \\ &= \hat{e}(Q+\hat{Q}+\pi^6(Q),P)^N \\ &= \hat{e}(\hat{Q},P)^N \cdot \hat{e}(Q+\pi^6(Q),P)^N \\ &= R_n(P,\hat{Q}), \end{aligned}$$

since $Q + \pi^6(Q) \in E(\mathbb{F}_{p^6})$ whence $\hat{e}(Q+\pi^6(Q)) = 1$ [**12**, Lemma IX.8]. $\square$

2.3. A particular BN curve

Consider the BN curve

$$E/\mathbb{F}_p : y^2 = x^3 + 3$$

with BN parameter $z = 6000000000001\mathrm{F2D}$ (in hexadecimal) [**11**]. For this choice of BN parameter, p is a 256-bit prime of Hamming weight 87, $n = \#E(\mathbb{F}_p)$ is a 256-bit prime of Hamming weight 91, and the R-ate parameter $a = 6z + 2$ is a 66-bit integer of Hamming weight 9. Note that $p \equiv 7 \pmod 8$ (whence -2 is a nonsquare modulo p) and $p \equiv 1 \pmod 6$.

2.3.1. Field representation. The extension field $\mathbb{F}_{p^{12}}$ is represented using tower extensions

$$\begin{aligned} \mathbb{F}_{p^2} &= \mathbb{F}_p[u]/(u^2+2), \\ \mathbb{F}_{p^6} &= \mathbb{F}_{p^2}[v]/(v^3-\xi) \text{ where } \xi = -u-1, \text{ and} \\ \mathbb{F}_{p^{12}} &= \mathbb{F}_{p^6}[w]/(w^2-v). \end{aligned}$$

We also have the representation

$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[W]/(W^6-\xi) \text{ where } W = w.$$

Hence an element $\alpha \in \mathbb{F}_{p^{12}}$ can be represented in any of the following three ways:

$$\begin{aligned} \alpha &= a_0 + a_1 w, \quad \text{where } a_0, a_1 \in \mathbb{F}_{p^{12}} \\ &= (a_{0,0} + a_{0,1}v + a_{0,2}v^2) + (a_{1,0} + a_{1,1}v + a_{1,2}v^2)w \quad \text{where } a_{i,j} \in \mathbb{F}_{p^2} \\ &= a_{0,0} + a_{1,0}W + a_{0,1}W^2 + a_{1,1}W^3 + a_{0,2}W^4 + a_{1,2}W^5. \end{aligned}$$

We let (m,s,i), $(\tilde{m},\tilde{s},\tilde{\imath})$, (M,S,I) denote the cost of multiplication, squaring, inversion in $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^{12}}$, respectively. Experimentally, we have $s \approx 0.9m$ and $i \approx 41m$ on a Pentium 4 processor [**16**]. In our cost estimates that follow, we will make the simplifying assumption $s \approx m$.

2.3.1.1. *Arithmetic in $\mathbb{F}_{p^2}$.* We have $\tilde{m} \approx 3m$ using Karatsuba's method which reduces a multiplication in a quadratic extension to 3 (rather than 4) small field multiplications; $\tilde{s} \approx 2m$ using the complex method:

$$(a+bu)^2 = (a-b)(a+2b) - ab + (2ab)u;$$

and $\tilde{i} \approx i + 2m + 2s$ since

$$(a+bu)^{-1} = \frac{a-bu}{a^2+2b^2}.$$

Note also that p-th powering is free in $\mathbb{F}_{p^{12}}$ because $(a+bu)^p = a - bu$.

2.3.1.2. *Arithmetic in $\mathbb{F}_{p^6}$.* Karatsuba's method reduces a multiplication in a cubic extension to 6 (rather than 9) multiplications in the smaller field. Hence a multiplication in $\mathbb{F}_{p^6}$ costs $18m$. Squaring in $\mathbb{F}_{p^6}$ costs $2\tilde{m} + 3\tilde{s} = 12m$ via the following formulae [**9**]: if

$$\beta = b_0 + b_1 v + b_2 v^2 \in \mathbb{F}_{p^6}$$

where $b_i \in \mathbb{F}_{p^2}$, then

$$\beta^2 = (A + D\xi) + (B + E\xi)v + (B + C + D - A - E)v^2$$

where $A = b_0^2$, $B = 2b_0b_1$, $C = (b_0 - b_1 + b_2)^2$, $D = 2b_1b_2$, and $E = b_2^2$. Finally, as shown in [**31**, Section 3.2], inversion in $\mathbb{F}_{p^6}$ can be reduced to 1 inversion, 9 multiplications, and 3 squarings in $\mathbb{F}_{p^2}$.

2.3.1.3. *Arithmetic in $\mathbb{F}_{p^{12}}$.* Since $\mathbb{F}_{p^{12}}$ is a tower of quadratic, cubic, and quadratic extensions, Karatsuba's method gives $M \approx 54m$. By using the complex method for squaring in $\mathbb{F}_{p^{12}}$ and Karatsuba for multiplication in $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^2}$, we have $S \approx 36m$. Since inversion in $\mathbb{F}_{p^{12}}$ can be reduced to 1 inversion, 2 multiplications, and 2 squarings in $\mathbb{F}_{p^2}$, it follows that $I \approx i + 97m$.

2.3.2. Elliptic curve operations. A point (X, Y, Z) in Jacobian coordinates corresponds to the point (x, y) is affine coordinates with $x = X/Z^2$ and $y = Y/Z^3$. The formulas for doubling a point in $E(\mathbb{F}_{p^d})$ represented in Jacobian coordinates require 3 multiplications and 4 squarings in $\mathbb{F}_{p^d}$, while the formulas for mixed Jacobian-affine addition in $E(\mathbb{F}_{p^d})$ require 8 multiplications and 3 squarings in $\mathbb{F}_{p^d}$.

2.3.3. Type 2 versus Type 3 pairings. Table 2 lists the bitlengths of elements in G_1, G_2, G_2' and G_T, and the estimated costs of performing essential operations in these groups.

2.3.3.1. *Representing elements in G_1, G_2 and G_2'.* A point $Q = (x, y) \in G_1$ can be represented in compressed form by $x \in \mathbb{F}_p$ plus a sign bit of $y \in \mathbb{F}_p$. The full y-coordinate can be recovered by solving $y^2 = x^3 + 3$ over $\mathbb{F}_p$ via

$$y = \pm\sqrt{x^3+3} = (x^3+3)^{(p+1)/4}.$$

The exponentiation can be performed using sliding windows of width 5, at a cost of $315m$.

A point $\tilde{Q} = (x, y) \in \tilde{E}(\mathbb{F}_{p^2})$ can be represented by $x \in \mathbb{F}_{p^2}$ plus a sign bit of $y \in \mathbb{F}_{p^2}$. The full y-coordinate $y = \pm\sqrt{x^3+3}$ can be recovered at a cost of 2 square roots in $\mathbb{F}_p$ plus $i + m + 2s$ using Scott's method for computing square roots in $\mathbb{F}_{p^2}$ [**31**]. The overall cost is $674m$.

	Type 2	Type 3
Bitlength of elements in G_1	257	257
Bitlength of elements in G_2'/G_2	6144	513
Bitlength of elements in G_T	3072	3072
Decompressing elements in G_1	$315m$	$315m$
Decompressing elements in G_2'/G_2	—	$674m$
Exponentiation in G_1	$2{,}345m$	$2{,}345m$
Exponentiation in G_2'/G_2	$105{,}462m$	$5{,}859m$
Fixed-base exponentiation in G_1	$718m$	$718m$
Fixed-base exponentiation in G_2'/G_2	$34{,}312m$	$1{,}906m$
e_N/R_N Pairing	$15{,}175m$	$15{,}175m$
Hashing into G_1	$315m$	$315m$
Hashing into G_2'/G_2	—	$6{,}533m$

TABLE 2. Bitlengths of elements in G_1, G_2, G_2' and G_T, and estimated costs (in terms of $\mathbb{F}_p$ multiplications) of basic operations.

Finally, a point $Q = (x, y) \in E(\mathbb{F}_{p^{12}})$ can be represented by $x \in \mathbb{F}_{p^{12}}$ and $y \in \mathbb{F}_{p^{12}}$. Point compression is not used because decompression would be prohibitively expensive due to the high cost of computing square roots in $\mathbb{F}_{p^{12}}$.

2.3.3.2. *Exponentiation in G_1, G_2 and G_2'.* Computing kP (also known as point multiplication), where k is an integer and P is an elliptic curve point can be performed using the 5-NAF method. The cost of the w-NAF method with ℓ-bit multipliers k is

$$1D + (2^{w-2} - 1)A + \frac{\ell}{w+1}A + \ell D.$$

where D is the cost of doubling an elliptic curve point and A is the cost of adding two elliptic curve points (see [**17**, Algorithm 3.36]). Thus, the costs of exponentiation in G_1, G_2 and G_2' are $2,345m$, $5,859m$ and $105,462m$, respectively.

If the point P is fixed or known in advance, then the operation kP can be significantly accelerated by precomputing some multiples of P. For example, the fixed-base comb method with two tables (see [**17**, Algorithm 3.45]) with windows of width w has expected cost approximately

$$\left(\frac{2^w - 1}{2^w} 2e - 1\right) A + (e-1)D,$$

where $d = \lceil \ell/w \rceil$ and $e = \lceil d/2 \rceil$. Taking $w = 5$ yields the expected costs $718m$, $1,906m$ and $34,312m$ for fixed-base exponentiation in G_1, G_2 and G_2', respectively.

2.3.3.3. *Hashing into G_1, G_2 and G_2'.* Hashing into G_1 can be defined by first using a standard hash function (such as SHA-2) to hash to an x-coordinate of $E(\mathbb{F}_p)$. A y-coordinate can then be computed as $\sqrt{x^3+3}$. The dominant cost is for computing the square root in $\mathbb{F}_p$, yielding our cost estimate of $315m$ for hashing into G_1.

Hashing into G_2 can be defined by first using a standard hash function to hash to an x-coordinate of $\tilde{E}(\mathbb{F}_{p^2})$, then computing a y-coordinate as $\sqrt{x^3+b} \in \mathbb{F}_{p^2}$, and finally multiplying the resulting point by $\#\tilde{E}(\mathbb{F}_{p^2})/n$ to obtain a point of order

n. Since square roots in $\mathbb{F}_{p^2}$ cost $674m$ and point multiplication in $\tilde{E}(\mathbb{F}_{p^2})$ costs $5,859m$, hashing into G_2 can be performed at a cost of $6,533m$.

As noted in [**13**], no efficient method is known for hashing into G_2'. Based on Table 2, it seems that Type 3 pairings are superior to Type 2 pairings in all respects except for the lack of an efficiently computable isomorphism $\phi\colon G_2 \to G_1$.

2.4. Hard problems

In Chapter 1, we mentioned the CDH problem and stated the BLS scheme in terms of a symmetric pairing. In the remainder of the thesis, we fix a pairing $e\colon G_1 \times G_2 \to G_T$ such that G_1, G_2, and G_T are of prime order p and we fix generators g_1, g_2 of G_1, G_2 respectively. We do not restrict ourselves to a Type 1, Type 2, or Type 3 pairing; we describe all schemes in their most general setting.

In future chapters, we prove our schemes secure relative to a problem which we call *the co-CDH′ problem*.

The co-CDH′ problem. *Given g_1^x, g_2^x, g_2^y, produce g_1^{xy}. An algorithm which solves the co-CDH′ problem in time at most t with probability at least ε is referred to as a* co-CDH′ $\langle t, \varepsilon \rangle$-solver.

The CDH problem is a special instance of the co-CDH′ problem, when $G_1 = G_2$ and $g_1 = g_2$. If we are given an efficiently computable isomorphism $\psi\colon G_2 \to G_1$ such that $\psi(g_2) = g_1$, then given g_2^x, g_2^y, the problem is to compute g_1^{xy} and is referred to as *the co-CDH problem*. Since g_1^x can be computed using ψ, we see that the the co-CDH problem is a special case of the co-CDH′ problem. In the schemes we present, we do not assume the existence of the map ψ. If we are given such a map, parts of the protocol can be modified to reduce public-key size, signature size, and remove certain checks.

CHAPTER 3

The Boneh-Lynn-Shacham Signature Scheme

3.1. The basic scheme

The Boneh-Lynn-Shacham signature scheme (BLS) was first proposed in [**7**]. The authors assume the existence of an efficiently computable isomorphism $\psi\colon G_2 \to G_1$ such that $\psi(g_2) = g_1$. Although this map is not used in the actual scheme, the authors point out that the map is required for the security reduction. With a few modifications to the underlying hard problem, the requirement for the map ψ can be dropped.

The Boneh-Lynn-Shacham signature scheme. *Let $H\colon \{0,1\}^* \to G_1$ be a hash function. We define the algorithms* `Generate`*,* `Sign`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and returns the private key x and the public key $X = g_2^x$.*

`Sign` *takes as input a private key x, a message m, and returns a* signature *$\sigma = H(m)^x$.*

`Verify` *takes as input a public key X, a message m, and a signature σ. The algorithm accepts the signature iff*

$$e(\sigma, g_2) = e(H(m), X).$$

Consider an algorithm which can forge BLS signatures and is permitted to make q_S queries to a signing oracle and q_H queries to the hash function. If the algorithm succeeds with probability at least ε, in time at most t, the algorithm is said to be a *BLS $\langle q_H, q_S, t, \varepsilon\rangle$-forger.*

In the next section, we prove that the BLS scheme is existentially unforgeable under chosen-message attacks, assuming the intractability of the co-CDH′ problem.

3.1.1. Security.

It is convenient to abstract the problem of forging BLS signatures. Next, we state an interactive version of the co-CDH′ problem which is equivalent to forging BLS signatures. The problem was first stated by Galindo [**14**] and was inspired by the RSA1 problem [**21**]. We prove the equivalence to forging BLS signatures in Lemma 2.

The interactive co-CDH′ problem (co-CDH′1). *Let q_H be a positive integer. Given*

(i) *an element g_2^x of G_2;*
(ii) *elements $h_1, \ldots, h_{q_H}$ selected uniformly at random from G_1;*
(iii) *an oracle, which on input h_i, returns h_i^x,*

determine h_k^x, for some h_k not given as input to the oracle.

An algorithm which solves co-CDH$'$1 in time at most t, with probability at least ε, using at most q_S oracle queries, is said to $\langle q_H, q_S, t, \varepsilon\rangle$*-solve* the co-CDH$'$1 problem.

We prove the tight equivalence of forging BLS signatures and solving the co-CDH$'$1 problem in the next lemma.

Lemma 2. *Given a $\langle q_H, q_S, t, \varepsilon\rangle$-forger of the BLS scheme, we can construct an algorithm which $\langle q_H, q_S, t + q_H + q_S, \varepsilon\rangle$-solves the co-CDH$'$1 problem.*

Conversely, given an algorithm which $\langle q_H, q_S, t, \varepsilon\rangle$-solves the co-CDH$'$1 problem, we can construct a BLS $\langle q_H, q_S, t + q_H + q_S, \varepsilon\rangle$-forger.

Proof. In the *forward* direction, we begin with an instance of the co-CDH$'$1 problem. We construct an instance of BLS, using the element $X = g_2^x$ as the public key. We must respond to hash function queries and a signing oracle queries. The BLS forger queries the hash function H with distinct messages $m_1, \ldots, m_{q_H}$. To each query, we respond with $H(m_i) = h_i$. We assume that before making a signing query, the forger makes the hash query on the message to be signed. Informally, a signing query *uses up* one of the q_H hash queries as well as a signing query. To respond to a signing query on message m_i, we use the oracle given to us in the co-CDH$'$1 problem to produce $h_i^x = H(m_i)^x$ which we return as the signature of the message m_i. With probability ε, the BLS forger returns $H(m_k)^x$, for some m_k not queried to the signing oracle. We return $H(m_k)^x = h_k^x$ as our solution to the co-CDH$'$1 problem.

Conversely, we are given the public key $X = g_2^x$ of an instance of BLS. We select distinct messages $m_1, \ldots, m_{q_H}$ and set $h_1 = H(m_1), \ldots, h_{q_H} = H(m_{q_H})$. Since H is a random oracle, our choice of messages is irrelevant so long as they are distinct; we may chose them randomly or even sequentially starting from the zero bit-string. Since H is a random oracle, the h_i's are uniformly random elements of G_1. We give $X = g_2^x$ and the h_i's as input to the co-CDH$'$1 solver. To respond to an oracle query on h_i, we respond by using a BLS signing query on the message m_i. We obtain $H(m_i)^x = h_i^x$, our response to the oracle query. With probability ε, the algorithm returns h_k^x for some h_k not queried. The element $h_k^x = H(m_k)^x$ is a forgery of the message m_k. □

Having proven the equivalence of forging BLS signatures and the co-CDH$'$1 problem, we give an informal argument for the security of BLS. An algorithm which solves the co-CDH$'$1 problem is given g_2^x in G_2 and elements $h_1, \ldots, h_{q_H}$ selected uniformly at random from G_1. Solving co-CDH$'$1 without using the oracle is equivalent to solving co-CDH$'$. The oracle generates pairs of the form $\langle g_1^a, g_1^{xa}\rangle$ where $a \in \mathbb{Z}_p$ is selected uniformly at random. This is something we may do without the oracle. Intuitively, the oracle gives us no useful information and forging BLS signatures seems as difficult as solving co-CDH$'$.

In the next lemma, we show that we may build a co-CDH$'$ solver from a BLS forger.

Lemma 3. *Given a BLS $\langle q_H, q_S, t, \varepsilon\rangle$-forger, we may construct an algorithm which $\langle t + t', \varepsilon\varepsilon'\rangle$-solves the co-CDH′ problem with one query of the forger where*

$$t' = q_H + q_S,$$
$$\varepsilon' = \frac{1}{e(q_S + 1)},$$

and e is the constant 2.182 . . . .

Proof. We are given a BLS $\langle q_H, q_S, t, \varepsilon\rangle$-forger. By Lemma 2, we can construct an algorithm which $\langle q_H, q_S, t+q_H+q_S, \varepsilon\rangle$-solves the co-CDH′1 problem. We assume that we have such an algorithm.

Fix an instance of the co-CDH′ problem. We have elements g_1^x, g_1^y, g_2^x and our goal is to produce g_1^{xy}. We need to construct an instance of the co-CDH′1 problem and respond to oracle queries.

For $i = 1, \ldots, q_H$, we select an integer $a_i \in \mathbb{Z}_p$ uniformly at random. With probability $1/(q_S + 1)$, we set $h_i = g_1^y \cdot g_1^{a_i}$ and with probability $1 - 1/(q_S + 1)$, we set $h_i = g_1^{a_i}$. We give g_2^x and the h_i's as input to the co-CDH′1 solver.

To respond to an oracle query on input h_i, we abort if $h_i = g_1^y \cdot g_1^{a_i}$. Otherwise, $h_i = g_1^{a_i}$. Since we know a_i and g_1^x, we respond with $(g_1^x)^{a_i} = h_i^x$.

With probability at least ε, the algorithm will produce h_k^x, for some h_k not given as input to the oracle. If $h_k = g_1^{a_k}$, then we abort. Otherwise, we have that $h_k = g_1^y \cdot g_1^{a_k}$. Since we know a_k and g_1^x, we may compute

$$h_k^x \cdot (g_1^x)^{-a_k} = g_1^{xy}.$$

Next, we analyze the probability of success. With probability

$$\left(1 - \frac{1}{q_S + 1}\right)^{q_S},$$

we are able to respond to all signing queries. With probability $1/(q_S + 1)$, we have $h_k = g_1^y \cdot g_1^{a_k}$. Finally, the algorithm succeeds with probability at least ε. Putting this all together, we obtain an algorithm which solves the co-CDH′ problem with probability at least

$$\varepsilon\left(1 - \frac{1}{q_S + 1}\right)^{q_S} \frac{1}{q_S + 1} \geq \frac{\varepsilon}{e(q_S + 1)}.$$

□

We remark that in absence of an efficiently computable isomorphism $\psi \colon G_2 \to G_1$, there is no obvious way to use a co-CDH′ solver to forge BLS signatures. Counter to intuition, this suggests that in some sense, we should be more confident in the security of BLS than we are in the intractability of the co-CDH′ problem.

Lemma 3 tells us that if co-CDH′ is difficult, then forging BLS signatures is difficult. However, given a BLS forger which takes time 2^{80} (with $\varepsilon = 1/2$) and makes $q_S = 2^{30}$ signature queries, by Lemma 3 we can only conclude that there exists a co-CDH′ solver which takes time $2^{80} \cdot e2^{30} \approx 2^{110}$ (with $\varepsilon' = 1/2$). Because of this q_S gap in the reduction, we need G_1 to have 110 bits of security to ensure BLS has 80 bits of security via Lemma 3.

If we select G_1 with only 80 bits of security, the q_S gap is not known to provide an attacker with an advantage. That is, the best-known method for forging signature is to determine the private key using Pollard-Rho. Since the problem of forging BLS signature is not as well studied as the co-CDH′ problem, removing the q_S in the reduction's probability is desirable. Unfortunately, the following lemma shows that we cannot make a non-negligible improvement in Lemma 3.

Lemma 4. *Let $\mathcal{R}$ be a reduction which $\langle t + t', \varepsilon\varepsilon' \rangle$-solves co-CDH′, making one query to an algorithm which $\langle q_H, q_S, t, \varepsilon \rangle$-solves co-CDH′1. Let*

$$\varepsilon' = \frac{1}{e(q_S + 1)} + \delta$$

for some $\delta \in [0, 1]$. Then there exists an algorithm which can $\langle 2t', \delta \rangle$-solve the co-CDH′ problem.

Proof. This result is proven in Chapter 7. □

If δ is some non-negligible probability and $\mathcal{R}$ is an efficient reduction, then we obtain an efficient co-CDH′ solver. Since it is conjectured that co-CDH′ is intractable, the reduction in Lemma 3 is essentially optimal.

3.2. The Katz-Wang bit

The signatures in the BLS scheme are deterministic; for a fixed key, each message determines a unique signature. We present a variation of the signature scheme due to Galindo [**14**]. Galindo's technique was a variation of the technique used by Katz and Wang [**20**] to obtain a tight reduction of the RSA problem to a variant of RSA-FDH.

Non-deterministic BLS (NBLS). *Let $H\colon \{0,1\}^* \to G_1$ be a hash function. We define the algorithms* `Generate`*,* `Sign`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and returns a public key $X = g_2^x$ and a private key x.*

`Sign` *takes as input a message m and a private key x. The algorithm selects a bit $b \in \{0, 1\}$ uniformly at random and returns the signature $\sigma = \langle H(b, m)^x, b \rangle$. The bit b should be fixed for a given message. As suggested by Galindo* [**14**]*, we can hash the message along with a secret seed to obtain b.*

`Verify` *takes as input a message m, a public key X, and a signature $\sigma = \langle \sigma_1, \sigma_2 \rangle$. Just as in the BLS scheme, the algorithm accepts the signature iff*

$$e(\sigma_1, g_2) = e(H(\sigma_2, m), X).$$

The NBLS scheme enjoys the strongest notion of security for non-deterministic signatures captured in the following definition.

Definition 1. Consider a signature scheme which produces non-deterministic signatures and an adversary which is permitted q_H hash queries and q_S signature queries to some signing oracle. The signature scheme is said to be *strongly unforgeable under chosen-message attacks*, if the adversary cannot produce a signature σ on some message m in time at most t with probability at least ε such that σ was not returned by a signing oracle query on input m. If there exists such an adversary, we refer to it as a *strong $\langle q_H, q_S, t, \varepsilon \rangle$-forger* of the signature scheme, or simply a *strong forger* when the signature scheme and the parameters are understood.

An attack on the NBLS scheme can be modeled as solving the following interactive problem.

The co-CDH′2 problem. *Let q_H be a positive integer. Given*

(i) *an element g_2^x of G_2;*

(ii) *pairs of elements $\langle h_{0,1}, h_{1,1}\rangle, \ldots, \langle h_{0,q_H}, h_{1,q_H}\rangle$ selected uniformly at random from $G_1 \times G_1$;*

(iii) *an oracle, which on input $\langle h_{0,i}, h_{1,i}\rangle$, returns either $(h_{0,i})^x$ or $(h_{1,i})^x$ selected uniformly at random for each pair and responds deterministically for a particular pair,*

determine $(h_{b,k})^x$, where $(h_{b,k})^x$ was not returned as output by the oracle.

The equivalence of the co-CDH′2 problem with breaking the NBLS scheme is captured in the following lemma.

Lemma 5. *Given an NBLS strong $\langle q_H, q_S, t, \varepsilon\rangle$-forger, we can construct an algorithm which $\langle q_H, q_S, t, \varepsilon\rangle$-solves the co-CDH′2 problem.*

Conversely, given an algorithm which $\langle q_H, q_S, t, \varepsilon\rangle$-solves the co-CDH′2 problem, we can construct an NBLS strong $\langle q_H, q_S, t, \varepsilon\rangle$-forger.

We omit the proof of Lemma 5, since it is very similar to the proof of Lemma 2.

In the next lemma, we show that an NBLS strong forger can be used to construct an algorithm which can solve the co-CDH′ problem.

Lemma 6. *Given an NBLS strong $\langle q_H, q_S, t, \varepsilon\rangle$-forger, we can construct an algorithm which $\langle t + q_H + q_S, \varepsilon/2\rangle$-solves the co-CDH′ problem.*

Proof. By Lemma 5, a strong forger of the NBLS scheme can be used to construct a co-CDH′2 solver. We will show that the co-CDH′2 solver can be used to construct a co-CDH′ solver.

We begin with an instance of the co-CDH′ problem. We have elements g_1^x, g_1^y, g_2^x, and wish to produce g_1^{xy}. For each $i = 1, \ldots, q_H$, we select a bit $b_i \in \{0,1\}$ uniformly at random and we select integers c_i, $d_i \in \mathbb{Z}_p$ uniformly at random. Set $h_{b_i,i} = g_1^{c_i}$ and $h_{1-b_i,i} = g_1^{d_i} \cdot g_1^y$. We give g_2^x and the $h_{i,j}$'s as input to the co-CDH′2 solver.

To respond to an oracle query on input $\langle h_{0,i}, h_{1,i}\rangle$, we have that $h_{b_i,i} = g_1^{c_i}$. Since we know c_i and g_1^x, we return $(g_1^x)^{c_i} = (h_{b_i,i})^x$.

With probability at least ε, in time at most t, the algorithm returns $h_{b,k}^x$ for some $h_{b,k}^x$ not returned as output by the oracle. With probability at least 1/2, we have that $b = 1 - b_k$ and so $h_{1-b_k,k} = g_1^{d_k} \cdot g_1^y$. Since we know d_k and g_1^x, we may compute

$$(h_{b,k})^x \cdot (g_1^x)^{-d_k} = g_1^{xy}.$$

□

Comparing the results of Lemma 6 with Lemma 3, we see that with only a slight modification of BLS, we obtain a tight reduction. The importance of tight reductions in the BLS scheme is discussed in Section 3.4.

3.3. Point compression

The points on an elliptic curve E defined over a finite field $\mathbb{F}$ are elements $(x, y) \in \mathbb{F} \times \mathbb{F}$. For points of order greater than 2, the x-coordinate determines two distinct points on the curve. These two points are mutual inverses in the group $E(\mathbb{F})$. We can determine a point on the curve using the x-coordinate of the point and a single extra bit to determine one of the two points corresponding to the x-coordinate. This essentially reduces key and signature sizes by a half.

In order to avoid some patents on point compression, some authors have suggested dropping the y-coordinate entirely and modifying the protocols to handle the resulting ambiguity [**28**]. We define the map $\widetilde{\cdot}\colon E - \{\infty\} \to \mathbb{F}$ such that for $Q = (x, y)$, we have $\widetilde{Q} = x$. Given an x-coordinate $\widetilde{Q}$, we may determine Q up to a sign.

For the remainder of the section, we fix elliptic curve groups G_1, G_2 with generators P_1, P_2. Following convention, we use additive notation. We use the notation $\widetilde{\cdot}$ for both groups G_1 and G_2.

We present the BLS scheme with this modification. The compression modification we describe can be applied to all other variations of BLS we have mentioned.

Compression BLS (CBLS). *Let $H\colon \{0,1\}^* \to G_1$ be a hash function. We define, the algorithms* `Generate`*,* `Sign`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes $X = xP_2$. The algorithm returns the public key $\widetilde{X}$ and a private key x.*

`Sign` *takes as input a message m and a private key x. The algorithm computes $Q = xH(m)$ and returns the signature $\sigma = \widetilde{Q}$.*

`Verify` *takes as input a message m, a public key $\widetilde{X}$, and a signature σ. The algorithm computes elements Q and R such that $\widetilde{Q} = \sigma$ and $\widetilde{R} = X$. The signature is accepted iff either of the following two equations hold:*

$$e(Q, P_2) = e(H(m), R);$$
$$e(Q, P_2) = e(H(m), R)^{-1}.$$

Even with the added ambiguity, we still obtain the same security as BLS.

Lemma 7. *If there exists a $\langle q_H, q_S, t, \varepsilon\rangle$-CBLS forger, then we can construct a $\langle q_H, q_S, t + q_H + q_S, \varepsilon\rangle$-BLS forger.*

Proof. Fix a public key X of some BLS instance. We run the CBLS forger with public key $\widetilde{X}$.

To respond to signing queries, we use the BLS signing oracle to obtain a signature $\sigma \in E$ and respond with $\widetilde{\sigma} \in \mathbb{F}$.

Eventually, the forger will produce a signature $\sigma \in \mathbb{F}$. We compute Q such that $\widetilde{Q} = \sigma$. If

$$e(Q, P_2) = e(H(m), X),$$

then we return Q as a BLS forgery. Otherwise, we have

$$e(-Q, P_2) = e(H(m), X)$$

and we return $-Q$ as a BLS forgery. □

3.4. Practical considerations

The security of BLS depends on the type of pairing used and the size of the groups G_1, G_2, and G_T. Care needs to be taken in selecting a pairing for which the co-CDH′ problem is hard. To select a group, we consider the best attack known on the co-CDH′ problem. It is conjectured that the best attack is to solve the discrete log problem in G_1, G_2, or G_T. If the pairing is selected appropriately, the best method known of solving co-CDH′ is to determine the private key from the public key using the Pollard-Rho algorithm [**29**], which runs in time $\mathrm{O}\left(\sqrt{p}\right)$.

If we desire 80 bits of security against co-CDH′ solvers, we need to choose $|G_1| = 2^{160}$. Lemma 3 tells us that a forger of the BLS scheme can be used so solve co-CDH′. In practice, we can bound the number of signature queries by 2^{30} queries. If we take the proof of Lemma 3 seriously, then a deterministic BLS forger which runs in time 2^{80} yields a CDH forger which runs in time $\approx 2^{80}$ and succeeds with probability 2^{-30}. The CDH solver we obtain from the reduction is much weaker than the Pollard-Rho method we have protected against. We need to select a group size which takes into account Lemma 3, say $|G_1| = 2^{220}$.

There is no attack known on co-CDH′1 other than to solve co-CDH′ itself. We know by Lemma 4, that if co-CDH′ is intractable, then no tight reduction from co-CDH′ to co-CDH′1 will ever be found. However, this is not known to give any advantage to an adversary. Intuitively, it seems unlikely that such an adversary exists but it should be noted that co-CDH′1 has not been as well studied as co-CDH′. Choosing our keys without consideration of Lemma 4 and Lemma 3 might be too bold an assumption. We must either choose larger keys or assume that the co-CDH′-to-co-CDH′1 gap gives an attacker no advantage. If we are concerned enough to choose larger keys, using NBLS as opposed to BLS will allow us a tight reduction between forging BLS signature and solving co-CDH′ at the expense of only a 1-bit increase in signature size.

CHAPTER 4

The Boneh-Gentry-Lynn-Shacham Aggregate Signature Scheme

4.1. The scheme

Motivated by applications to secure routing protocols and certificate chains, Boneh et al. proposed the first aggregate signature scheme [**5**]. The scheme has key generation and signing algorithms which are similar to those of BLS.

The Boneh-Gentry-Lynn-Shacham aggregate signature scheme (BGLS). *Let $H\colon \{0,1\}^* \to G_1$ be a hash function. We define the algorithms* `Generate`*,* `Sign`*,* `Aggregate`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes $X = g_1^x$, $Y = g_2^x$. The algorithm returns the private key x and the public key $\langle X, Y\rangle$.*

`Sign` *takes as input a private key x, a message m, and returns the signature $\sigma = H(m)^x$.*

`Aggregate` *takes as input signatures σ_1, …, σ_N and returns the signature*

$$\sigma = \prod_{i=1}^{N} \sigma_i.$$

`Verify` *takes as input public keys $\langle X_1, Y_1\rangle$, …, $\langle X_N, Y_N\rangle$, messages m_1, …, m_N, and a signature σ. For each key $\langle X_i, Y_i\rangle$, the algorithm verifies that the key was correctly constructed by checking*

$$e(X_i, g_2) = e(g_1, Y_i). \tag{3}$$

This key verification step needs to be done only once per public key, across all invocations of the `Verify` *algorithm. To verify the actual signature, the algorithm checks*

$$e(\sigma, g_2) = \prod_{i=1}^{N} e(H(m_i), Y_i). \tag{4}$$

The algorithm accepts the signature iff the messages m_1, …, m_N are distinct and equations (3) *and* (4) *hold.*

We remark that verifying Equation (3) is required to guarantee security and the component X_i of the public key is used nowhere else.

If we are given a Type 2 pairing, then an efficiently computable isomorphism $\psi\colon G_2 \to G_1$ is known and we may compute X_i from Y_i. This shortens the length of the public key and we need not perform the check of Equation (3).

Consider an adversary which is given a single public key, referred to as the *challenge public key*. The goal of the adversary is to produce an aggregate signature on some set of public keys including the one given. The adversary may choose all other public keys and all messages. An aggregate signature scheme is said to be *existentially unforgeable under chosen-message attacks*, if given a challenge public key, it is infeasible for an adversary, who is given access to a signing oracle on the challenge public key, to produce a forgery of some set of messages containing a message corresponding to the challenge public key which was not given as input to the signing oracle. An algorithm which produces a forgery on N messages, with probability at least ε, in time at most t, making at most q_H hash queries and q_S signing queries is referred to as a BGLS $\langle q_H, q_S, N, t, \varepsilon\rangle$-forger.

In the next lemma, we prove that a BGLS forger can be used to solve the co-CDH′ problem.

Lemma 8. *Given a BGLS $\langle q_H, q_S, N, t, \varepsilon\rangle$-forger we can construct an algorithm which $\langle t+t', \varepsilon\cdot\varepsilon'\rangle$-solves co-CDH′, making one query to the BGLS forger where*

$$t' = q_H + q_S + N,$$
$$\varepsilon' = \frac{1}{e(q_S+N)}.$$

PROOF. We fix an instance of the co-CDH′ problem. We are given elements g_1^x, g_1^y in G_1 and g_2^x in G_2. We wish to produce g_1^{xy}.

We construct an instance of BGLS with public key $\langle X_1, Y_1\rangle = \langle g_1^x, g_2^x\rangle$ and run the forger. We need to respond to hash queries and signature queries.

To respond to hash query on message m, we select an integer $a_m \in \mathbb{Z}_p$ uniformly at random. With probability $1/(q_S+N)$, we respond with $g_1^y \cdot g_1^{a_m}$ and with probability $(q_S+N-1)/(q_S+N)$, we respond with $g_1^{a_m}$.

To respond to a signature query on message m, we abort if $H(m) = g_1^y \cdot g_1^{a_m}$. Otherwise, since we know a_m and g_1^x, we respond with $(g_1^x)^{a_m} = H(m)^x$.

Eventually, the forger returns the signature σ on some messages $m_1, \ldots, m_N$ and public keys $\langle X_1, Y_1\rangle, \ldots, \langle X_N, Y_N\rangle$ such that no signing query was made for m_1. If either $H(m_1) = g_1^{a_{m_1}}$ or for some $i > 1$, we have that $H(m_i) = g_1^y \cdot g_1^{a_{m_i}}$, then we abort. Otherwise, letting for each i, the integer x_i be the private key corresponding to the public key $\langle X_i, Y_i\rangle = \langle g_1^{x_i}, g_2^{x_i}\rangle$, we have that

$$\sigma = \prod_{i=1}^{N} H(m_i)^{x_i} = g_1^{x_1 y}\cdot g_1^{x_1 a_{m_1}} \cdot \prod_{i=2}^{N} g_1^{x_i a_{m_i}} = g_1^{xy}\cdot \prod_{i=1}^{N} g_1^{x_i a_{m_i}}$$

and may compute

$$\sigma \cdot \prod_{i=1}^{N} X_i^{-a_{m_i}} = g_1^{xy}.$$

Now, we determine the probability of success. With probability

$$\left(1 - \frac{1}{q_S+N}\right)^{q_S},$$

we are able to answer all signature queries. With probability

$$\left(1 - \frac{1}{q_S+N}\right)^{N-1}\frac{1}{q_S+N},$$

the signature returned by the forger is of the desired form.

Putting this all together, we obtain a co-CDH$'$ solver which succeeds with probability at least

$$\left(1 - \frac{1}{q_S + N}\right)^{q_S+N-1} \frac{1}{q_S + N} \geq \frac{1}{e(q_S + N)}.$$

□

4.1.1. The necessity of distinct messages. The authors of the BGLS scheme present an attack that succeeds if the requirement for distinct messages is dropped [**5**]. They mention a modification that allows this restriction to be dropped. The modification is discussed more in the next section. We present the attack.

Given a public key $\langle X_1, Y_1\rangle$, we select an integer $a \in \mathbb{Z}_p$ uniformly at random and compute $X_2 = g_1^a \cdot X_1^{-1}$, $Y_2 = g_2^a \cdot Y_1^{-1}$. For an arbitrary message $m_1 = m_2$, we compute $\sigma = H(m_1)^a$. The signature σ verifies on the public keys $\langle X_1, Y_1\rangle$, $\langle X_2, Y_2\rangle$ and messages m_1, m_2.

4.2. BNN modifications

In the paper which first describes BGLS [**5**], Boneh, et al. point out that by prepending the public key to each message, the requirement for distinct messages can be dropped. They do not give any details of the modified scheme.

If the signing algorithm is given messages $m_1, \ldots, m_N$ and public keys $\langle X_1, Y_1\rangle, \ldots, \langle X_N, Y_N\rangle$, for each $i = 1, \ldots, N$, the signing algorithm constructs a message $\hat{m}_i = (X_i, Y_i, m_i)$. To sign the message m_i with the public key $\langle X_i, Y_i\rangle = \langle g_1^{x_i}, g_2^{x_i}\rangle$, the signing algorithm computes $H(\hat{m}_i)^{x_i}$. It follows from the security of BGLS that this scheme is secure if the $\hat{m}_i$'s are distinct.

Bellare-Namprempre-Neven analyze the enhanced scheme in detail and show that the requirement for the $\hat{m}_i$'s to be distinct can be dropped [**2**].

BNN-BGLS. *Let $H\colon \{0,1\}^* \to G_1$ be a hash function.*

We define the algorithms `Generate`, `Sign`, `Aggregate`, *and* `Verify`.

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes $X = g_1^x$, $Y = g_2^x$. The algorithm returns the private key x and the public key $\langle X, Y\rangle$.*

`Sign` *takes as input a private key x, a message m, and returns a signature $\sigma = H(g_1^x, g_2^x, m)^x$.*

`Aggregate` *takes as input signatures $\sigma_1, \ldots, \sigma_N$ and returns the signature*

$$\sigma = \prod_{i=1}^{N} \sigma_i.$$

`Verify` *takes as input public keys $\langle X_1, Y_1\rangle, \ldots, \langle X_N, Y_N\rangle$, messages $m_1, \ldots, m_N$, and a signature σ. For each key $\langle X_i, Y_i\rangle$, the algorithm verifies that the key was correctly constructed by checking*

$$e(X_i, g_2) = e(g_1, Y_i). \tag{5}$$

This key verification step needs to be done only once per public key, across all invocations of the `Verify` *algorithm. To verify the actual signature, the algorithm* *checks*

$$e(\sigma, g_2) = \prod_{i=1}^{N} e(H(X_i, Y_i, m_i), Y_i). \tag{6}$$

The algorithm accepts the signature iff equations (5) *and* (6) *hold.*

In addition to the lifting of the distinct message restriction we obtain a tighter reduction to the co-CDH′ problem.

Lemma 9. *Given a BNN-BGLS $\langle q_H, q_S, N, t, \varepsilon\rangle$-forger, we can construct a co-CDH′ $\langle t + t', \varepsilon \cdot \varepsilon'\rangle$-solver which makes one query to the BNN-BGLS forger, where*

$$t' \le q_H + q_S,$$
$$\varepsilon' \ge \frac{1}{e(q_S + 1)}.$$

Proof. First, we fix an instance of the co-CDH′ problem. We are given elements g_1^x, g_1^y, g_2^x and our goal is to produce g_1^{xy}. We create an instance of the BNN-BGLS signature scheme with the public key $\langle X_1, Y_1\rangle = \langle g_1^x, g_2^x\rangle$ and run the BNN-BGLS forger. We need to respond to hash queries and signature queries.

To respond to a hash query on input (X, Y, m), if $\langle X, Y\rangle \ne \langle X_1, Y_1\rangle$, then we select an integer $a_{(X,Y,m)} \in \mathbb{Z}_p$ uniformly at random and return $H(X, Y, m) = (g_1)^{a_{(X,Y,m)}}$. Otherwise, $X = X_1$ and $Y = Y_1$. We select an integer $a_{(X,Y,m)} \in \mathbb{Z}_p$ uniformly at random and with probability $1/(q_S + 1)$, we return $H(X, Y, m) = g_1^y \cdot (g_1)^{a_{(X,Y,m)}}$; with probability $q_S/(q_S + 1)$, we return $H(X, Y, m) = (g_1)^{a_{(X,Y,m)}}$.

To respond to a signature query on some message m, we abort if $H(X_1, Y_1, m) = g_1^y \cdot g_1^{a_{(X,Y,m)}}$. Otherwise, we have that $H(X, Y, m) = g_1^{a_{(X,Y,m)}}$ and we return $(g_1^x)^{a_{(X,Y,m)}}$.

Eventually, the forger returns a signature σ. We may write σ as

$$\sigma = \prod_{i=1}^{N} \prod_{j=1}^{k_i} \left(H(X_i, Y_i, m_{i,j})^{x_i}\right)^{s_{i,j}},$$

where $m_{i,j}$ is a message signed by the private key x_i corresponding to the public key $\langle X_i, Y_i\rangle$ and the message $m_{i,j}$ has multiplicity $s_{i,j}$.

For each $i = 2, \ldots, N$, and each message $m_{i,j}$, since we know X_i and $a_{(X_i,Y_i,m_{i,j})}$, we may compute

$$\sigma_i = \prod_{j=1}^{k_i} X_i^{a_{(X_i,Y_i,m_{i,j})} \cdot s_{i,j}} = \prod_{j=1}^{k_i} \left(H(X_i, Y_i, m_{i,j})^{x_i}\right)^{s_{i,j}}.$$

Set

$$\sigma_1 = \sigma \cdot \prod_{i=2}^{N} \sigma_i^{-1},$$
$$A = \{m_{i_j} \in \{0,1\}^* \mid H(X_1, Y_1, m) = g_1^y \cdot g_1^{a_{(X_1,Y_1,m)}}\},$$
$$B = \{m_{i_j} \in \{0,1\}^* \mid H(X_1, Y_1, m) = g_1^{a_{(X_1,Y_1,m)}}\}.$$

If $A = \varnothing$, we abort. Otherwise, we rewrite σ_1 as

$$\begin{aligned}
\sigma_1 &= \prod_{j=1}^{k_1} \left(H(X_1, Y_1, m_{1,j})^x\right)^{s_{1,j}} \\
&= \left(\prod_{m_{1,j} \in A} \left(g_1^{xy} \cdot g_1^{xa_{(X_1,Y_1,m_{1,j})}}\right)^{s_{1,j}}\right)\left(\prod_{m_{1,j} \in B} \left(g_1^{xa_{(X_1,Y_1,m_{1,j})}}\right)^{s_{1,j}}\right) \\
&= g_1^{xy \sum_{m_{1,j} \in A} s_{1,j}} \cdot \prod_{j=1}^{k_1} g_1^{xa_{(X_1,Y_1,m_{1,j})} s_{1,j}} \\
&= g_1^{xy \sum_{m_{1,j} \in A} s_{1,j}} \cdot (g_1^x)^{\sum_{j=1}^{k_1} (a_{(X_1,Y_1,m_{1,j})} s_{1,j})}.
\end{aligned}$$

Finally, we compute

$$\left(\sigma_1 \cdot (g_1^x)^{-\sum_{j=1}^{k_1} (a_{(X_1,Y_1,m_{1,j})} s_{1,j})}\right)^{\left(\sum_{m_{1,j} \in A} s_{1,j}\right)^{-1}} = g_1^{xy}.$$

Now, we analyze the probability of success. The probability that we successfully respond to a single signing query is $q_S/(1+q_S)$. Since $k_1 > 0$ and $\sum_{j=1}^{k_1} s_{1,j} > 0$, we have that $A \neq \varnothing$ with probability at least $1/(q_S + 1)$. Putting these observations together, we obtain that the reduction succeeds with probability at least

$$\left(\frac{q_S}{q_S + 1}\right)^{q_S} \frac{1}{q_S + 1} = \left(1 - \frac{1}{q_S + 1}\right)^{q_S} \frac{1}{q_S + 1} \geq \frac{1}{e(q_S + 1)}.$$

□

4.3. The Katz-Wang bit

In addition to lifting the distinct signer/message restriction, Bellare-Namprempre-Neven [**2**] observed that the technique of Katz-Wang [**20**] can be used to obtain a tight reduction to co-CDH′. We mention three different versions of this technique.

The first version is the direct application of the technique used by Katz and Wang.

KW1-BGLS. *Let $H\colon \{0,1\}^* \to G_1$ be a hash function.*

We define the algorithms `Generate`*,* `Sign`*,* `Aggregate`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes $X = g_1^x$, $Y = g_2^x$. The algorithm returns the private key x and the public key $\langle X, Y\rangle$.*

`Sign` *takes as input a private key x, a message m. A bit $b_m \in \{0,1\}$ is selected uniformly at random and the algorithm returns the signature $\langle \sigma, \beta\rangle = \langle H(b_m, g_1^x, g_2^x, m)^x, b_m\rangle$. The bit b_m is fixed for each message m.*

`Aggregate` *takes as input valid signatures $\langle \sigma_1, \beta_1\rangle$, ..., $\langle \sigma_N, \beta_N\rangle$, computes*

$$\sigma = \prod_{i=1}^{N} \sigma_i,$$

and returns the signature $\langle \sigma, \beta_1, \ldots, \beta_N\rangle$.

`Verify` *takes as input public keys* $\langle X_1, Y_1\rangle$, *...,* $\langle X_N, Y_N\rangle$*, messages* m_1*, ...,* m_N*, and a signature* $\langle \sigma, \beta_1, \ldots, \beta_N\rangle$*. For each key* $\langle X_i, Y_i\rangle$*, the algorithm verifies that the key was correctly constructed by checking*

$$e(X_i, g_2) = e(g_1, Y_i). \tag{7}$$

This key verification step needs to be done only once per public key, across all invocations of the `Verify` *algorithm. To verify the actual signature, the algorithm checks*

$$e(\sigma, g_2) = \prod_{i=1}^{N} e(H(\beta_i, X_i, Y_i, m_i), Y_i). \tag{8}$$

The algorithm accepts the signature iff equations (7) *and* (8) *hold.*

The following lemma gives us a tight reduction from co-CDH′ to the KW1-BGLS scheme.

Lemma 10. *Given a KW1-BGLS* $\langle q_H, q_S, N, t, \varepsilon\rangle$*-forger, we can construct a co-CDH′* $\langle t + t', \varepsilon \cdot \varepsilon'\rangle$*-solver which makes one query to the forger, where*

$$t' = q_S + q_H + N,$$
$$\varepsilon' = \frac{1}{2}.$$

Proof. We fix an instance of the co-CDH′ problem. We are given elements g_1^x, g_1^y, and g_2^x. Our goal is to produce g_1^{xy}. We will construct an instance of the KW1-BGLS signature scheme with public key $\langle X_1, Y_1\rangle = \langle g_1^x, g_2^x\rangle$ and run the forger. We must respond to hash queries and signing queries.

For each hash function query on input (b, X, Y, m), we assume that $H(b, X, Y, m)$ has not been defined before. We will define $H(b, X, Y, m)$ and $H(1-b, X, Y, m)$ simultaneously. Select a random bit $b' = b_{(X,Y,m)} \in \{0, 1\}$ and integers $a' = a_{(b',X,Y,m)}$, $a = a_{(1-b',X,Y,m)} \in \mathbb{Z}_p$ uniformly at random. We set $H(b', X, Y, m) = (g_1)^{a'}$. If $\langle X, Y\rangle = \langle X_1, Y_1\rangle$, we set $H(1 - b', X, Y, m) = g_1^y \cdot (g_1)^a$. Otherwise, we set $H(1 - b', X, Y, m) = (g_1)^a$. We have now defined $H(b, X, Y, m)$ and so we return $H(b, X, Y, m)$.

To respond to a signature query on the message m, we let $b' = b_{(X,Y,m)}$ and compute $(g_1^x)^{a_{(b',X,Y,m)}} = H(b', X, Y, m)^x$ and return $\langle \sigma, b'\rangle$.

Eventually, the adversary returns an aggregate signature

$$\sigma = \prod_{i=1}^{N} \prod_{j=1}^{k_i} \prod_{b=0}^{1} \left(H(b, X_i, Y_i, m_{i,j})^{x_i}\right)^{s_{b,i,j}},$$

where $m_{i,j}$ is a message signed by the private key x_i corresponding to the public key $\langle X_i, Y_i\rangle$ and the message $m_{i,j}$ has multiplicity $s_{b,i,j}$ for the bit b. That is, the value $H(b, X_i, Y_i, m_{i,j})^{x_i}$ was aggregated into the signature $s_{b,i,j}$ times.

For each $i = 2, \ldots, N$, we may compute

$$\sigma_i = \prod_{j=1}^{k_i} \prod_{b=0}^{1} X_i^{a_{(b,X_i,Y_i,m_{i,j})} s_{b,i,j}} = \prod_{j=1}^{k_i} \prod_{b=0}^{1} \left(H(b, X_i, Y_i, m_{i,j})^{x_i}\right)^{s_{b,i,j}}.$$

We set

$$\sigma_1 = \sigma \cdot \prod_{i=2}^{N} \sigma_i^{-1} = \prod_{j=1}^{k_1} \prod_{b=0}^{1} \left(H(b, X_1, Y_1, m_{1,j})^{x_1}\right)^{s_{b,1,j}}.$$

We set $b_{i,j} = b_{(X_i,Y_i,m_{i,j})}$. Now, we may rewrite σ_1 as

$$\begin{aligned}\sigma_1 &= \prod_{j=1}^{k_1} \Big(\left(H(b_{1,j}, X_1, Y_1, m_{1,j})^{x_1}\right)^{s_{b_{1,j},1,j}} \cdot \left(H(1-b_{1,j}, X_1, Y_1, m_{1,j})^{x_1}\right)^{s_{1-b_{1,j},1,j}} \Big) \\ &= \prod_{j=1}^{k_1} \Big(\left((g_1^x)^{a_{(b_{1,j},X_1,Y_1,m_{1,j})} \cdot s_{b_{1,j},1,j}}\right) \cdot \left((g_1^x)^{(y+a_{(1-b_{1,j},X_1,Y_1,m_{1,j})}) \cdot s_{1-b_{1,j},1,j}}\right) \Big) \\ &= \Big(\prod_{j=1}^{k_1} \prod_{b=0}^{1} (g_1^x)^{a_{(b,X_1,Y_1,m_{1,j})} \cdot s_{b,1,j}} \Big) \cdot (g_1^{xy})^{\sum_{j=1}^{k_1} s_{1-b_{1,j},1,j}}.\end{aligned}$$

If $\sum_{j=1}^{k_i} s_{b_{1,j},1,j} = 0$, we abort. Otherwise, we compute

$$\sigma_1 \cdot \Big((g_1^x)^{-\sum_{j=1}^{k_1} \sum_{b=0}^{1} a_{(b,X_1,Y_1,m_{1,j})} \cdot s_{b,1,j}} \Big)^{\left(\sum_{j=1}^{k_1} s_{1-b_{1,j},1,j}\right)^{-1}} = g_1^{xy}.$$

Since $k_1 \geq 1$, with probability at least 1/2, at least one message $m_{1,j}$ signed with the key $\langle X_1, Y_1 \rangle$ in the aggregate signature is such that $s_{1-b_{1,j},1,j} \geq 1$. Therefore, with probability at least 1/2, we have that

$$\sum_{j=1}^{k_1} s_{1-b_{1,j},1,j} \geq 1.$$

We conclude that the reduction succeeds with probability at least 1/2. □

Unfortunately, the bit-length of signatures in the KW1-BGLS scheme increases linearly with the number of signatures aggregated. In practice, the one-bit increase per signature is negligible. For hypothetical applications where a fixed signature size is a hard requirement, we offer two alternatives.

The next variation produces aggregate signatures of a fixed size but becomes infeasible when a large number of signatures are aggregated.

KW2-BGLS. *Let $H\colon \{0,1\}^* \to G_1$ be a hash function. We define the algorithms* `Generate`*,* `Sign`*,* `Aggregate`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes $X = g_1^x$, $Y = g_2^x$. The algorithm returns the private key x and the public key $\langle X, Y \rangle$.*

`Sign` *takes as input a private key x, a message m. The algorithm selects a bit $b_m \in \{0,1\}$ uniformly at random and returns the signature $\sigma = H(b_m, g_1^x, g_2^x, m)^x$. The bit b_m is selected so that b_m is fixed for a given message m.*

`Aggregate` *takes as input valid signatures $\sigma_1, \ldots, \sigma_N$ and returns the signature*

$$\sigma = \prod_{i=1}^{N} \sigma_i.$$

`Verify` *takes as input public keys* $\langle X_1, Y_1\rangle$, …, $\langle X_N, Y_N\rangle$*, messages* m_1, …, m_N*, and a signature* σ*. For each key* $\langle X_i, Y_i\rangle$*, the algorithm verifies that the key was correctly constructed by checking*

$$e(X_i, g_2) = e(g_1, Y_i). \tag{9}$$

This key verification step needs to be done only once per public key, across all invocations of the `Verify` *algorithm. To verify the actual signature, for each* $b \in \{0,1\}^N$*, the algorithm checks*

$$e(\sigma, g_2) = \prod_{i=1}^{N} e(H(b_i, X_i, Y_i, m_i), Y_i). \tag{10}$$

The algorithm accepts the signature iff equation (9) *holds and there exists* $b \in \{0,1\}^N$ *such that* (10) *holds.*

To verify a signature, we must check on average 2^{N-1} conditions, computing at least one pairing for each check. As we see in the next lemma, the KW2-BGLS scheme enjoys the same security of KW1-BGLS.

Lemma 11. *Given a KW2-BGLS* $\langle q_H, q_S, N, t, \varepsilon\rangle$*-forger, we can construct a KW1-BGLS* $\langle q_H, q_S, N, t + 2^N, \varepsilon\rangle$*-forger.*

Proof. We fix an instance of the KW1-BGLS scheme. We run the KW2-BGLS on this instance of the KW1-BGLS scheme. We obtain a signature σ on messages m_1, …, m_N corresponding to keys $\langle X_1, Y_1\rangle$, …, $\langle X_N, Y_N\rangle$ such that there exists $b \in \{0,1\}^N$ with

$$e(\sigma, g_2) = \prod_{i=1}^{N} e(H(b_i, X_i, Y_i, m_i), Y_i).$$

Clearly, we have that $\langle \sigma, b_1, \ldots, b_N\rangle$ is a valid KW1-BGLS signature. □

As the number of signatures we aggregate grows, the KW2-BGLS scheme becomes infeasible. The next variation is a sequential aggregate signature which fixes the size of aggregate signatures and verification time does not increase exponentially with the number of signatures aggregated.

KW3-BGLS. *Let* $H\colon \{0,1\}^* \to G_1$ *be a hash function. We define the algorithms* `Generate`*,* `Sign-Aggregate`*, and* `Verify`*.*

`Generate` *selects an integer* $x \in \mathbb{Z}_p$ *uniformly at random and computes* $X = g_1^x$*,* $Y = g_2^x$*. The algorithm returns the private key* x *and the public key* $\langle X, Y\rangle$*.*

`Sign-Aggregate` *takes as input a message* m*, a private key* x*, and optionally a valid signature* $\langle \sigma', \beta\rangle$*. If no signature is given as part of the input, then the algorithm selects a bit* $\beta \in \{0,1\}$ *uniformly at random, computes* $\sigma = H(\beta, g_1^x, g_2^x, m)$*, and returns the signature* $\langle \sigma, \beta\rangle$*. If a signature* $\langle \sigma', \beta\rangle$ *is given as input, the algorithm computes* $\sigma = \sigma' \cdot H(\beta, g_1^x, g_2^x, m)$ *and returns the signature* $\langle \sigma', \beta\rangle$*. The bit* β *is determined by the first message.*

`Verify` *takes as input public keys* $\langle X_1, Y_1\rangle$, …, $\langle X_N, Y_N\rangle$*, messages* m_1, …, m_N*, and a signature* $\langle \sigma, \beta\rangle$*. For each key* $\langle X_i, Y_i\rangle$*, the algorithm verifies that the key was correctly constructed by checking*

$$e(X_i, g_2) = e(g_1, Y_i). \tag{11}$$

This key verification step needs to be done only once per public key, across all invocations of the `Verify` *algorithm. To verify the actual signature, the algorithm checks*

$$e(\sigma, g_2) = \prod_{i=1}^{N} e(H(\beta, X_i, Y_i, m_i), Y_i). \tag{12}$$

The algorithm accepts the signature iff equations (11) *and* (12) *hold.*

Lemma 12. *Given a KW3-BGLS $\langle q_H, q_S, N, t, \varepsilon \rangle$-forger, we can construct a KW1-BGLS $\langle q_H, q_S, N, \varepsilon \rangle$-forger.*

Proof. The result follows from the fact that if $\langle \sigma, \beta \rangle$ is a KW3-BGLS signature, then $\langle \sigma, \beta_1, \ldots, \beta_N \rangle$ is a KW1-BGLS signature, for $\beta_1 = \cdots = \beta_N = \beta$. $\square$

4.4. Public key proof-of-possession

For Type 1 and Type 2 pairings, we are given an efficiently computable isomorphism $\psi \colon G_2 \to G_1$. In the BGLS scheme, this allows for a shorter public key; we may exclude the element X from the public key $\langle X, Y \rangle$. The element X is not used in the protocol and is only required for the proof of Lemma 8. For Type 3 pairings, no such isomorphism is known to exist and we require the element X for the security proof. We show that for Type 3 pairings, a *proof-of-possesion* can be used to exclude the element X from the scheme and maintain the security provided by Lemma 8.

In this modified scheme, after generating public and private keys, a user must sign a special message for some certifying authority. The message m_Y determined by the public key Y could be something like "I, Alice, know the private key corresponding to Y". Fortunately, the private key is not revealed to the certifying authority. The technique can be applied to BNN-BGLS, KW1-BGLS, KW2-BGLS, or KW3-BGLS. For simplicity purposes, we modify the BGLS scheme.

POP-BGLS. *Let $H \colon \{0,1\}^* \to G_1$, $H_P \colon \{0,1\}^* \to G_1$ be hash functions.*

We define the algorithms `Generate`*,* `Pop`*,* `Sign`*,* `Aggregate`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes $Y = g_2^x$. The algorithm returns the private key x and the public key Y.*

`Pop` *takes an input a signature $\sigma = H_P(m_Y)^x$ on a message m_Y and verifies that*

$$e(\sigma, g_2) = e(H_P(m_Y), Y).$$

The message m_Y is corresponds to the public key $Y = g_2^x$. After the the `Pop` *algorithm is run on public key Y, we say that Y is* pop-certified.

`Sign` *takes as input a private key x, a message m, and returns the signature $\sigma = H(m)^x$.*

`Aggregate` *takes as input valid signatures $\sigma_1, \ldots, \sigma_N$ on distinct messages and returns the signature*

$$\sigma = \prod_{i=1}^{N} \sigma_i.$$

`Verify` *takes as input pop-certified public keys $Y_1, \ldots, Y_N$, messages $m_1, \ldots, m_N$, and a signature σ. The algorithm accepts the signature iff*

$$e(\sigma, g_2) = \prod_{i=1}^{N} e(H(m_i), Y_i).$$

We prove the scheme secure in the following lemma. The proof is similar to that of Lemma 8 except that the random oracle H_P in effect allows us to construct X ourselves.

Lemma 13. *Given a POP-BGLS $\langle q_H, q_{H_P}, q_S, N, t, \varepsilon\rangle$-forger, we can construct a co-CDH′ $\langle t + t', \varepsilon \cdot \varepsilon'\rangle$-solver where*

$$t' = q_H + q_S + N,$$
$$\varepsilon' = \frac{1}{q_S + N}.$$

Proof. We fix an instance of the co-CDH′ problem. We are given elements g_1^x, g_1^y, and g_2^x. Our goal is to produce g_1^{xy}. We construct an instance of the POP-BGLS scheme with public key $Y_1 = g_2^x$. We must respond to hash queries of H, hash queries of H_P, and signature queries.

To respond to a hash query of H on input m, we select an integer $a_m \in \mathbb{Z}_p$ uniformly at random and with probability $1/(q_S + N)$, we return $H(m) = g_1^y \cdot g_1^{a_m}$. With probability $(q_S + N - 1)/(q_S + N)$, we return $H(m) = g_1^{a_m}$.

To respond to a hash query of H_P on input m_Y, we select an integer $b_Y \in \mathbb{Z}_p$ uniformly at random and return $H_P(m_Y) = g_1^{b_Y}$.

To respond to a signature query on the message m, we abort if $H(m) = g_1^y \cdot g_1^{a_m}$. Otherwise, we compute and return $(g_1^x)^{a_m} = H(m)^x$.

Eventually, the forger returns a signature σ of messages $m_1, \ldots, m_N$ with certified public keys $Y_1, \ldots, Y_N$. If $H(m_1) = g_1^{a_{m_1}}$, or if some $i = 2, \ldots, N$, we have $H(m_i) = g_1^y \cdot g_1^{a_{m_i}}$, then we abort. Otherwise, we have that

$$\sigma = \prod_{i=1}^{N} H(m_i)^{x_i} = (g_1^y \cdot g_1^{a_{m_1}})^{x_1} \cdot \prod_{i=2}^{N} (g_1^{x_i})^{a_{m_i}} = g_1^{xy} \cdot \prod_{i=1}^{N} (g_1^{x_i})^{a_{m_i}}$$

where x_i is the private key corresponding to Y_i. Since each public key Y_i is certified, we are given $H_P(m_{Y_i})^{x_i}$. We set $X_1 = g_1^x$ and for each $i = 2, \ldots, N$, we compute

$$X_i = (H_P(m_{Y_i})^{x_i})^{b_Y^{-1}} = g_1^{x_i}.$$

Finally, we compute

$$\sigma \cdot \prod_{i=1}^{N} X_i^{-a_{m_i}} = g_1^{xy}.$$

With probability $q_S/(q_S + N)$, we are able to respond to a given signature query. With probability

$$\left(\frac{q_S}{q_S + N}\right)^{N-1} \frac{1}{q_S + N},$$

the forger returns a signature of the desired form. The reduction succeeds with probability

$$\Big(\frac{q_S}{q_S+N}\Big)^{q_S+N-1}\frac{1}{q_S+N} = \Big(1-\frac{1}{q_S+N}\Big)^{q_S+N-1}\frac{1}{q_S+N} \geq \frac{1}{e(q_S+N)}.$$

□

4.5. Point compression

Just as in the BLS signature scheme, we may compress points on an elliptic curve defined over some finite field $\mathbb{F}$. Given a point P on an elliptic curve, we have that $P = (x, y)$ where x, $y \in \mathbb{F}$. The field element x determines two points on the elliptic curve. With the addition of an extra bit, we may reduce the memory requirements of storing P by almost 1/2.

We may desire (to possibly avoid a patent) to remove this extra bit. In Chapter 3, we showed that removing the y-coordinate entirely introduced a manageable amount of ambiguity. We may naively extend this technique to BGLS. We switch to additive notation. Let P_1 be a generator of G_1 and P_2 be a generator of G_2.

For a point $Q = (x, y)$ on an elliptic curve, we let $\widetilde{Q} = x$. We define $\widetilde{G}_1 = \{\widetilde{Q} \in \mathbb{F} \mid Q \in G_1\}$. For the identity element ∞ of the elliptic curve, we use a bit-string of length at most $\log_2|\mathbb{F}| + 1$ to represent $\widetilde{\infty}$.

For simplicity's sake, we restrict ourselves to modifying the BGLS scheme. We could apply the technique to any of the schemes previously mentioned in this chapter.

C1-BGLS. *Let $H\colon \{0,1\}^* \to G_1$ be a hash function. We define the algorithms* `Generate`*,* `Sign`*,* `Aggregate`*, and* `Verify`*.*

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes $X = xP_1$, $Y = xP_2$. The algorithm returns the public key $\langle\widetilde{X}, \widetilde{Y}\rangle$ and the private key x.*

`Sign` *takes as input a message m and a private key x. The algorithm computes $Q = xH(m)$ and returns the signature $\sigma = \widetilde{Q}$.*

`Aggregate` *takes as input valid signatures $\sigma_1, \ldots, \sigma_N$. For each $i = 1, \ldots, N$, we let Q_i be such that $\widetilde{Q}_i = \sigma_i$. The algorithm computes*

$$R = \sum_{i=1}^{N} Q_i$$

and returns $\sigma = \widetilde{R}$.

`Verify` *takes as input messages $m_1, \ldots, m_N$ corresponding to public keys $\langle\widetilde{X}_1, \widetilde{Y}_1\rangle, \ldots, \langle\widetilde{X}_N, \widetilde{Y}_N\rangle$, and a signature σ. The algorithm computes Q such that $\widetilde{Q} = \sigma$. For each key $\langle\widetilde{X}_i, \widetilde{Y}_i\rangle$, the elements X_i, Y_i can be determined up to sign. To verify that the key was correctly constructed, the algorithm checks that one of the following equations hold:*

$$e(X_i, P_2) = e(P_1, Y_i),$$
$$e(X_i, P_2) = e(P_1, Y_i)^{-1}.$$

This key verification step needs to be done only once per public key, across all invocations of the `Verify` *algorithm. If one of the two equations above holds and there exists* $b \in \{-1, 1\}^N$ *such that*

$$e(Q, P_2) = \prod_{i=1}^{N} e(H(m_i), Y_i)^{b_i},$$

then the algorithm accepts the signature.

We show in the next lemma that a C1-BGLS forger can be used to construct a BGLS forger.

Lemma 14. *Given a C1-BGLS* $\langle q_H, q_S, N, t, \varepsilon\rangle$*-forger, we can construct a BGLS* $\langle q_H, q_S, N, t + 2^N, \varepsilon\rangle$*-forger.*

Proof. We begin with an instance of the BGLS scheme with a hash function H and a challenge public key $\langle X_1, Y_1\rangle$. We construct an instance of the C1-BGLS scheme with public key $\langle \widetilde{X}_1, \widetilde{Y}_1\rangle$ and run the C1-BGLS forger. We need to respond to hash queries and signing queries.

To respond to a hash query on the message m, we return $H(m)$. To respond to a signing query on the message m, we use the BGLS signing oracle to obtain $Q = xH(m)$ and respond with $\widetilde{Q}$.

Eventually, the forger returns a signature σ on messages $m_1, \ldots, m_N$ and public keys $\langle \widetilde{X}_1, \widetilde{Y}_1\rangle, \ldots, \langle \widetilde{X}_N, \widetilde{Y}_N\rangle$ where no signing query was made for m_1. For each i, we select X_i, Y_i such that

$$e(X_i, P_2) = e(P_1, Y_i).$$

We enumerate the elements of $\{-1, 1\}^N$ until we obtain $b \in \{-1, 1\}^N$ such that

$$e(Q, P_2) = \prod_{i=1}^{N} e(H(m_i), Y_i)^{b_i}.$$

We return the signature $\sigma' = b_1 Q$, the messages $m_1, \ldots m_N$, and public keys $\langle X_1, Y_1\rangle, \langle b_1 b_2 X_2, b_1 b_2 Y_2\rangle, \ldots, \langle b_1 b_N X_N, b_1 b_N Y_N\rangle$. This is a valid BGLS forgery. $\square$

Unfortunately, the scheme becomes infeasible as the number of messages aggregated into the signature increases.

By hashing bitstrings into pairs of elliptic curve points, we obtain an efficient scheme. For each element h of the finite field underlying some elliptic curve E, the element h determines at most two elements of the elliptic curve. We arbitrarily fix one element P_h of E such that $h = \widetilde{P}_h$. Since a point consists of two field elements x, y and field elements are represented by bitstrings, we can take, for instance, the smallest of the two bistrings representing the y's under the lexicographical ordering.

C2-BGLS. *Let* $H\colon \{0,1\}^* \to \widetilde{G}_1$ *be a hash function. We define the algorithms* `Generate`, `Sign`, `Aggregate`, *and* `Verify`.

`Generate` *selects an integer* $x \in \mathbb{Z}_p$ *uniformly at random and computes* $X = xP_1$, $Y = xP_2$. *If* $P_{\widetilde{Y}} = Y$, *then the algorithm returns the public key* $\langle \widetilde{X}, \widetilde{Y}\rangle$ *and private key* x. *Otherwise, the algorithm returns the public key* $\langle \widetilde{X}, \widetilde{Y}\rangle$ *and private key* $-x$.

`Sign` *takes as input a message m and a private key x. The algorithm computes $Q = xP_{H(m)}$ and returns the signature $\sigma = \widetilde{Q}$.*

`Aggregate` *takes as input valid signatures $\sigma_1, \ldots, \sigma_N$. For each $i = 1, \ldots, N$, the algorithm determines $Q_i = P_{\sigma_i}$ such that $\widetilde{Q}_i = \sigma_i$. The algorithm computes*

$$R = \sum_{i=1}^{N} Q_i$$

and returns $\sigma = \widetilde{R}$.

`Verify` *takes as input messages $m_1, \ldots, m_N$ corresponding to public keys $\langle \widetilde{X}_1, \widetilde{Y}_1 \rangle, \ldots, \langle \widetilde{X}_N, \widetilde{Y}_N \rangle$, and a signature σ. We compute Q such that $\widetilde{Q} = \sigma$. For each key $\langle \widetilde{X}_i, \widetilde{Y}_i \rangle$, we may determine the X_i, Y_i up to sign. Take Y_i to be such that $P_{\widetilde{Y}_i} = Y_i$. To verify that the key was correctly constructed, we check that one of the following equations hold:*

$$e(X_i, P_2) = e(P_1, Y_i),$$
$$e(X_i, P_2) = e(P_1, Y_i)^{-1}.$$

This key verification step needs to be done only once per public key, across all invocations of the `Verify` *algorithm. If one of the two equations above holds and there exists $b \in \{-1, 1\}$ such that*

$$e(Q, P_2)^b = \prod_{i=1}^{N} e(P_{H(m_i)}, Y_i),$$

then the algorithm accepts the signature.

Remark. All previous schemes permitted sequential aggregation. That is, given only an "aggregate signature so far" σ, a new message m, and a private key, a user may append a signature on the message m to σ without knowledge of the previous messages and keys. Unfortunately, the same is not true for C2-BGLS.

Consider a signature σ on messages $m_1, \ldots, m_N$ and public keys $\langle \widetilde{X}_1, \widetilde{Y}_1 \rangle, \ldots, \langle \widetilde{X}_N, \widetilde{Y}_N \rangle$. If we wish to perform sequential aggregation, we determine Q such that $\widetilde{Q} = \sigma$ and

$$e(Q, P_2) = \prod_{i=1}^{N} e(P_{H(m_i)}, P_{\widetilde{Y}_i}).$$

To aggregate a copy of m signed under the private key x into the signature σ, we compute

$$R = Q + xP_{H(m)}$$

and return the signature $\widetilde{R}$.

In the next lemma, we show that we can construct a BGLS forger, using a C2-BGLS forger.

Lemma 15. *Given a C2-BGLS $\langle q_H, q_S, N, t, \varepsilon \rangle$-forger, we can construct a BGLS $\langle q_H, q_S, N, t, \varepsilon \rangle$-forger.*

Proof. We are given a BGLS instance with hash function $\hat{H}$ and challenge public key $\langle X_1, Y_1 \rangle$ corresponding to the private key x. Let $b \in \{-1, 1\}$ be such that $bY_1 = P_{\widetilde{Y}_1}$. We construct an instance C2-BGLS with hash function H and challenge key $\langle \widetilde{X}_1, \widetilde{Y}_1 \rangle$ and we run the C2-BGLS forger. We need to respond to queries to the hash function H and signing queries.

To respond to a hash query on the message m, we compute $Q = \hat{H}(m)$ and return $H(m) = \widetilde{Q}$.

To respond to a signature query on message m, we use our BGLS signing oracle to obtain $\sigma_m = x\hat{H}(m)$. We return $\widetilde{\sigma}_m$.

Eventually, the C2-BGLS forger returns a signature σ on messages $m_1, \ldots, m_N$ and public keys $\langle \widetilde{X}_1, \widetilde{Y}_1 \rangle, \ldots, \langle \widetilde{X}_N, \widetilde{Y}_N \rangle$. Let $Q = P_\sigma$. For each $i = 2, \ldots, N$, let Y_i be such that $Y_i = P_{\widetilde{Y}_i}$ and take X_i to be such that

$$e(X_i, P_2) = e(P_1, Y_i).$$

We have that there exists $c \in \{-1, 1\}$ such that

$$e(Q, P_2)^c = e(\hat{H}(m_1), bY_1) + \sum_{i=2}^{N} e(\hat{H}(m_i), Y_i).$$

Therefore,

$$Q = cbx_1 H(m_1) + c \sum_{i=2}^{N} x_i \hat{H}(m_i),$$

where x_i is the private key corresponding to $\langle X_i, Y_i \rangle$. Set

$$\sigma' = bcQ$$

and return the signature σ', a BGLS forgery on messages $m_1, \ldots, m_N$ and public keys $\langle X_1, Y_1 \rangle$, $\langle bX_2, bY_2 \rangle, \ldots, \langle bX_N, bY_N \rangle$ □

CHAPTER 5

The Waters Signature Scheme

5.1. The scheme

In 2005, Waters presented an Identity-Based Encryption scheme based on the decisional bilinear Diffie-Hellman assumption, without the use of random oracles [**32**]. Based on this scheme, Waters describes a signature scheme which is secure in the standard model relative to the co-CDH$'$ problem. That is, the scheme is secure without assuming the existence of a random oracle.

Waters presented the scheme in the special case $G_1 = G_2$. We present the scheme in its full generality.

The Waters signature scheme (Waters). *We define the functions* `Generate`*,* `Sign`*, and* `Verify`*. By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Generate` *selects integers x, $y \in \mathbb{Z}_p$ and elements c', c_1, $\ldots$, $c_n \in \mathbb{Z}_1$ uniformly at random. The algorithm returns the private key $Z = g_1^{xy}$ and the public key $\langle X, Y\rangle = \langle g_1^x, g_2^y\rangle$, $u' = g_1^{c'}$, $u_1 = g_1^{c_1}$, $\ldots$, $u_n = g_1^{c_n}$. For notational convenience we define the functions $H\colon \{0,1\}^n \to G_1$ and $K\colon \{0,1\} \to \mathbb{Z}_p$ by*

$$H(m) = H(m_1, \ldots, m_n) = u' \cdot \prod_{i=1}^{n} u_i^{m_i};$$

$$K(m) = K(m_1, \ldots, m_n) = c' + \sum_{i=1}^{n} c_i m_i.$$

It should be noted that since u', u_1, $\ldots$, u_n are public, the function H is public.

`Sign` *takes as input a message m and a private key Z. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\sigma = \langle Z \cdot g_1^{rK(m)}, g_1^r, g_2^r\rangle.$$

`Verify` *takes as input a message m, a public key $\langle X, Y\rangle$, and a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$. The algorithm checks*

$$e(\sigma_2, g_2) = e(g_1, \sigma_3) \tag{13}$$

and

$$e(\sigma_1, g_2) = e(X, Y) \cdot e(H(m), \sigma_3). \tag{14}$$

The algorithm accepts the signature iff equations (13) *and* (14) *hold.*

For a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$, although it seems that the component σ_2 is superfluous, the component is necessary for the security of the Waters scheme. However, if the pairing is Type 1 or Type 2, then we are given an efficiently computable isomorphism $\psi\colon G_2 \to G_1$ such that $\psi(g_2) = g_1$ and may exclude the component σ_2 from the signature. Since we may compute $\sigma_2 = \psi(\sigma_3)$ such that Equation (13) holds, the proof of security remains valid and we need not check Equation (14) as part of the `Verify` algorithm.

An algorithm is said to be a *Waters $\langle q_S, n, t, \varepsilon\rangle$-forger*, if after q_S queries to a signing oracle, the algorithm outputs the signature of some message not queried, in time at most t, with probability at least ε.

Before we prove the Waters scheme secure, we give the following technical lemma. We will use this lemma to reduce solving the co-CDH′ problem to forging Waters signatures.

Waters lemma. *Let n, q_S be integers. Set $t = 2q_S$. Let $m, m_1, \ldots, m_{q_S} \in \{0,1\}^n$ be arbitrary distinct bit-strings. Let $a', a_1, \ldots, a_n \in \{0, \ldots, t-1\}$ be selected uniformly at random. Let $k \in \{0, \ldots, n\}$ be selected uniformly at random. Define $F\colon \{0,1\}^n \to \mathbb{Z}$ by*

$$F(\mu) = F(\mu_1, \ldots, \mu_n) = a' - kt + \sum_{i=1}^{n} a_i \mu_i.$$

Then

$$\Pr\Big[\big(F(m) = 0\big) \wedge \big(\bigwedge_{i=1}^{q_S} F(m_i) \neq 0\big)\Big] \geq \frac{1}{4q_S(n+1)}.$$

Proof. Let $\equiv_t$ denote equivalence modulo t. For an arbitrary $\mu \in \{0,1\}^n$, we claim the probability that $F(\mu) \equiv_t 0$ is $1/t$. To see this, we fix $\ell = -kt + \sum_{i=1}^{n} a_i \mu_i$ and notice that since a' is chosen at random, the probability that $a' \equiv_t -\ell$ is $1/t$.

If $F(\mu) \equiv_t 0$, then since $F(\mu) \leq (n+1)(t-1) - kt < (n+1-k)t$, the probability that $F(\mu) = 0$ is $1/(n+1)$, the probability of choosing the correct k. Therefore,

$$\Pr[F(\mu) = 0] = \frac{1}{t(n+1)}. \tag{15}$$

For each $i = 1, \ldots, q_S$, since m and m_i differ in at least one bit and since $F(m) = 0$ implies $F(m) \equiv_t 0$, the events $F(m) = 0$ and $F(m_i) \equiv_t 0$ are independent. So,

$$\Pr\big[F(m_i) \equiv_t 0 \mid F(m) = 0\big] = \frac{1}{t}. \tag{16}$$

We conclude,

$$\begin{aligned}
&\Pr\Big[\big(F(m)=0\big)\wedge\big(\bigwedge_{i=1}^{q_S} F(m_i)\neq 0\big)\Big]\\
&\geq \Pr\Big[\big(F(m)=0\big)\wedge\big(\bigwedge_{i=1}^{q_S} F(m_i)\not\equiv_t 0\big)\Big]\\
&= \Pr[F(m)=0]\Pr\Big[\bigwedge_{i=1}^{q_S} F(m_i)\not\equiv_t 0 \mid F(m)=0\Big]\\
&= \frac{1}{t(n+1)}\Pr\Big[\bigwedge_{i=1}^{q_S} F(m_i)\not\equiv_t 0 \mid F(m)=0\Big] && \text{by (15)}\\
&= \frac{1}{t(n+1)}\bigg(1-\Pr\Big[\bigvee_{i=1}^{q_S} F(m_i)\equiv_t 0 \mid F(m)=0\Big]\bigg)\\
&\geq \frac{1}{t(n+1)}\bigg(1-\sum_{i=1}^{q_S}\Pr\big[F(m_i)\equiv_t 0 \mid F(m)=0\big]\bigg)\\
&= \frac{1}{t(n+1)}\bigg(1-\frac{q_S}{t}\bigg) && \text{by (16)}\\
&= \frac{1}{4q_S(n+1)}.
\end{aligned}$$

□

In the next lemma, we show that forging Waters signatures is at least difficult as solving the co-CDH′ problem.

Lemma 16. *Given a Waters $\langle q_S, n, t, \varepsilon\rangle$-forger, we can construct a co-CDH′ $\langle t+t', \varepsilon\cdot\varepsilon'\rangle$-solver, where*

$$\begin{aligned}
t' &\leq q_S,\\
\varepsilon' &\geq \frac{1}{4q_S(n+1)}.
\end{aligned}$$

Proof. We begin with an instance of the co-CDH′ problem. We are given g_1^x, g_1^y, and g_2^x and we wish to produce g_1^{xy}.

We define a few terms which will be used to construct a Waters public key. Set $t = 2q_S$ and let $k \in \{0,\dots,n\}$ be selected uniformly at random. Let a', a_1, …, a_n be integers selected uniformly at random from $\{0,\dots,t-1\}$ and let b', b_1, …, b_n be integers selected uniformly at random from $\mathbb{Z}_p$. For notational convenience, we define the functions $F(m) = -kt + a' + \sum_{i=1}^n a_i m_i$ and $J(m) = b' + \sum_{i=1}^n b_i m_i$. Finally, set $u' = (g_1^y)^{-kt+a'} g_1^{b'}$ and for each $i = 1, \dots, n$, set $u_i = (g_1^y)^{a_i} g_1^{b_i}$. We observe that

$$H(m) = g_1^{yF(m)+J(m)}.$$

We create an instance of the Waters signature scheme with public key $\langle X, Y\rangle = \langle g_1^y, g_2^x\rangle$, u', u_1, …, u_n and run the forger.

To respond to a signature query on the message m, we abort if $F(m) = 0$. Otherwise, let $\hat{r} \in \mathbb{Z}_p$ be an integer selected uniformly at random. The signature is computed as

$$\sigma = \langle (g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}}, g_1^{\hat{r}} (g_1^x)^{-\frac{1}{F(m)}}, g_2^{\hat{r}} (g_2^x)^{-\frac{1}{F(m)}} \rangle.$$

To show that this is a valid signature, we set $r = \hat{r} - \frac{x}{F(m)}$ and so

$$\begin{aligned}
(g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}} &= (g_1^{xy})(g_1^{-xy})(g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}} \\
&= g_1^{xy} \big((g_1^{-y})(g_1)^{-\frac{J(m)}{F(m)}} \big)^x H(m)^{\hat{r}} \\
&= g_1^{xy} \big((g_1^y)^{F(m)} g_1^{J(m)} \big)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} \\
&= g_1^{xy} \big((g_1^{yF(m)+J(m)} \big)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} \\
&= g_1^{xy} H(m)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} = (g_1^{xy}) H(m)^{\hat{r} - \frac{x}{F(m)}} \\
&= g_1^{xy} H(m)^r.
\end{aligned}$$

Eventually, the forger returns the signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle = \langle g_1^{xy} H(m)^r, g_1^r, g_2^r \rangle$ for some m not queried. If $F(m) \neq 0$, we abort. Otherwise, we compute

$$\begin{aligned}
\sigma_1 (\sigma_2)^{-J(m)} &= \big(g_1^{xy} H(m)^r \big) \big(g_1^r \big)^{-J(m)} = g_1^{xy} g_1^{rJ(m)} g_1^{-rJ(m)} \\
&= g_1^{xy}.
\end{aligned}$$

Now, we analyze the probability that the reduction can successfully respond to all signature queries and return a forgery. We assume that the forger makes exactly q_S signing queries. Let $m_1, \ldots, m_{q_S}$ be the distinct messages given as input to the signing oracle. Let m be the message on which the forger returns a signature. The reduction succeeds if $F(m) = 0$ and $F(m_i) \neq 0$, for each $i = 1, \ldots, q_S$. Applying Waters lemma, we have, as required, that

$$\Pr \Big[\big(F(m) = 0 \big) \wedge \big(\bigwedge_{i=1}^{q_S} F(m_i) \neq 0 \big) \Big] \geq \frac{1}{4 q_S (n+1)}.$$

$\square$

5.1.1. Alternative public-key representation. If we desire, we can replace the public key $\langle X, Y \rangle$ with an element $\mathcal{X} = e(X, Y)$. For Type 2 pairings, the representation of G_T requires fewer bits than $G_1 \times G_2$. However, for Type 3, the converse is true. In addition, replacing $\langle X, Y \rangle$ with $\mathcal{X} = e(X, Y)$ results in one fewer pairing evaluation during varification. Depending on the type of pairing used and the cost of a pairing operation, one of the public key representations may be more desirable than the other.

For completeness, we present the modified scheme and reduce its security to the Waters scheme.

Waters with an alternative public-key representation (APK-Waters). *We define the functions* `Generate`, `Sign`, *and* `Verify`. *By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Generate` *selects elements Z, u', u_1, …, $u_n \in G_1$ uniformly at random. The algorithm returns the private key Z and the public key $\mathcal{X} = e(Z, g_2)$, u', u_1, …, u_n. For notational convenience we define the function $H\colon \{0,1\}^n \to G_1$ by*

$$H(m) = H(m_1, \ldots, m_n) = u' \cdot \prod_{i=1}^{n} u_i^{m_i}.$$

It should be noted that since u', u_1, …, u_n are public, the function H is public.

`Sign` *takes as input a message m and a private key Z. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\sigma = \langle Z \cdot (H(m))^r, g_1^r, g_2^r \rangle.$$

`Verify` *takes as input a message m, a public key $\mathcal{X}$, and a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$. The algorithm checks*

$$e(\sigma_2, g_2) = e(g_1, \sigma_3) \tag{17}$$

and

$$e(\sigma_1, g_2) = \mathcal{X} \cdot e(H(m), \sigma_3). \tag{18}$$

The algorithm accepts the signature iff equations (17) *and* (18) *hold.*

In the next lemma, we show that an APK-Waters forger can be used to forge Waters signatures.

Lemma 17. *Given an APK-Waters $\langle q_S, n, t, \varepsilon \rangle$-forger, we can construct a Waters $\langle q_S, n, t, \varepsilon \rangle$-forger.*

Proof. We fix an instance of the Waters signature scheme with challenge public-key $\langle X, Y \rangle$, u', u_1, …, u_n. We construct an instance of the APK-Waters scheme with challenge public-key $\mathcal{X} = e(X, Y)$, u', u_1, …, u_n and run the forger.

We need to respond to APK-Waters signing queries. Since APK-Waters signatures and Waters signatures verify identically, to respond to signature queries, we use the signing oracle given by the Waters scheme.

The forgery returned by the APK-Waters forger is also a forgery for the Waters instance. Since the reduction is deterministic, the result follows. □

Since an APK-Waters forger can be used to construct a Waters forger and a Waters forger can be used to construct a co-CDH′ solver, we obtain the following lemma.

Lemma 18. *Given an APK-Waters $\langle q_S, n, t, \varepsilon \rangle$-forger, we can construct a co-CDH′ $\langle t + t', \varepsilon \cdot \varepsilon' \rangle$-solver, where*

$$\begin{aligned} t' &\le q_S, \\ \varepsilon' &\ge \frac{1}{4 q_S (n+1)}. \end{aligned}$$

Proof. Follows from Lemma 16 and Lemma 17. □

5.1.2. Selecting u_i's. In the two versions of the Waters scheme which we have described, each user picks his own u_i's, contributing to the size of his public key. To help minimize the size of public keys, it is desirable to have the users share the u_i's. If we assume the existence of a *trusted third-party* (*TTP*), we can reduce the public key size by $n+1$ elements of G_1, while introducing some public parameters shared by all users.

A public key in the Waters scheme is of the form $(X, Y) \in G_1 \times G_2$. The TTP may select $Y_1 \in G_1$, $Y_2 \in G_2$ as a public parameter for all users. The element Y_1 is used for key generation and Y_2 for verification.

We present the following modification. A TTP runs an algorithm called `Setup` which returns public parameters.

Waters with a trusted third-party (TTP-Waters). *We define the functions* `Setup`*,* `Generate`*,* `Sign`*, and* `Verify`*. By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Setup` *selects integers $y \in \mathbb{Z}_p$, u', u_1, ..., $u_n \in G_1$ uniformly at random. The algorithm returns u', u_1, ..., u_n, and $Y_1 = g_1^y$, $Y_2 = g_2^y$. For notational convenience we define the function $H\colon \{0,1\}^n \to G_1$ by*

$$H(m) = H(m_1, \dots, m_n) = u' \cdot \prod_{i=1}^{n} u_i^{m_i}.$$

It should be noted that since u', u_1, ..., u_n are public, the function H is public.

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random. The algorithm returns the private key $Z = Y_1^x$ and the public key $X = g_1^x$.*

`Sign` *takes as input a message m and a private key Z. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\sigma = \langle Z \cdot (H(m))^r, g_1^r, g_2^r \rangle.$$

`Verify` *takes as input a message m, a public key X, and a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$. The algorithm checks*

$$e(\sigma_2, g_2) = e(g_1, \sigma_3) \tag{19}$$

and

$$e(\sigma_1, g_2) = e(X, Y_2) \cdot e(H(m), \sigma_3). \tag{20}$$

The algorithm accepts the signature iff equations (19) *and* (20) *hold.*

The next lemma reduces solving co-CDH′ to forging TTP-Waters signatures. The proof is essentially the same as that of Lemma 16.

Lemma 19. *Given a TTP-Waters $\langle q_S, n, t, \varepsilon \rangle$-forger, we can construct a co-CDH′ $\langle t + t', \varepsilon \cdot \varepsilon' \rangle$-solver, where*

$$t' \le q_S,$$
$$\varepsilon' \ge \frac{1}{4q_S(n+1)}.$$

Proof. We begin with an instance of the co-CDH′ problem. We are given elements g_1^x, g_1^y, g_2^x and wish to produce g_1^{xy}. To create an instance of TTP-Waters, we first define a few terms. Set $t = 2q_S$ and let $k \in \{0, \dots, n\}$ be selected uniformly at random. Let a', a_1, …, a_n be integers selected uniformly at random from $\{0, \ldots, t-1\}$ and let b', b_1, …, b_n be integers selected uniformly at random from $\mathbb{Z}_p$. For notational convenience, we define the functions $F(m) = -kt + a' + \sum_{i=1}^{n} a_i m_i$ and $J(m) = b' + \sum_{i=1}^{n} b_i m_i$. Finally, set $u' = (g_1^y)^{-kt+a'} g_1^{b'}$ and for each $i = 1$, …, n, set $u_i = (g_1^y)^{a_i} g_1^{b_i}$. We observe that

$$H(m) = g_1^{yF(m)+J(m)}.$$

We create an instance of the Waters signature scheme with public parameters $Y_1 = g_1^x$, $Y_2 = g_2^x$, u', u_1, …, u_n. We run the forger with the challenge public-key $X = g_1^y$.

To respond to a signature query on the message m, we abort if $F(m) = 0$. Otherwise, let $\hat{r} \in \mathbb{Z}_p$ be an integer selected uniformly at random. The signature is computed as

$$\sigma = \langle (g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}}, g_1^{\hat{r}} (g_1^x)^{-\frac{1}{F(m)}}, g_2^{\hat{r}} (g_2^x)^{-\frac{1}{F(m)}} \rangle.$$

To show that this is a valid signature, we set $r = \hat{r} - \frac{x}{F(m)}$ and so

$$\begin{aligned}
(g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}} &= (g_1^{xy})(g_1^{-xy})(g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}} \\
&= g_1^{xy} \big((g_1^{-y})(g_1)^{-\frac{J(m)}{F(m)}}\big)^x H(m)^{\hat{r}} \\
&= g_1^{xy} \big((g_1^y)^{F(m)} g_1^{J(m)}\big)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} \\
&= g_1^{xy} \big((g_1^{yF(m)+J(m)}\big)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} \\
&= g_1^{xy} H(m)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} = (g_1^{xy}) H(m)^{\hat{r}-\frac{x}{F(m)}} \\
&= g_1^{xy} H(m)^r.
\end{aligned}$$

Eventually, the forger returns the signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle = \langle g_1^{xy} H(m)^r, g_1^r, g_2^r \rangle$ for some m not queried. If $F(m) \neq 0$, we abort. Otherwise, we compute

$$\begin{aligned}
\sigma_1 (\sigma_2)^{-J(m)} &= \big(g_1^{xy} H(m)^r\big)\big(g_1^r\big)^{-J(m)} = g_1^{xy} g_1^{rJ(m)} g_1^{-rJ(m)} \\
&= g_1^{xy}.
\end{aligned}$$

Now, we analyze the probability that the reduction can successfully respond to all signature queries and return a forgery. We assume that the forger makes exactly q_S signing queries. Let m_1, …, m_{q_S} be the queried made to the signing oracle. Let m be the message on which the forger returns a signature.

The reduction succeeds if $F(m) = 0$ and $F(m_i) \neq 0$, for each $i = 1$, …, q_S. Applying Waters lemma, we have, as required, that

$$\Pr\Big[\big(F(m) = 0\big) \wedge \big(\bigwedge_{i=1}^{q_S} F(m_i) \neq 0\big)\Big] \geq \frac{1}{4q_S(n+1)}.$$

□

Since the TTP can compute the u_i's knowing their discrete logs, the TTP can recover the private key of a user, given only one signature produced by this user. To demonstrate this, we let $u' = g_1^{a'}$, $u_1 = g_1^{a_1}$, …, $g_1^{a_n}$. If a', a_1, …, a_n are known by the TTP and the TTP obtains a signature

$$\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle = \langle g_1^{xy} H(m)^r, g_1^r, g_2^r \rangle,$$

then the TTP can recover the private key used to produce this signature by computing

$$\sigma_1 \cdot (\sigma_2)^{-a' - \sum_{i=1}^{n} a_i m_i} = \sigma_1 \cdot H(m)^{-r} = g_1^{xy}.$$

Allowing a TTP to choose the u_i's, essentially reveals the private keys to the TTP. We may construct a threshold version of the construction of the u_i's which requires that all parties collude in order to recover private keys. Given k parties, for each party $i = 1, \ldots, k$, party i selects $a'^{(i)}$, $a_1^{(i)}, \ldots, a_n^{(i)}$ and publishes $u'^{(i)} = g_1^{a'^{(i)}}$, $u_1^{(i)} = g_1^{a_1^{(i)}}, \ldots, u_n^{(i)} = g_1^{a_n^{(i)}}$. Each party is associated with a public hash function

$$H^{(i)}(m) = u'^{(i)} \prod_{j=1}^{n} (u_j^{(i)})^{m_j}.$$

A user may construct a function $H\colon \{0,1\}^n \to G_1$ by computing $u' = \prod_{i=1}^{k} u'^{(i)}$, $u_1 = \prod_{i=1}^{k} u_1^{(i)}, \ldots, u_n = \prod_{i=1}^{k} u_n^{(i)}$, and so

$$H(m) = \prod_{i=1}^{k} H^{(i)}(m).$$

Considering the effectiveness of the attack by a dishonest TTP, it seems that having some party select Y_1, Y_2, u', $u_1, \ldots, u_n$ verifiably at random is more desirable than merely assuming that a TTP will generate the u_i's at random. Unfortunately, in the standard model it is not clear what verifiably at random should mean. In practice, we would use a hash function to determine the u_i's. Certainly, if we assume the hash function is a random oracle, we can prove the scheme secure. But this defeats the purpose of the Waters scheme.

Furthermore, for Type 2 and Type 3 pairings, there is no way to produce a pair Y_1, Y_2 such that $e(Y_1, g_2) = e(g_1, Y_2)$ and one of Y_1, Y_2 is produced using a hash function. Combining the TTP modification with the APK modification, we alleviate this problem. In some implementations, this will be at the expense of larger public keys.

Alternatively, we can make another slight modification to the scheme. During the setup algorithm, instead of producing $Y_1 \in G_1$, $Y_2 \in G_2$, an element $Y \in G_1$ is selected uniformly at random and published. The key generation algorithm selects an integer $x \in \mathbb{Z}_p$ uniformly at random and computes the private key $Z = Y^x$ and the public key $X = g_2^x$. The public key is now an element of G_2 rather than G_1. In practice, this means the public key of each user is larger for Type 2 and Type 3 pairings. To verify a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$ on the message m, we replace Equation (20) with

$$e(\sigma_1, g_2) = e(Y, X) \cdot e(H(m), \sigma_3)$$

and perform verification as usual.

5.2. Small signatures

In previous versions of the Waters scheme, a signature is represented by an element of $G_1 \times G_1 \times G_2$. If we are willing to accept larger public keys, we may obtain smaller signatures, taken from the set $G_1 \times G_1$.

Waters with small signatures (SS-Waters). *We define the functions* `Generate`, `Sign`, *and* `Verify`. *By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Generate` *selects integers x, y, c', c_1, …, $c_n \in \mathbb{Z}_p$ uniformly at random. The algorithm returns the private key $Z = g_1^{xy}$, c', c_1, …, c_n and the public key $\langle X, Y\rangle = \langle g_1^x, g_2^y\rangle$, $v' = g_2^{c'}$, $v_1 = g_2^{c_1}$, …, $v_n = g_2^{c_n}$. For notational convenience we define the functions $H\colon \{0,1\}^n \to G_2$ and $K\colon \{0,1\}^n \to \mathbb{Z}_p$ by*

$$H(m) = H(m_1, \ldots, m_n) = v' \cdot \prod_{i=1}^{n} v_i^{m_i}$$

$$K(m) = K(m_1, \ldots, m_n) = c' + \sum_{i=1}^{n} c_i m_i.$$

It should be noted that since v', v_1, …, v_n are public, the function H is public.

`Sign` *takes as input a message m and a private key Z. The algorithm selects a random integer r and returns the signature*

$$\sigma = \langle Z \cdot (g_1^{rK(m)}, g_1^r\rangle.$$

`Verify` *takes as input a message m, a public key $\mathcal{X}$, and a signature σ. Using $\sigma = \langle \sigma_1, \sigma_2\rangle$, the algorithm accepts the signature iff*

$$e(\sigma_1, g_2) = e(X, Y) \cdot e(\sigma_2, H(m)).$$

The scheme appears to enjoy a level of security equivalent to Waters scheme. However, in general, we cannot expect to reduce the security of one to the other. Specifically, we cannot generate the third component of the Waters signature $\langle X \cdot H(m)^r, g_1^r, g_2^r\rangle$, given only a TTP-SS-Waters signature $\langle X \cdot H(m), g_1^r\rangle$. Conversely, given Waters hash function parameters u', u_1, …, u_n, we cannot generate the v_i's.

The next lemma proves that SS-Waters is secure. We note that when compared to the security reduction in Lemma 16 which proves the Waters scheme secure, the roles of g_1^x and g_1^y in the co-CDH$'$ problem are reversed.

Lemma 20. *Given an SS-Waters $\langle q_S, n, t, \varepsilon\rangle$-forger, we can construct a co-CDH$'$ $\langle t + q_S, \varepsilon\varepsilon'\rangle$-solver which makes one query to the forger where*

$$\varepsilon' = \frac{1}{4q_S(n+1)}.$$

Proof. We begin with an instance of the co-CDH$'$ problem. We are given elements g_1^x, g_1^y, g_2^x and wish to produce g_1^{xy}. We construct an instance of the SS-Waters scheme with challenge key $\langle X, Y\rangle = \langle g_1^y, g_2^x\rangle$. Let $t = 2q_S$ and let $k \in \{0, \ldots, n\}$ be selected uniformly at random. Let a', a_1, …, $a_n \in \{0, \ldots, t-1\}$, b', b_1, …, $b_n \in \mathbb{Z}_p$ be integers chosen uniformly at random. Set $v' = (g_2^x)^{a'} g_2^{b'}$ and for each $i = 1$, …, n, set

$$v_i = (g_2^x)^{a_i} g_2^{b_i}.$$

Define

$$F(m) = a' - tk + \sum_{i=1}^{n} a_i m_i,$$

$$J(m) = b' + \sum_{i=1}^{n} b_i m_i.$$

For a message $m \in \{0,1\}^n$, we have that $H(m) = g_2^{xF(m)+J(m)}$. Now, we run the forger.

To respond to a signature query on the message m, we abort if $F(m) = 0$. Otherwise, we select an integer $\hat{r} \in \mathbb{Z}_p$ uniformly at random and return the signature

$$\sigma = \langle (g_1^y)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}}, g_1^{\hat{r}}(g_1^y)^{-\frac{1}{F(m)}} \rangle.$$

To show that this is a valid signature, we set $r = \hat{r} - \frac{y}{F(m)}$ and so

$$\begin{aligned}
(g_1^y)^{-\frac{J(m)}{F(m)}} g_1^{\hat{r}K(m)} &= (g_1^{xy})(g_1^{-xy})(g_1^y)^{-\frac{J(m)}{F(m)}} g_1^{\hat{r}K(m)} \\
&= g_1^{xy}\big((g_1^{-x})(g_1)^{-\frac{J(m)}{F(m)}}\big)^y g_1^{\hat{r}K(m)} \\
&= g_1^{xy}\big((g_1^{x})^{F(m)} g_1^{J(m)}\big)^{\frac{-y}{F(m)}} g_1^{\hat{r}K(m)} \\
&= g_1^{xy}\big((g_1^{xF(m)+J(m)}\big)^{\frac{-y}{F(m)}} g_1^{\hat{r}K(m)} \\
&= g_1^{xy} H(m)^{\frac{-x}{F(m)}} g_1^{\hat{r}K(m)} = (g_1^{xy}) g_1^{(\hat{r}-\frac{x}{F(m)})K(m)} \\
&= g_1^{xy} g_1^{rK(m)}.
\end{aligned}$$

Eventually, the forger returns a signature $\sigma = \langle \sigma_1, \sigma_2 \rangle = \langle g_1^{xy} g_1^{rK(m)}, g_1^r \rangle$ on the message m. If $F(m) \neq 0$, we abort. Otherwise, we compute

$$\sigma_1 \cdot \sigma_2^{-J(m)} = g_1^{xy}.$$

To compute the reduction's probability of success, we let $m_1, \ldots, m_{q_S}$ be the messages on which signature queries were made and m be the message for which a forgery was returned. The reduction succeeds when $F(m) = 0$ and $F(m_i) \neq 0$, for all $i = 1, \ldots, q_S$. Applying Waters lemma, we have, as required, that

$$\Pr\Big[\big(F(m) = 0\big) \wedge \big(\bigwedge_{i=1}^{q_S} F(m_i) \neq 0\big)\Big] \geq \frac{1}{4q_S(n+1)}.$$

□

Unfortunately, the SS-Waters scheme has a significantly larger public key. For Type 3 pairings, the number of bits required to represent elements of G_2 is twice than the number of bits required to represent elements of G_1. In effect, we double the size of the public key, a key which is already very large.

If we assume the existence of a TTP, we can significantly reduce the size of the public key but at the expense of larger public parameters.

Waters with small signatures and a trusted third-party (TTP-SS-Waters). *We define the functions* `Setup`, `Generate`, `Sign`, *and* `Verify`. *By making* *use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Setup` *selects integers y, c', c_1, ..., $c_n \in \mathbb{Z}_p$ uniformly at random. The algorithm publishes $u' = g_1^{c'}$, $u_1 = g_1^{c_1}$, ..., $u_n = g_1^{c_n}$, $v' = g_2^{c'}$, $v_1 = g_2^{c_1}$, ..., $v_n = g_2^{c_n}$, $Y_1 = g_1^y$, $Y_2 = g_2^y$. For notational convenience we define the functions $H_1\colon \{0,1\}^n \to G_1$, $H_2\colon \{0,1\}^n \to G_2$ by*

$$H_1(m) = H_1(m_1, \ldots, m_n) = u' \cdot \prod_{i=1}^{n} u_i^{m_i},$$
$$H_2(m) = H_2(m_1, \ldots, m_n) = u' \cdot \prod_{i=1}^{n} v_i^{m_i}.$$

It should be noted that since u', u_1, ..., u_n, v', v_1, ..., v_n are public, the functions H_1 and H_2 are public.

`Generate` *selects an integer $x \in \mathbb{Z}_p$ uniformly at random and returns the private key $Z = Y_1^x$ and the public key $X = g_1^x$.*

`Sign` *takes as input a message m and a private key Z. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\sigma = \langle Z \cdot (H_1(m))^r, g_1^r \rangle.$$

`Verify` *takes as input a message m, a public key X, and a signature $\sigma = \langle \sigma_1, \sigma_2 \rangle$. The algorithm must ensure $e(u', g_2) = e(g_1, v')$ and for all $i = 1$, ..., n,*

$$e(u_i, g_2) = e(g_1, v_i). \tag{21}$$

The `Verify` *algorithm accepts the signature iff*

$$e(\sigma_1, g_2) = e(X, Y_2) \cdot e(\sigma_2, H_2(m)),$$

$e(u', g_2) = e(g_1, v')$, and Equation (21) holds for all u_i, v_i's.

The next lemma states that the TTP-SS-Waters scheme enjoys the same level of security as SS-Waters. We omit the proof as it is simply the application of two modifications to the proof of the security of the Waters scheme in Lemma 16. The modifications were applied separately to obtain security proofs of TTP-Waters in Lemma 19 and SS-Waters in Lemma 20.

Lemma 21. *Given a TTP-SS-Waters $\langle q_S, n, t, \varepsilon \rangle$-forger, we can construct a co-CDH′ $\langle t + t', \varepsilon \cdot \varepsilon' \rangle$-solver where*

$$\varepsilon' \geq \frac{1}{4q_S(n+1)}.$$

Of course, we may also modify the SS-Waters and TTP-SS-Waters so that they have public keys in G_T. This will reduce verification time by one pairing evaluation but may be at the cost of (in the case of Type 3 pairings) increasing the size of public keys and public parameters.

For Type 2 and Type 3 pairings, it is not possible to replace the TTP with a party that selects the public parameters verifiably at random. For Type 2 pairings, we are unable to hash into G_2. For Type 3 pairings, it is not known how one may select a pair $(u, v) \in G_1 \times G_2$ such that $e(u, g_2) = e(g_1, v)$ and one of u, v is selected verifiably at random. This restriction holds regardless of whether one makes the random oracle assumption.

5.3. Reducing public key or public parameter size

If we are willing to accept a loss in the success probability of the security reduction, we may dramatically reduce the number of u_i's, the largest contributing factor to size of the public key or public parameters.

Chatterjee-Sarkar [**8**] and Naccache [**26**] independently proposed the modification in the context of the Waters identity-based encryption scheme. The modified scheme is as follows.

The Waters signature scheme with a smaller hash function (SH-Waters). *We define the functions* `Generate`*,* `Sign`*, and* `Verify`*. By making use of a collision-resistant hash function, we assume that all messages are of bit-length $n \cdot \ell$.*

`Generate` *selects integers x, $y \in \mathbb{Z}_p$ uniformly at random and elements u', u_1, $\ldots$, $u_n \in G_1$ uniformly at random. The algorithm returns the private key $Z = g_1^{xy}$ and the public key $\langle X, Y\rangle = \langle g_1^x, g_2^y\rangle$, u', u_1, $\ldots$, u_n. For notational convenience we define the function $H\colon \mathbb{Z}_{2^\ell}^n \to G_1$ by*

$$H(m) = H(m_1, \ldots, m_n) = u' \cdot \prod_{i=1}^{n} u_i^{m_i}.$$

It should be noted that since u', u_1, $\ldots$, u_n are public, the function H is public.

`Sign` *takes as input a message m and a private key Z. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\sigma = \langle Z \cdot (H(m))^r, g_1^r, g_2^r\rangle.$$

`Verify` *takes as input a message m, a public key $\langle X, Y\rangle$, and a signature $\sigma = \langle\sigma_1, \sigma_2, \sigma_3\rangle$. The algorithm checks*

$$e(\sigma_2, g_2) = e(g_1, \sigma_3) \tag{22}$$

and

$$e(\sigma_1, g_2) = e(X, Y) \cdot e(H(m), \sigma_3). \tag{23}$$

The algorithm accepts the signature iff equations (22) *and* (23) *hold.*

A SH-Waters forger which uses a public key u', u_1, $\ldots$, $u_n \in G_1$, uses at most q_S signing queries, and can forge signatures in time at most t with probability at least ε is referred to as an *SH-Waters $\langle q_S, n, \ell, t, \varepsilon\rangle$-forger.*

To prove the SH-Waters secure, we require a strengthened version of Waters lemma.

Waters lemma (strong version). *Let n, ℓ, q_S be integers. Set $t = 2q_S$. Let m, m_1, $\ldots$, $m_{q_S} \in \{0, \ldots, 2^\ell - 1\}^n$ be distinct. Let a', a_1, $\ldots$, $a_n \in \{0, \ldots, t-1\}$ be selected uniformly at random. Let $k \in \{0, \ldots, (n+2)2^\ell - 1\}$ be selected uniformly at random. Define $F\colon \mathbb{Z}_{2^\ell}^n \to \mathbb{Z}$ by*

$$F(\mu) = F(\mu_1, \ldots, \mu_n) = a' - kt + \sum_{i=1}^{n} a_i \mu_i.$$

Then

$$\Pr\Big[\big(F(m)=0\big)\wedge\big(\bigwedge_{i=1}^{q_S}F(m_i)\neq 0\big)\Big]\geq\frac{1}{4q_S(n+2)2^{\ell}}.$$

PROOF. Let $\equiv_t$ denote equivalence modulo t. For an arbitrary $m^*\in\mathbb{Z}_{2^\ell}^n$, we claim the probability that $F(m^*)\equiv_t 0$ is $1/t$. To see this, we fix $s=-kt+\sum_{i=1}^{n}a_i$ and notice that since a' is chosen at random, the probability that $a'\equiv_t -s$ is $1/t$.

Since

$$F(m^*)\leq(n+1)(2^\ell-1)(t-1)-kt<\big((n+2)2^\ell-k\big)t,$$

if $F(m^*)\equiv_t 0$, then the probability that $F(m^*)=0$ is $1/(2^\ell(n+2))$, the probability of choosing the correct k. Therefore,

$$\Pr[F(m^*)=0]=\frac{1}{2^\ell t(n+2)}. \tag{24}$$

For each $i=1,\ldots,q_S$, since m and m_i differ in at least one bit and $F(m)=0$ implies $F(m)\equiv_t 0$, the events $F(m)=0$ and $F(m_i)\equiv_t 0$ are independent. So,

$$\Pr\big[F(m_i)\equiv_t 0\mid F(m)=0\big]=\frac{1}{t}. \tag{25}$$

We conclude,

$$\begin{aligned}
&\Pr\Big[\big(F(m)=0\big)\wedge\big(\bigwedge_{i=1}^{q_S}F(m_i)\neq 0\big)\Big]\\
&\geq\Pr\Big[\big(F(m)=0\big)\wedge\big(\bigwedge_{i=1}^{q_S}F(m_i)\not\equiv_t 0\big)\Big]\\
&=\Pr[F(m)=0]\Pr\Big[\bigwedge_{i=1}^{q_S}F(m_i)\not\equiv_t 0\mid F(m)=0\Big]\\
&=\frac{1}{2^\ell t(n+2)}\Pr\Big[\bigwedge_{i=1}^{q_S}F(m_i)\not\equiv_t 0\mid F(m)=0\Big] && \text{by (24)}\\
&=\frac{1}{2^\ell t(n+2)}\left(1-\Pr\Big[\bigvee_{i=1}^{q_S}F(m_i)\equiv_t 0\mid F(m)=0\Big]\right)\\
&\geq\frac{1}{2^\ell t(n+2)}\left(1-\sum_{i=1}^{q_S}\Pr\big[F(m_i)\equiv_t 0\mid F(m)=0\big]\right)\\
&=\frac{1}{2^\ell t(n+2)}\left(1-\frac{q_S}{t}\right) && \text{by (25)}\\
&=\frac{1}{4\cdot 2^\ell q_S(n+2)}.
\end{aligned}$$

□

In the next lemma, we reduce solving the co-CDH′ problem to forging SH-Waters signatures.

Lemma 22. *Given a SH-Waters $\langle q_S, n, \ell, t, \varepsilon\rangle$-forger, we can construct a co-CDH′ $\langle t+t', \varepsilon \cdot \varepsilon'\rangle$-solver which makes one query to the forger where*

$$\begin{aligned} t' &\leq q_S + n, \\ \varepsilon' &\geq \frac{1}{4 \cdot 2^\ell q_S(n+2)}. \end{aligned}$$

Proof. We begin with an instance of the co-CDH′ problem. We are given g_1^x, g_1^y, and g_2^x and we wish to produce g_1^{xy}.

We define a few terms which will be used to construct an SH-Waters public key. Set $t = 2q_S$ and let $k \in \{0, \ldots, (n+2)2^\ell - 1\}$ be selected uniformly at random. Let a', a_1, …, $a_n \in \{0, \ldots, t-1\}$, b', b_1, …, $b_n \in \mathbb{Z}_p$ be integers selected uniformly at random. For notational convenience, we define the functions F, $J \colon \mathbb{Z}_{2^\ell}^n \to \mathbb{Z}_p$ by $F(m) = -kt + a' + \sum_{i=1}^n a_i m_i$ and $J(m) = b' + \sum_{i=1}^n b_i m_i$. Finally, set $u' = (g_1^y)^{-kt+a'} g_1^{b'}$ and for each $i = 1$, …, n, set $u_i = (g_1^y)^{a_i} g_1^{b_i}$. We observe that

$$H(m) = g_1^{yF(m)+J(m)}.$$

We create an instance of the SH-Waters signature scheme with public key $\langle X, Y\rangle = \langle g_1^y, g_2^x\rangle$, u', u_1, …, u_n and run the forger.

To respond to a signature query on the message m, we abort if $F(m) = 0$. Let $\hat{r} \in \mathbb{Z}_p$ be an integer selected uniformly at random. The signature is computed as

$$\sigma = \langle (g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}}, g_1^{\hat{r}}(g_1^x)^{-\frac{1}{F(m)}}, g_2^{\hat{r}}(g_2^x)^{-\frac{1}{F(m)}}\rangle.$$

To show that this is a valid signature, we set $r = \hat{r} - \frac{x}{F(m)}$ and so

$$\begin{aligned} (g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}} &= (g_1^{xy})(g_1^{-xy})(g_1^x)^{-\frac{J(m)}{F(m)}} H(m)^{\hat{r}} \\ &= g_1^{xy}\left((g_1^{-y})(g_1)^{-\frac{J(m)}{F(m)}}\right)^x H(m)^{\hat{r}} \\ &= g_1^{xy}\left((g_1^y)^{F(m)} g_1^{J(m)}\right)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} \\ &= g_1^{xy}\left((g_1^{yF(m)+J(m)}\right)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} \\ &= g_1^{xy} H(m)^{\frac{-x}{F(m)}} H(m)^{\hat{r}} = (g_1^{xy})H(m)^{\hat{r}-\frac{x}{F(m)}} \\ &= g_1^{xy} H(m)^r. \end{aligned}$$

Eventually, the forger returns the signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle = \langle g_1^{xy} H(m)^r, g_1^r, g_2^r\rangle$ for some m not queried. If $F(m) \neq 0$, we abort. Otherwise, we compute

$$\begin{aligned} \sigma_1(\sigma_2)^{-J(m)} &= \left(g_1^{xy} H(m)^r\right)\left(g_1^r\right)^{-J(m)} = g_1^{xy} g_1^{rJ(m)} g_1^{-rJ(m)} \\ &= g_1^{xy}. \end{aligned}$$

Now, we analyze the probability that the reduction can successfully respond to all signature queries and return a forgery. We assume that the forger makes exactly q_S signing queries. Let m_1, …, m_{q_S} be the input for the queries made to the signing oracle. Let m be the message on which the forger returns a signature.

The reduction succeeds if $F(m) = 0$ and $F(m_i) \neq 0$, for each $i = 1, \ldots, q_S$. Applying Waters lemma (strong version), as required, we obtain

$$\Pr\Big[\big(F(m) = 0\big) \wedge \big(\bigwedge_{i=1}^{q_S} F(m_i) \neq 0\big)\Big] \geq \frac{1}{4q_S(n+2)2^{\ell}}.$$

□

5.3.1. Practical implications. If we choose parameters while considering the tightness of the reductions, at first glance, it seems that the $1/2^{\ell}$ factor in the security reduction of the SH-Waters scheme is a great weakness. Naccache remarked that for $n\ell = 160$, taking $\ell = 32$ is an acceptable loss of probability in the reduction, while obtaining much smaller public keys (or public parameters) which consist of only $n = 5$ elements of G_1 [**26**]. However, we will see that by making our group larger, we can reduce public key size at the cost of signature size, while maintaining the same guarantee of security as the Waters scheme. Chatterjee and Sarkar give a more detailed analysis in [**8**].

In practice, if we take $n\ell = 256 = 2^8$, we obtain 128-bits of security. We compare two specific options for ℓ. If we take $\ell = 1$, then this is simply the Waters scheme. The reduction from co-CDH′ to Waters suceeds with probability

$$\frac{1}{4q_S n} = \frac{1}{2^{10}q_S}.$$

If we take $\ell = 128$ and $n = 2$, then we obtain a reduction which succeeds with probability

$$\frac{1}{4q_S 2^{\ell} n} = \frac{1}{2^{131}q_S}.$$

We fix the group G_1 used in the $\ell = 1$ case (which gives us, say, 128-bits of security). To obtain the same level of security as the $\ell = 1$ case in the $\ell = 128$ case, we need an additional 121-bits of security. To obtain this level of security, we can use a group of size $|G_1|^2$. This has an obvious negative impact on signature sizes, doubling the size of signatures. However, the factor by which space required to store the u_i's decreases is very dramatic, when we compare $\ell = 1$ to $\ell = 128$. We require $257 \log |G_1|$-bits to store the u_i's, for $\ell = 1$. For $\ell = 128$, there are only three u_i's, but each is represented by twice as many bits. We require $3 \log |G_1|^2 = 6 \log |G_1|$-bits to store the u_i's in the $\ell = 128$ case.

In summary, we can reduce the space requirements for the u_i's by a factor of around 40 at the expense of doubling the signature size and increasing computational cost, all the while maintaining an equivalent level of security.

5.4. Point compression

Just as in the BLS scheme, we may compress points on an elliptic curve. We could store only the x-coordinate and deal with the ambiguity. For a point $Q = (x, y)$, we let $\widetilde{\cdot}\colon E - \{\infty\} \to \mathbb{F}$ be the operation defined by $\widetilde{Q} = x$. For each element h of $\mathbb{F}$ that is the x-coordinate of a point in $E(\mathbb{F})$, we fix a point P_h such that $h = \widetilde{P}_h$.

Once again, since we are dealing with elliptic curves instead of an arbitrary group, we use additive notation. Let P_1, P_2 be the generators of the elliptic curve groups G_1, G_2 respectively. We use the notation $\widetilde{\cdot}$ for both groups G_1 and G_2.

We present the Waters scheme using compressed points.

The Waters signature scheme with compression (C-Waters). *We define the functions* `Generate`*,* `Sign`*, and* `Verify`*. By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Generate` *selects integers x, $y \in \mathbb{Z}_p$ uniformly at random. Also, we select elements u', $u_1, \ldots, u_n \in G_1$ uniformly at random. Set $X = xP_1$, $Y = yP_2$. The algorithm determines b_x, $b_y \in \{-1, 1\}$ such that $P_{\widetilde{X}} = b_x X$, $P_{\widetilde{Y}} = b_y Y$. The algorithm returns the private key $b_x b_y Z$ and the public key $\widetilde{X}$, $\widetilde{Y}$, $\widetilde{u}'$, $\widetilde{u}_1, \ldots, \widetilde{u}_n$. For notational convenience we define the function $H\colon \{0,1\}^n \to G_1$ by*

$$H(m) = P_{\widetilde{u}'} + \sum_{i=1}^{n} m_i P_{\widetilde{u}_i}.$$

Since $\widetilde{u}'$ and the $\widetilde{u}_i$'s are public, so is the function H.

`Sign` *takes as input a message m and a private key Z. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and computes $S = Z + rH(m)$, $R_1 = rP_1$, and $R_2 = rP_2$. Finally, the algorithm returns the signature $\sigma = \langle \widetilde{S}, \widetilde{R}_1, \widetilde{R}_2 \rangle$.*

`Verify` *takes as input a message m, a public key $\langle \widetilde{X}, \widetilde{Y} \rangle$, and a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$. The algorithm checks*

$$\begin{aligned} e(P_{\sigma_2}, P_2) &= e(P_1, P_{\sigma_3}), \\ e(P_{\sigma_2}, P_2) &= e(P_1, P_{\sigma_3})^{-1}. \end{aligned}$$

If either of the above two signatures hold and any of the following four equations holds, the algorithm accepts the signature:

$$\begin{aligned} e(P_{\sigma_1}, P_2) &= e(P_{\widetilde{X}}, P_{\widetilde{Y}}) \cdot e(H(m), P_{\sigma_3}), \\ e(P_{\sigma_1}, P_2)^{-1} &= e(P_{\widetilde{X}}, P_{\widetilde{Y}}) \cdot e(H(m), P_{\sigma_3}), \\ e(P_{\sigma_1}, P_2) &= e(P_{\widetilde{X}}, P_{\widetilde{Y}}) \cdot e(H(m), P_{\sigma_3})^{-1}, \\ e(P_{\sigma_1}, P_2)^{-1} &= e(P_{\widetilde{X}}, P_{\widetilde{Y}}) \cdot e(H(m), P_{\sigma_3})^{-1}. \end{aligned}$$

In the next lemma, we show that an algorithm which forges C-Waters signatures can be used to forge Waters signatures.

Lemma 23. *Given a C-Waters $\langle q_S, n, t, \varepsilon \rangle$-forger, we can construct a Waters $\langle q_S, n, t, \varepsilon \rangle$-forger which makes one call to the C-Waters forger.*

Proof. We begin with an instance of the Waters scheme with challenge public-key X, Y, u', $u_1, \ldots, u_n$. We construct an instance of the C-Waters signature scheme with challenge public-key $\widetilde{X}$, $\widetilde{Y}$, $\widetilde{u}'$, $\widetilde{u}_1, \ldots, \widetilde{u}_n$. We run the C-Waters forger on this instance of the C-Waters scheme.

To respond to a signature query on the message m, we make a signature query to our Waters oracle to obtain $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$. We respond with the signature $\langle \widetilde{\sigma}_1, \widetilde{\sigma}_2, \widetilde{\sigma}_3 \rangle$.

The C-Waters forger eventually returns a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$. There exist a, b, $c \in \{-1, 1\}$ such that

$$\begin{aligned} e(P_{\sigma_2}, P_2) &= e(P_1, P_{\sigma_3})^a, \\ e(P_{\sigma_1}, P_2)^b &= e(P_{\widetilde{X}}, P_{\widetilde{Y}}) \cdot e(H(m), P_{\sigma_3})^c. \end{aligned}$$

We return the signature $\langle bP_{\sigma_1}, acP_{\sigma_2}, cP_{\sigma_3} \rangle$, a forgery of the Waters scheme. $\square$

5.5. Summary of the variations of the Waters signature scheme

There are several degrees of freedom in selecting a variation of the Waters signature scheme. Initially, we presented the Waters signature scheme. We then presented a few small variations of the scheme. The variations are APK-Waters, TTP-Waters, SS-Waters, SH-Waters, and C-Waters, representing alternative public-key representation, trusted third-party, small signature, small hash, and compression modifications respectively. We can chose any subset of these modifications to obtain a variation of the Waters signature scheme. Indeed, we could choose to use "APK-TTP-SS-SH-C-Waters".

CHAPTER 6

The Lu-Ostrovsky-Sahai-Shacham-Waters Aggregate Signature Scheme

6.1. The scheme

Motivated by the Waters signature scheme, Lu, Ostrovsky, Sahai, Shacham, and Waters (LOSSW) [**24**] developed an aggregate signature scheme whose security does not rely on the random oracle assumption. The scheme is a *sequential aggregate signature scheme*; each user sequentially aggregates his contribution into an aggregate signature.

LOSSW. *We define the functions* `Generate`*,* `Sign-Aggregate`*, and* `Verify`*. By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Generate` *selects integers x, y, c', c_1, …, $c_n \in \mathbb{Z}_p$ uniformly at random. The algorithm returns the public key $X = g_1^x$, $Y = g_2^y$, $u' = g_1^{c'}$, $u_1 = g_1^{c_1}$, …, $u_n = g_1^{c_n}$ and the private key $Z = g_1^{xy}$, c', c_1, …, c_n. For notational convenience, we define the functions $H\colon \{0,1\}^n \to G_1$ and $K\colon \{0,1\}^n \to \mathbb{Z}_p$ by*

$$H(m) = u' \prod_{i=1}^{n} u_i^{m_i};$$

$$K(m) = c' + \sum_{i=1}^{n} c_i m_i.$$

By definition, we have $H(m) = g_1^{K(m)}$. Notice that since the u_i's are public, the function H is public. We use $\langle X, Y, H\rangle$ to refer to the public key of a user and $\langle Z, K\rangle$ to refer to the private key.

`Sign-Aggregate` *takes as input a message m, a private key $\langle Z, K\rangle$, an aggregate signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$ on messages $m^{(1)}$, …, $m^{(N)}$ corresponding to the public keys $\langle X_1, Y_1, H_1\rangle$, …, $\langle X_N, Y_N, H_N\rangle$. If the user is the first signer, then $N = 0$ and we set $\sigma = \langle g_1^0, g_1^0, g_2^0\rangle$. First, the algorithm verifies that the signature σ is valid. Finally, the algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\langle \sigma_1 \cdot Z \cdot (\sigma_2 \cdot g_1^r)^{K(m)} \cdot \prod_{i=1}^{N} H_i(m^{(i)})^r, \sigma_2 \cdot g_1^r, \sigma_3 \cdot g_2^r\rangle.$$

`Verify` *takes as input an aggregate signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$ on messages $m^{(1)}$, …, $m^{(N)}$ corresponding to the public keys $\langle X_1, Y_1, H_1\rangle$, …, $\langle X_N, Y_N, H_N\rangle$. The* *algorithm accepts the signature iff the following two equations hold*

$$e(\sigma_1, g_2) = e(\prod_{i=1}^{N} H_i(m^{(i)}), \sigma_3) \cdot \prod_{i=1}^{N} e(X_i, Y_i);$$
$$e(\sigma_2, g_2) = e(g_1, \sigma_3).$$

Observe that the `Generate` algorithm and for $N = 1$, the `Verify` algorithm are identical to those of the Waters scheme. Also, for the first signer, the `Sign-Aggregate` algorithm is identical to the Waters `Sign` algorithm.

The scheme is secure in the *registered-key model.* An adversary is given a challenge public-key $\langle X_1, Y_1, H_1 \rangle$ and access to a signing oracle for this key. The adversary must output a signature σ on messages $m^{(1)}, \ldots, m^{(N)}$ on public keys $\langle X_1, Y_1, H_1 \rangle, \ldots, \langle X_N, Y_N, H_N \rangle$ such that $m^{(1)}$ was not given as input to the signing oracle. Additionally, the adversary must reveal the private keys corresponding to $\langle X_2, Y_2, H_2 \rangle, \ldots, \langle X_N, Y_N, H_N \rangle$. An adversary which succeeds in time at most t with probability at least ε making at most q_S signing queries is referred to as an *LOSSW* $\langle q_S, n, N, t, \varepsilon \rangle$*-forger.*

This model differs from that used in the BGLS scheme; in the registered-key model, we assume the existence of a trusted third-party which possesses all private keys.

In this security model we may reduce an LOSSW forger to a Waters forger.

Lemma 24. *Given an LOSSW* $\langle q_S, n, N, t, \varepsilon \rangle$*-forger, we can construct a Waters* $\langle q_S, n, t + q_S + N, \varepsilon \rangle$*-forger which makes a single call to the LOSSW forger.*

Proof. We begin with an instance of the Waters signature scheme. We are given a challenge public-key $\langle X, Y, H \rangle$ and access to a Waters signing oracle. Our goal is to produce a Waters signature on some message not queried to the signing oracle. We construct an instance of the LOSSW signature scheme with challenge public-key $\langle X_1, Y_1, H_1 \rangle$ and run the LOSSW forger.

To respond to the LOSSW forger signature query on message m, we use the Waters signing oracle. As we have observed, the Waters signature is an LOSSW signature with a single signer.

Eventually, the LOSSW forger returns the signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$ on the messages $m^{(1)}, \ldots, m^{(N)}$ corresponding to public keys $\langle X_1, Y_1, H_1 \rangle, \ldots, \langle X_N, Y_N, H_N \rangle$. Since we are working in the registered-key model, we have the private keys $\langle Z_2, K_2 \rangle, \ldots, \langle Z_N, K_N \rangle$ corresponding to the public keys $\langle X_2, Y_2, H_2 \rangle, \ldots, \langle X_N, Y_N, H_N \rangle$. We compute

$$\begin{aligned} \sigma_1' &= \sigma_1 \cdot \left(\prod_{i=2}^{N} Z_i \cdot (\sigma_2)^{K(m_i)} \right)^{-1} \\ &= \left(\prod_{i=1}^{N} Z_i \cdot (\sigma_2)^{K(m_i)} \right) \cdot \left(\prod_{i=2}^{N} Z_i \cdot (\sigma_2)^{K(m_i)} \right)^{-1} \\ &= Z_1 \cdot (\sigma_2)^{K(m_1)}. \end{aligned}$$

We return the signature $\sigma = \langle \sigma_1', \sigma_2, \sigma_3 \rangle$, a valid Waters signature. □

Since we obtained in Lemma 16 a reduction from the Waters scheme to the co-CDH′ problem, we conclude that if co-CDH′ is intractable, then the LOSSW scheme is secure under the registered-key model.

6.1.1. Smaller public keys with a TTP. Since we are already assuming the existence of a certifying authority which will possess all private keys, this certifying authority is a Trusted Third-Party (TTP). As we have seen in the Waters scheme, if the TTP publishes some global parameters, the size of our public key can be reduced. In addition, we may reduce the number of pairing computations required to verify signatures. The size of the public key remains very large due to the u_i's. We present the modified scheme.

LOSSW with a TTP (TTP-LOSSW). *We define the functions* `Setup`, `Generate`, `Sign-Aggregate`, *and* `Verify`. *By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Setup` *selects an integer $y \in \mathbb{Z}_p$ uniformly at random and publishes $\langle Y_1, Y_2 \rangle = \langle g_1^y, g_2^y \rangle$. If the certifying authority issues private and public keys, the element Y_1 need not be made public. Making Y_1 public allows users to compute their keys offline. It should be noted that users need to register their keys with the certifying authority anyway.*

`Generate` *selects integers x, c', c_1, …, $c_n \in \mathbb{Z}_p$ uniformly at random. The algorithm returns the public key $X = g_1^x$, $u' = g_1^{c'}$, $u_1 = g_1^{c_1}$, …, $u_n = g_1^{c_n}$ and the private key $Z = Y_1^x$, c', c_1, …, c_n. For notational convenience, we define the functions $H \colon \{0,1\}^n \to G_1$ and $K \colon \{0,1\}^n \to \mathbb{Z}_p$ by*

$$H(m) = u' \prod_{i=1}^{n} u_i^{m_i};$$
$$K(m) = c' + \sum_{i=1}^{n} c_i m_i.$$

By definition, we have $H(m) = g_1^{K(m)}$. Notice that since u' and the u_i's are public, the function H is public. We use $\langle X, H \rangle$ to refer the public key of a user and $\langle Z, K \rangle$ to refer the private key.

`Sign-Aggregate` *takes as input a message m, a private key $\langle Z, K \rangle$, and optionally an aggregate signature σ on messages $m^{(1)}$, …, $m^{(N)}$ corresponding to the public keys $\langle X_1, H_1 \rangle$, …, $\langle X_N, H_N \rangle$. If the optional signature is not part of the input, the algorithm sets $\sigma = \langle g_1^0, g_1^0, g_2^0 \rangle$. Using $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$, the algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\langle \sigma_1 \cdot Z \cdot (\sigma_2 \cdot g_1^r)^{K(m)} \cdot \prod_{i=1}^{N} H_i(m^{(i)})^r, \sigma_2 \cdot g_1^r, \sigma_3 \cdot g_2^r \rangle.$$

`Verify` *takes as input an aggregate signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$ on messages $m^{(1)}$, …, $m^{(N)}$ corresponding to the public keys $\langle X_1, H_1 \rangle$, …, $\langle X_N, H_N \rangle$. The algorithm* *accepts the signature iff the following two equations hold*

$$\begin{aligned} e(\sigma_1, g_2) &= e(\prod_{i=1}^{N} H_i(m^{(i)}), \sigma_3) \cdot e(\prod_{i=1}^{N} X_i, Y_2); \\ e(\sigma_2, g_2) &= e(g_1, \sigma_3). \end{aligned}$$

Lemma 25. *Given a TTP-LOSSW $\langle q_S, n, N, t, \varepsilon\rangle$-forger, we can construct Waters $\langle q_S, n, t + q_S + N, \varepsilon\rangle$-forger which makes a single call to the TTP-LOSSW forger.*

We omit the proof as it is essentially the same as Lemma 24.

6.2. Small signatures

Signatures in the LOSSW scheme are elements of $G_1 \times G_1 \times G_2$. In this section, we modify the scheme so that signatures are in $G_1 \times G_1$. However, this is done at the cost of having a much larger public key.

LOSSW with small signatures (SS-LOSSW). *We define the functions* `Generate`, `Sign-Aggregate`*, and* `Verify`*. By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Generate` *selects integers x, y, c', c_1, …, $c_n \in \mathbb{Z}_p$ uniformly at random. The algorithm returns the public key $X = g_1^x$, $Y = g_2^y$, $u' = g_1^{c'}$, $u_1 = g_1^{c_1}$, …, $u_n = g_1^{c_n}$, $v' = g_2^{c'}$, $v_1 = g_2^{c_1}$, …, $v_n = g_2^{c_n}$ and the private key $Z = g_1^{xy}$, c', c_1, …, c_n. For notation convenience, we define the functions $H^{(1)} \colon \{0,1\}^n \to G_1$, $H^{(2)} \colon \{0,1\}^n \to G_2$, and $K \colon \{0,1\}^n \to \mathbb{Z}_p$ by*

$$\begin{aligned} H^{(1)}(m) &= u' \prod_{i=1}^{n} u_i^{m_i}; \\ H^{(2)}(m) &= v' \prod_{i=1}^{n} v_i^{m_i}; \\ K(m) &= c' + \sum_{i=1}^{n} c_i m_i. \end{aligned}$$

By definition, we have that $H^{(1)}(m) = g_1^{K(m)}$ and $H^{(2)}(m) = g_2^{K(m)}$. Notice that since the u_i's, v_i's are public, the functions $H^{(1)}$, $H^{(2)}$ are public. Define $\boldsymbol{U} = \langle u', u_1, \ldots, u_n\rangle$, $\boldsymbol{V} = \langle v', v_1, \ldots, v_n\rangle$. We use $\langle X, Y, \boldsymbol{U}, \boldsymbol{V}\rangle$ to refer to the public key of a user and $\langle Z, K\rangle$ to refer to the private key.

`Sign-Aggregate` *takes as input a message m, a private key $\langle Z, K\rangle$, and optionally an aggregate signature $\sigma = \langle \sigma_1, \sigma_2\rangle$ on messages $m^{(1)}$, …, $m^{(N)}$ corresponding to the public keys $\langle X_1, Y_1, \boldsymbol{U}^{(1)}, \boldsymbol{V}^{(1)}\rangle$, …, $\langle X_N, Y_N, \boldsymbol{U}^{(N)}, \boldsymbol{V}^{(N)}\rangle$. If the optional signature is not part of the input, the algorithm sets $\sigma = \langle g_1^0, g_1^0\rangle$. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\langle \sigma_1 \cdot Z \cdot (\sigma_2 \cdot g_1^r)^{K(m)} \cdot \prod_{i=1}^{N} H_i^{(1)}(m^{(i)})^r, \sigma_2 \cdot g_1^r\rangle.$$

`Verify` *takes as input an aggregate signature σ on messages $m^{(1)}$, …, $m^{(N)}$ corresponding to the public keys $\langle X_1, Y_1, \boldsymbol{U}^{(1)}, \boldsymbol{V}^{(1)}\rangle$, …, $\langle X_N, Y_N, \boldsymbol{U}^{(N)}, \boldsymbol{V}^{(N)}\rangle$. For* *each user $i = 1, \ldots, N$, let*

$$\boldsymbol{U}^{(i)} = \langle u'^{(i)}, u_1^{(i)}, \ldots, u_n^{(i)} \rangle,$$
$$\boldsymbol{V}^{(i)} = \langle v'^{(i)}, v_1^{(i)}, \ldots, v_n^{(i)} \rangle.$$

To verify the integrity of the public key, the algorithm checks

$$e(u'^{(i)}, g_2) = e(g_1, v'^{(i)}) \tag{26}$$

and for each $j = 1, \ldots, n$, the algorithm checks

$$e(u_j^{(i)}, g_2) = e(g_1, v_j^{(i)}). \tag{27}$$

The signature is accepted iff for each $i = 1, \ldots, N$, Equation (26) *holds, for each $i = 1, \ldots, N$ and each $j = 1, \ldots, n$, Equation* (27) *holds, and using $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$ the following equation holds:*

$$e(\sigma_1, g_2) = e(\sigma_2, \prod_{i=1}^{N} H_i^{(2)}(m^{(i)})) \cdot \prod_{i=1}^{N} e(X_i, Y_i).$$

We notice that expensive checks of Equation (26) and Equation (27) can be done by the certifying authority before they key is certified.

We obtain the following proof of security for the SS-LOSSW scheme.

Lemma 26. *Given a SS-LOSSW $\langle q_S, n, N, t, \varepsilon \rangle$-forger, we may construct a Waters $\langle q_S, n, t + q_S + N, \varepsilon \rangle$-forger which makes one query to the SS-LOSSW forger.*

We omit the proof, as it is similar to that of Lemma 24.

6.3. Reducing public key size

Using the techniques of Chatterjee-Sarkar [**8**] and Naccache [**26**], we may reduce the size of public keys. We prove the scheme secure relative to the forging of SH-Waters signatures. The security of the SH-Waters scheme relative to the co-CDH′ problem decreases exponentially as ℓ increases.

LOSSW with smaller hash function (SH-LOSSW). *We define the functions* `Generate`, `Sign-Aggregate`, *and* `Verify`. *By making use of a collision-resistant hash function, we assume that all messages are of bit-length $n \cdot \ell$.*

`Generate` *selects integers x, y, c', $c_1, \ldots, c_n \in \mathbb{Z}_p$ uniformly at random. The algorithm returns the public key $X = g_1^x$, $Y = g_2^y$, $u' = g_1^{c'}$, $u_1 = g_1^{c_1}, \ldots, u_n = g_1^{c_n}$ and the private key $Z = g_1^{xy}$, c', $c_1, \ldots, c_n$. For notational convenience, we define the functions $H \colon \mathbb{Z}_{2^\ell}^n \to G_1$ and $K \colon \mathbb{Z}_{2^\ell}^n \to \mathbb{Z}_p$ by*

$$H(m) = u' \prod_{i=1}^{n} u_i^{m_i};$$

$$K(m) = c' + \sum_{i=1}^{n} c_i m_i.$$

By definition, we have $H(m) = g_1^{K(m)}$. Notice that since the u_i's are public, the function H is public. We use $\langle X, Y, H \rangle$ to refer to the public key of a user and $\langle Z, K \rangle$ to refer to the private key.

`Sign-Aggregate` *takes as input a message m, a private key $\langle Z, K\rangle$, and optionally an aggregate signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$ on messages $m^{(1)}, \ldots, m^{(N)}$ corresponding to the public keys $\langle X_1, Y_1, H_1\rangle, \ldots, \langle X_N, Y_N, H_N\rangle$. If the optional signature is not part of the input, the algorithm sets $\sigma = \langle g_1^0, g_1^0, g_2^0\rangle$. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature*

$$\langle \sigma_1 \cdot Z \cdot (\sigma_2 \cdot g_1^r)^{K(m)} \cdot \prod_{i=1}^{N} H_i(m^{(i)})^r, \sigma_2 \cdot g_1^r, \sigma_3 \cdot g_2^r\rangle.$$

`Verify` *takes as input an aggregate signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$ on messages $m^{(1)}, \ldots, m^{(N)}$ corresponding to the public keys $\langle X_1, Y_1, H_1\rangle, \ldots, \langle X_N, Y_N, H_N\rangle$. The algorithm accepts the signature iff the following two equations hold*

$$\begin{aligned} e(\sigma_1, g_2) &= e(\prod_{i=1}^{N} H_i(m^{(i)}), \sigma_3) \cdot \prod_{i=1}^{N} e(X_i, Y_i); \\ e(\sigma_2, g_2) &= e(g_1, \sigma_3). \end{aligned}$$

In the following lemma, we reduce forging SH-Waters signatures to forging SH-LOSSW signatures.

Lemma 27. *Given a SH-LOSSW $\langle q_S, n, \ell, N, t, \varepsilon\rangle$-forger, we can construct a SH-Waters $\langle q_S, n, \ell, t, \varepsilon\rangle$-forger which makes one query to the SS-LOSSW forger.*

Once again, the proof of Lemma 27 is omitted as the proof is similar to that of Lemma 24.

6.4. Point compression

We present a modification of the LOSSW scheme similar to that used in Chapter 5 to compress elliptic curve points. Let $\widetilde{\cdot}\colon E - \{\infty\} \to \mathbb{F}$ be such that for a point Q of E, the element $h = \widetilde{Q}$ is the x-coordinate of Q. For each element h of $\mathbb{F}$ we fix some P_h in E such that $h = \widetilde{P}_h$.

In the remainder of this section, we fix elliptic curves G_1 and G_2 with generators P_1 and P_2. We switch from multiplicative notation to additive notation. We use the map $\widetilde{\cdot}$ for both G_1 and G_2.

LOSSW with compression (C-LOSSW). *We define the functions* `Generate`, `Sign-Aggregate`, *and* `Verify`. *By making use of a collision-resistant hash function, we assume that all messages are of bit-length n.*

`Generate` *selects integers x, y, c', $c_1, \ldots, c_n \in \mathbb{Z}_p$ uniformly at random. Set $X = xP_1$, $Y = yP_2$, $u' = c'P_1$, $u_1 = c_1P_1, \ldots, u_n = c_nP_1$. Let b_x, b_y, b', $b_1, \ldots, b_n \in \{-1, 1\}$ be such that $P_{\widetilde{X}} = b_xX$, $P_{\widetilde{Y}} = b_yY$, $P_{\widetilde{u'}} = b'u'$, $P_{\widetilde{u}_1} = b_1u_1, \ldots, P_{\widetilde{u}_n} = b_nu_n$. The algorithm returns the public key $\widetilde{X}$, $\widetilde{Y}$, $\widetilde{u}'$, $\widetilde{u}_1, \ldots, \widetilde{u}_n$ and the private key $Z = b_xb_yxyP_1$, $b'c'$, $b_1c_1, \ldots, b_nc_n$. For notational convenience, we* *define the functions $H\colon \{0,1\}^n \to G_1$ and $K\colon \{0,1\}^n \to \mathbb{Z}_p$ by*

$$H(m) = P_{\widetilde{u}'} + \sum_{i=1}^{n} P_{\widetilde{u}_i} m_i;$$

$$K(m) = c' + \sum_{i=1}^{n} c_i m_i.$$

By definition, we have $H(m) = K(m)P_1$. Notice that since $\widetilde{u}'$ and the $\widetilde{u}_i$'s are public, the function H is public. We use $\langle \widetilde{X}, \widetilde{Y}, H\rangle$ to refer to the public key of a user and $\langle Z, K\rangle$ to refer to the private key.

`Sign-Aggregate` *takes as input a message m and a private key $\langle Z, K\rangle$. Optionally the algorithm takes as input an aggregate signature σ on messages $m^{(1)}$, $\ldots$, $m^{(N)}$ corresponding to the public keys $\langle \widetilde{X}_1, \widetilde{Y}_1, H_1\rangle$, $\ldots$, $\langle \widetilde{X}_N, \widetilde{Y}_N, H_N\rangle$. If the optional signature is not part of the input, the algorithm sets $\sigma = \langle \widetilde{\infty}, \widetilde{\infty}, \widetilde{\infty}\rangle$ and $N = 0$. Using $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$, the algorithm determines a, b, $c \in \{-1, 1\}$ such that*

$$e(P_{\sigma_2}, P_2) = e(P_1, P_{\sigma_3})^a,$$
$$e(P_{\sigma_1}, P_2)^b = e(P_{\widetilde{X}}, P_{\widetilde{Y}}) \cdot e(H(m), P_{\sigma_3})^c.$$

Let $S_1 = bP_{\sigma_1}$, $S_2 = acP_{\sigma_2}$, and $S_3 = cP_{\sigma_3}$. The algorithm selects an integer $r \in \mathbb{Z}_p$ uniformly at random and returns the signature

$$\langle S_1 + Z + K(m)(S_2 + rP_1) + \sum_{i=1}^{N} rH_i(m^{(i)}), S_2 + rP_1, S_3 + rP_2\rangle.$$

`Verify` *takes as input an aggregate signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle$ on messages $m^{(1)}$, $\ldots$, $m^{(N)}$ corresponding to the public keys $\langle \widetilde{X}_1, \widetilde{Y}_1, H_1\rangle$, $\ldots$, $\langle \widetilde{X}_N, \widetilde{Y}_N, H_N\rangle$. The algorithm accepts the signature iff one of the following two equations hold*

$$e(P_{\sigma_2}, P_2) = e(P_1, P_{\sigma_3}),$$
$$e(P_{\sigma_2}, P_2) = e(P_1, P_{\sigma_3})^{-1},$$

and one of the following equations hold

$$e(P_{\sigma_1}, P_2) = e(\sum_{i=1}^{N} H_i(m^{(i)}), P_{\sigma_3}) \cdot \prod_{i=1}^{N} e(P_{\widetilde{X}_i}, P_{\widetilde{Y}_i}),$$
$$e(P_{\sigma_1}, P_2)^{-1} = e(\sum_{i=1}^{N} H_i(m^{(i)}), P_{\sigma_3}) \cdot \prod_{i=1}^{N} e(P_{\widetilde{X}_i}, P_{\widetilde{Y}_i}),$$
$$e(P_{\sigma_1}, P_2) = e(\sum_{i=1}^{N} H_i(m^{(i)}), P_{\sigma_3})^{-1} \cdot \prod_{i=1}^{N} e(P_{\widetilde{X}_i}, P_{\widetilde{Y}_i}),$$
$$e(P_{\sigma_1}, P_2)^{-1} = e(\sum_{i=1}^{N} H_i(m^{(i)}), P_{\sigma_3})^{-1} \cdot \prod_{i=1}^{N} e(P_{\widetilde{X}_i}, P_{\widetilde{Y}_i}).$$

Lemma 28. *Given a C-LOSSW $\langle q_S, n, N, t, \varepsilon\rangle$-forger, we may construct a Waters $\langle q_S, n, t + q_S + N, \varepsilon\rangle$ which makes one query to the C-LOSSW forger.*

Since the proof of Lemma 28 is similar to that of Lemma 24, we omit the proof.

CHAPTER 7

Coron's Theorem

7.1. Coron's Lemma

In Chapters 3 and 5, we have proven that given a BLS forger or a Waters forger, we can construct a co-CDH′ solver. As we noted in the respective chapters, the co-CDH′ solver we produce using these forgers succeeds with a lower probability than the forgers we were originally given. In this chapter, we will show that in the case of BLS and Waters, we cannot hope to find a reduction which succeeds with probability non-negligibly greater than $1/(q_S + 1)$, where q_S is the number of signing queries allowed.

Coron proved an analogous result for the Full-Domain Hash signature scheme and observed that the same technique can be applied to any deterministic signature scheme including BLS [**10**]. Coron remarked that the proof does not apply to non-deterministic signature schemes. In this Chapter, we reproduce Coron's results for deterministic signature schemes. We also show that with a slight modification of the proof, we can obtain a similar result for a class of non-deterministic signature schemes which includes the Waters scheme.

Before we prove Coron's result, we need to define a few terms. Let n be a positive integer and let M be a finite set. A collection

$$Q \subseteq \bigcup_{i=0}^{n} M^i$$

of subsequences of M^n is said to be *prefix-closed* if

$$(\gamma_1, \dots, \gamma_k) \in Q \implies (\gamma_1, \dots, \gamma_{k-1}) \in Q, \qquad \text{for all } k > 0.$$

Coron's lemma. *Let Q be prefix-closed collection of M^n subsequences, for some positive integer n and some finite set M. Let α_1, …, α_n, β be distinct elements selected uniformly at random from M. Let i be an integer selected uniformly at random from $[1, n]$. For each $j = 0$, …, n, let A_j be the event $(\alpha_1, \dots, \alpha_j) \in Q$ and B_j be the event $(\alpha_1, \dots, \alpha_{j-1}, \beta) \in Q$. Then*

$$\Pr[A_n \wedge \neg B_i] \le \frac{1}{n+1}.$$

Proof. We will show by induction on n that

$$\Pr[A_n \wedge B_i] \ge \Pr[A_n]^{1+\frac{1}{n}}.$$

If $n = 1$, then since A_1 and B_1 are independent events,

$$\Pr[A_1 \wedge B_1] = \Pr[A_1] \Pr[B_1] = \Pr[A_1]^2 = \Pr[A_n]^{1+\frac{1}{n}}.$$

Therefore, the statement holds for $n = 1$ and for all M.

Assume $n > 1$ and that if $\hat{Q}$ is a prefix-closed collection of $\hat{M}^{n-1}$ subsequences, for some $\hat{M}$, then the statement holds for $\hat{Q}$.

Let Q be a prefixed closed collection of M^n subsequences for some M. We make a few observations. Since, A_n and B_1 are independent events,

$$\Pr[A_n \wedge B_1] = \Pr[A_n]\Pr[B_1] = \Pr[A_n]\Pr[A_1] \geq \Pr[A_n]^2. \tag{28}$$

Define

$$\hat{Q} = \{(\gamma_2, \ldots, \gamma_k) \mid (\alpha_1, \gamma_2, \ldots, \gamma_k) \in Q\}.$$

For $j = 1, \ldots, n-1$, set $\hat{\alpha}_j = \alpha_{j+1}$. We also set $\hat{M} = M - \{\alpha_1\}$. Finally, for $j = 1, \ldots, n-1$, we let $\hat{A}_j$ denote the event $(\hat{\alpha}_1, \ldots, \hat{\alpha}_j) \in \hat{Q}$ and $\hat{B}_j$ denote the event $(\hat{\alpha}_1, \ldots, \hat{\alpha}_{j-1}, \beta) \in \hat{Q}$. Notice that if $i \geq 2$, then $i-1$ is selected uniformly at random from $[1, n-1]$. Since $\hat{Q}$ is a prefix-closed collection of $\hat{M}^{n-1}$ subsequences, we have that

$$\Pr[\hat{A}_{n-1} \wedge \hat{B}_{i-1} \mid i \geq 2] \geq \Pr[\hat{A}_{n-1} \mid i \geq 2]^{1+\frac{1}{n-1}} = \Pr[\hat{A}_{n-1}]^{1+\frac{1}{n-1}}.$$

By definition,

$$\Pr[A_n \wedge B_i \mid i \geq 2] = \Pr[\hat{A}_{n-1} \wedge \hat{B}_{i-1}] \geq \Pr[\hat{A}_{n-1}]^{1+\frac{1}{n-1}} = \Pr[A_n]^{1+\frac{1}{n-1}}. \tag{29}$$

Next, recall the following relationship between the arithmetic and geometric mean. For any real number x,

$$\frac{1}{n}x + \frac{n-1}{n}x^{\frac{1}{n-1}} \geq x^{\frac{1}{n}}. \tag{30}$$

We have that

$$\begin{aligned}
\Pr[A_n \wedge B_i] &= \Pr[A_n \wedge B_i \mid i = 1]\Pr[i = 1] + \Pr[A_n \wedge B_i \mid i \geq 2]\Pr[i \geq 2] \\
&= \frac{1}{n}\Pr[A_n \wedge B_i \mid i = 1] + \frac{n-1}{n}\Pr[A_n \wedge B_i \mid i \geq 2] \\
&= \frac{1}{n}\Pr[A_n \wedge B_1] + \frac{n-1}{n}\Pr[A_n \wedge B_i \mid i \geq 2] \\
&\geq \frac{1}{n}\Pr[A_n]^2 + \frac{n-1}{n}\Pr[A_n \wedge B_i \mid i \geq 2] && \text{by (28)} \\
&\geq \frac{1}{n}\Pr[A_n]^2 + \frac{n-1}{n}\Pr[A_n]^{1+\frac{1}{n-1}} && \text{by (29)} \\
&= \Pr[A_n]\Big(\frac{1}{n}\Pr[A_n] + \frac{n-1}{n}\Pr[A_n]^{\frac{1}{n-1}}\Big) \\
&\geq \Pr[A_n]\Pr[A_n]^{\frac{1}{n}} && \text{by (30)} \\
&= \Pr[A_n]^{1+\frac{1}{n}}.
\end{aligned}$$

Finally, using elementary calculus,

$$\begin{aligned}
\Pr[A_n \wedge \neg B_i] &= \Pr[A_n] - \Pr[A_n \wedge B_i] \leq \Pr[A_n] + \Pr[A_n]^{1+\frac{1}{n}} \\
&= \Pr[A_n](1 - \Pr[A_n]^{\frac{1}{n}}) \leq \Big(\frac{n}{n+1}\Big)^n \frac{1}{n+1} \\
&\leq \frac{1}{n+1}.
\end{aligned}$$

□

7.2. Deterministic signature schemes

In this section, we consider an efficient reduction which makes a single query to a forger of some signature scheme and goes on to solve some problem. We show that either:

(i) the reduction succeeds with probability at most $1/(q_S + 1) + \delta$, where δ is negligible or

(ii) the problem which the reduction solves can be solved efficiently without a forger.

Essentially, we show that a forger can be simulated with one query to the reduction. This simulation can be used itself as input to the reduction which goes on to solve some problem. We then bound the probability of success using Coron's Lemma.

Theorem 1. *Consider a deterministic signature scheme. Let $\mathcal{F}$ be a $\langle q_H, q_S, t, \varepsilon\rangle$-forger of this signature scheme. Let $\boldsymbol{P}$ be some problem. Let $\mathcal{R}$ be a reduction which makes one query to $\mathcal{F}$ and is able to $\langle t + t', \varepsilon \cdot \varepsilon'\rangle$-solve $\boldsymbol{P}$. If*

$$\varepsilon' = \frac{1}{q_S + 1} + \delta,$$

then we can construct an algorithm $\mathcal{A}$ which $\langle 2t', \delta\rangle$-solves the problem $\boldsymbol{P}$ with two queries to the reduction.

Proof. We construct two forgers $\mathcal{F}_1$ and $\mathcal{F}_2$. Since a probabilistic algorithm is a deterministic algorithm with access to a random sequence of bits, we fix the random sequence of bits for $\mathcal{R}$ so that $\mathcal{R}$ behaves deterministically.

We select distinct messages $m_1, \ldots, m_{q_S}$ and m'. We select an integer i uniformly at random from $[1, q_S]$. The forger $\mathcal{F}_1$ makes i signing queries on the messages $m_1, \ldots, m_{i-1}, m'$ to obtain signatures $\sigma_1, \ldots, \sigma_{i-1}, \sigma'$. The forger then aborts. We run $\mathcal{R}$ with the forger $\mathcal{F}_1$ as input.

The forger $\mathcal{F}_2$ makes q_S signing queries on the messages $m_1, \ldots, m_{q_S}$ and outputs a forgery σ' of the message m'. We run $\mathcal{R}$ with the forger $\mathcal{F}_2$ as input.

Set $M = \{m_1, \ldots, m_{q_S}, m'\}$. We define a prefix-closed collection of subsequences of M^{q_S}

$$Q \subseteq \bigcup_{i=0}^{q_S} M^i$$

such that $(\gamma_1, \ldots, \gamma_j) \in Q$ iff the reduction $\mathcal{R}$ successfully returns signatures to a forger which makes signing queries for messages $\gamma_1, \ldots, \gamma_j$.

The forger $\mathcal{F}_2$ behaves correctly as a forger unless

$$(m_1, \ldots, m_{q_S}) \in Q \text{ and } (m_1, \ldots, m_{i-1}, m') \notin Q. \tag{31}$$

By Coron's lemma, the probability that (31) occurs is at most $1/(q_S + 1)$. So, the probability that the reduction $\mathcal{R}$ successfully solves the problem $\boldsymbol{P}$ is at least

$$\varepsilon' - \frac{1}{q_S + 1}.$$

If

$$\varepsilon' = \frac{1}{q_S + 1} + \delta$$

for some δ, then we are able to $\langle 2t', \delta\rangle$-solve the problem $\boldsymbol{P}$. □

Theorem 1 can be applied to any deterministic signature schemes as Coron observed in [**10**].

Corollary 1. *Consider a BLS $\langle q_H, q_S, t, \varepsilon\rangle$-forger and a reduction which is able to $\langle t + t', \varepsilon \cdot \varepsilon'\rangle$-solve the CDH problem with one query to the BLS forger. If*

$$\varepsilon' = \frac{1}{q_S + 1} + \delta,$$

then we can construct an algorithm which can $\langle 2t', \delta\rangle$-solve the co-CDH′ problem.

7.3. Non-deterministic signature schemes

In general, Theorem 1 cannot be applied to signature schemes for which each message has many distinct signatures. Our simulation of the forger $\mathcal{F}_2$ in Theorem 1 produces a signature which is determined by the reduction. A true forger will output a signature which is independent of the reduction.

Consider a signature scheme that is existentially unforgeable under chosen-message attacks. That is, given a forger of this signature scheme, we can solve some problem $\boldsymbol{P}$. Let $\mathcal{S}_m$ be the set of signatures of some message m. We will show that if there exists an efficiently computable collection of functions

$$\mathrm{reroll}_m \colon \mathcal{S}_m \to \mathcal{S}_m$$

such that for a signature $\sigma \in \mathcal{S}_m$, we have that $\mathrm{reroll}_m(\sigma)$ is selected uniformly at random from $\mathcal{S}_m$, then Coron's proof can be applied to this signature scheme.

Note that if such a function exists, then the signature scheme is not strongly unforgeable under chosen-message attacks.

Theorem 2. *Consider a $\langle q_H, q_S, t, \varepsilon\rangle$-forger of some non-deterministic signature scheme equipped with a collection of functions* reroll_m*. Also, consider a reduction which is able to $\langle t + t', \varepsilon \cdot \varepsilon'\rangle$-solve some problem $\boldsymbol{P}$ with one query to the forger. If*

$$\varepsilon' = \frac{1}{q_S + 1} + \delta,$$

then we can construct an algorithm which can $\langle 2t', \delta\rangle$-solve $\boldsymbol{P}$.

Proof. We proceed in exactly the same manner as in the proof of Theorem 1, except when the forger $\mathcal{F}_2$ is to return a forged signature σ' for message m', the forger $\mathcal{F}_2$ computes and outputs a second signature $\sigma'' = \mathrm{reroll}_m(\sigma')$. The forger $\mathcal{F}_2$ outputs forgeries which are independent of the reduction and the result follows. □

For the Waters signature scheme, if we are given a signature

$$\sigma = \langle \sigma_1, \sigma_2, \sigma_3\rangle = \langle g_1^{xy} H(m)^r, g_1^r, g_2^r\rangle,$$

then we may select a random integer r' and compute

$$\sigma' = \langle \sigma_1 \cdot H(m)^{r'}, \sigma_2 \cdot g_1^{r'}, \sigma_3 \cdot g_2^{r'}\rangle = \langle g_1^{xy} H(m)^{r+r'}, g_1^{r+r'}, g_2^{r+r'}\rangle.$$

Since we may generate a random signature on the message m, given a signature on the message m, the Waters signature scheme is equipped with a reroll function. We obtain the following corollary following from Theorem 2.

Corollary 2. *Consider a Waters $\langle q_H, q_S, t, \varepsilon \rangle$-forger and a reduction which is able to $\langle t + t', \varepsilon \cdot \varepsilon' \rangle$-solve the CDH problem with one query to the Waters forger. If*

$$\varepsilon' = \frac{1}{q_S + 1} + \delta,$$

then we can construct an algorithm which can $\langle 2t', \delta \rangle$-solve the co-CDH′ problem.

CHAPTER 8

Conclusions and Future Work

The initial descriptions of BLS, BGLS, Waters, and LOSSW mention that the schemes may be extended to Type 3 pairings. We have shown that there are various nuances to the modifications which are required to create secure schemes using Type 3 pairings.

The primary motivation for the Waters signature scheme was to avoid the random oracle assumption. Unfortunately, as we have seen, in the Waters signature scheme, we must either assume the existence of a trusted third-party or contend with extremely large public keys (around 256 elements of G_1). When we consider whether to use the Waters signature scheme or the BLS signature scheme, we must compare the Waters public-key size to BLS public-key size (which is a single element of G_2) and the trusted third-party assumption required by Waters to the random oracle assumption required by BLS. Fortunately, the security of both Waters and BLS is relative to a standard assumption, the co-CDH$'$ problem.

We made similar comparisons for LOSSW and BGLS. The LOSSW signature scheme has very large public keys and requires the assumption of a trusted third-party. The security model (the registered-key model) of LOSSW requires that the private key of each party is revealed to a certifying authority. When compared to BGLS, we must weigh the trusted third-party assumption of LOSSW against the random oracle assumption of BGLS. The public keys of LOSSW contain 256 elements of G_1. For BGLS, a public keys is a single element of G_2.

All of the schemes BLS, BGLS, Waters, and LOSSW are secure assuming that the co-CDH$'$ problem is computationally infeasible to solve. We prove the schemes secure by reducing the co-CDH$'$ problem to forging signatures. Unfortunately, the reduction succeeds with probability asymptotically $1/q_S$ where q_S is the number of signing queries made. We have shown that reductions for deterministic signature schemes necessarily succeed with probability at most $1/q_S$. Furthermore, the Waters signature scheme also suffers from a $1/q_S$ upper bound of the success probability. For BLS and BGLS, there exist non-deterministic versions of the schemes which have tight reductions to the co-CDH$'$ problem. No variation of the Waters or LOSSW schemes is known with a tight reduction.

The motivation for aggregate signature schemes is to produce a signature on multiple messages using multiple keys while using as little bandwidth as possible to transmit the signature. Unfortunately, public keys still need to be distributed. A few identity-based aggregate signature schemes which are pairing-based have been proposed but are secure assuming that non-standard problems are computationally infeasible [**15**] [**4**]. Recently, Neven shifted focus away from simply fixing the signature size to a focus on reducing total bandwidth required, introducing a new primitive called *sequential aggregate signed data* [**27**]. Neven's protocol is secure in the random-oracle model and assuming the existence of *claw-free permutations* meaning that the protocol can be based on RSA or factoring assumptions. It would be interesting to compare Neven's scheme based on RSA or factoring with the LOSSW and BGLS schemes in terms of security and efficiency.

References

[1] P. Barreto and M. Naehrig, *Pairing-friendly elliptic curves of prime order*, Selected Areas in Cryptography, LNCS **3897** (2006), 319–331

[2] M. Bellare, C. Namprempre, and G. Neven, *Unrestricted aggregate signatures,* Automata, Languages and Programming, LNCS **4596** (2007), 411–422.

[3] M. Bellare and P. Rogaway, *The exact security of digital signatures — how to sign with RSA and Rabin,* Advances in Cryptology — Eurocrypt 2006, LNCS **1070** (1996), 399–416.

[4] A. Boldreva, C. Gentry, A. ONeill, and D. H. Yum, *Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing,* 14th ACM Conference on Computer and Communications Security, (2007) 276–285.

[5] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, *Aggregate and verifiably encrypted signatures from bilinear maps,* Advances in Cryptology — Eurocrypt 2003, LNCS **2656** (2003), 416–432.

[6] P. Barreto, B. Lynn, and M. Scott *On the selection of pairing-friendly groups,* Selected areas in cryptography, LNCS **3006** (2004), 17–25.

[7] D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the Weil pairing,* Journal of Cryptology, **17** (2004), 297–319.

[8] S. Chatterjee and P. Sarkar, *Trading time for space: towards and efficient IBE scheme with short(er) public parameters in the standard model,* Information security and cryptology — ICISC 2005, LNCS **3935** (2006), 424–440.

[9] J. Chung and A. Hasan, *Asymmetric squaring formulae*, 18th IEEE Symposium on Computer Arithmetic — ARITH 2007, 113–122.

[10] J. S. Coron, *Optimal security proofs for PSS and other signature schemes,* Advances in Cryptology — Eurocrypt 2002, LNCS **2332** (2002), 272–287.

[11] A. Devegili, M. Scott and R. Dahab, *Implementing cryptographic pairings over Barreto-Naehrig curves*, Pairing-Based Cryptography — Pairing 2007, LNCS **4575** (2007), 197–207.

[12] S. Galbraith, *Pairings,* Chapter IX of I. Blake, G. Seroussi, and N. Smart, eds., Advances in Elliptic Curve Cryptography, Vol. 2, (2005).

[13] S. Galbraith, K. Paterson, and N. Smart, *Pairings for cryptographers*, Discrete Applied Mathematics, **6** (2008), 3113–3121.

[14] D. Galindo, *The exact security of pairing based encryption and signature schemes,* Workshop on Provable Security, 2004.

[15] C. Gentry and Z. Ramzan, *Identity-based aggregate signatures,* Public key cryptography — PKC 2006, LNCS **3958**, (2006) 257–273.

[16] D. Hankerson, A. Menezes, and M. Scott, *Software implementation of pairings*, in *Identity-Based Cryptography*, M. Joye and G. Neven, eds., IOS Press, to appear.

[17] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2004.

[18] F. Hess, N. Smart, and F. Vercauteren, *The Eta pairing revisited,* IEEE Transactions on Information Theory, **52** (2006), 4595–4602.

[19] B. Kang and J. Park, *On the relationship between squared pairings and plain pairings,* Cryptology ePrint Archive Report 2005/112, (2005). Available from `http://eprint.iacr.org/2005/112`.

[20] J. Katz and N. Wang, *Efficiency improvements for signature schemes with tight security reductions,* 10th ACM Conference on Computer and Communications Security (2003), 155–164.

[21] N. Koblitz and A. Menezes, *Another look at "provable security",* Journal of Cryptology, **20** (2007), 3–37.

[22] A. Lenstra, H. Lenstra, Jr., M. Manasse, and J. Pollard, *The number field sieve*, The Development of the Number Field Sieve, LNCS **1554** (1993), 11–42.

[23] E. Lee, H.-S. Lee, and C.-M. Park, *Efficient and generalized pairing computation on abelian varieties*, Cryptology ePrint Archive Report 2008/040, (2008). Available from `http://eprint.iacr.org/2008/040`.

[24] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, *Sequential aggregate signatures and multisignatures without random oracles,* Advances in Cryptology — Eurocrypt 2006, LNCS **4004** (2006), 465–485.

[25] V. Miller, *The Weil pairing and its efficient calculation*, Journal of Cryptology, **17** (2004), 235–261.

[26] D. Naccache, *Secure and practical identity-based encryption,* IET Information Security, **1** (2007), 59–64.

[27] Gregory Neven, *Efficient sequential aggregate signed data,* Advances in Cryptology - EUROCRYPT 2008, LNCS **4965**, (2008) 52-69.

[28] D. Page, N. Smart, and F. Vercauteren, *A comparison of MNT curves and supersingular curves,* Applicable Algebra in Engineering, Communication and Computing, **17**, (2006) 379–392.

[29] J. M. Pollard, *Monte Carlo methods for index computation* (mod p), Mathematics of Computation, **32**, (1978) 918–924.

[30] M. Scott, *Faster identity based encryption,* Electronics Letters, **40**, (2004) 861–862.

[31] M. Scott, *Implementing cryptographic pairings*, Pairing-Based Cryptography — Pairing 2007, LNCS **4575** (2007), 177–196.

[32] B. Waters, *Efficient identity-based encryption without random oracles,* Advances in Cryptology — Eurocrypt 2005, LNCS **3494** (2005), 114–127.