Sequential Aggregate Signatures and Multisignatures Without Random Oracles

Steve Lu⋆[1], Rafail Ostrovsky⋆⋆[2], Amit Sahai⋆⋆⋆[3], Hovav Shacham[4], and Brent Waters†[5]

[1] UCLA, stevelu@math.ucla.edu
[2] UCLA, rafail@cs.ucla.edu
[3] UCLA, sahai@cs.ucla.edu
[4] Weizmann Institute of Science, hovav.shacham@weizmann.ac.il
[5] SRI International, bwaters@csl.sri.com

Abstract. We present the first aggregate signature, the first multisignature, and the first verifiably encrypted signature provably secure without random oracles. Our constructions derive from a novel application of a recent signature scheme due to Waters. Signatures in our aggregate signature scheme are sequentially constructed, but knowledge of the order in which messages were signed is not necessary for verification. The aggregate signatures obtained are shorter than Lysyanskaya et al. sequential aggregates and can be verified more efficiently than Boneh et al. aggregates. We also consider applications to secure routing and proxy signatures.

1 Introduction

In this paper we present an aggregate signature scheme, a multisignature scheme, and a verifiably encrypted signature scheme. Unlike previous such schemes, our constructions are provably secure without random oracles. A series of papers beginning with the uninstantiability result of Canetti, Goldreich, and Halevi [10] has cast some doubt on the soundness of the random oracle methodology, making random-oracle–free schemes more attractive. Moreover, our proposed schemes are quite practical, and in some cases outperform the most efficient random-oracle–based schemes.

An aggregate signature scheme allows a collection of signatures to be able to be compressed into one short signature. Aggregate signatures are useful for applications such as secure route attestation and certificate chains where the

⋆ Supported in part by NSF grant DMS-0502315
⋆⋆ Supported in part by a gift from Teradata, Intel equipment grant, NSF Cybertrust grant No. 0430254, OKAWA research award, B. John Garrick Foundation and Xerox Innovation group Award.
⋆⋆⋆ Supported in part by grants from the NSF ITR and Cybertrust programs, a generous equipment grant from Intel, and an Alfred P. Sloan Foundation Fellowship.
† Supported by DHS and DOI contract No. NBCHF040146. Views expressed in this paper do not necessarily reflect those of DHS and DOI.

space requirements for a sequence of signatures can impact practical application performance.

Boneh et al. [8] presented the first aggregate signature scheme, which was based on the BLS signature [9] in groups with efficiently computable bilinear maps. Subsequently, Lysyanskaya et al. [20] presented a sequential RSA-based scheme that, while more limited, could be instantiated using more general assumptions. In a sequential aggregate signature scheme the aggregate signature must be constructed sequentially, with each signer modifying the aggregate signature in turn. However, most known applications are sequentially constructed anyway. One drawback of both schemes is that they are provably secure only in the random oracle model and thus there is only a heuristic argument for their security.

We present the first aggregate signature scheme that is provably secure without random oracles. Our signatures are sequentially constructed, however, unlike the scheme of Lysyanskaya et al., a verifier need not know the order in which the aggregate signature was created. Additionally, our signatures are shorter than those of Lysyanskaya et al. and can be verified more efficiently than those of Boneh et al.

In addition, we present the first multisignature scheme that is provably secure without random oracles. In a multisignature scheme, a single short object – the multisignature – can take the place of n signatures by n signers, all on the *same* message. (Aggregate signatures can be thought of as a multisignature without this restriction.) Boldyreva [6] gave the first multisignature scheme in which multisignature generation does not require signer interaction, based on BLS signatures.

Finally, we present the first verifiably encrypted signature scheme that is provably secure without random oracles. A verifiably encrypted signature is an object that anyone can confirm contains the encryption of a signature on some message, but from which only the party under whose key it was encrypted can recover the signature. Such a primitive is useful in contract signing. Boneh et al. [8] gave the first verifiably encrypted signature scheme, based on BLS signatures.

All our constructions derive from novel adaptations of the signature scheme of Waters [28], which follows from his Identity-Based Encryption scheme.

2 Preliminaries

In this section we first present some background on groups with efficiently computable bilinear maps. Next, we recall the definition of existentially unforgeable signatures. Then we present the Waters [28] signature algorithm.

2.1 Groups with Efficiently Computable Bilinear Maps

We briefly review the necessary facts about bilinear maps and bilinear map groups. (For more detail, see, e.g., [13, 27].) Consider the following setting:

- $\mathbb{G}$ and $\mathbb{G}_{\mathrm{T}}$ are multiplicative cyclic groups of order p;

– the group action on $\mathbb{G}$ and $\mathbb{G}_T$ can be computed efficiently;
– g is a generator of $\mathbb{G}$;
– $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable map with the following properties:
  • Bilinear: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$;
  • Non-degenerate: $e(g, g) \neq 1$.

We say that $\mathbb{G}$ is a bilinear group if it satisfies these requirements.

The security of our scheme relies on the hardness of the Computational Diffie-Hellman (CDH) problem in bilinear groups. We state the problem and our assumption as follows. Define the success probability of an algorithm $\mathcal{A}$ in solving the Computational Diffie-Hellman problem on $\mathbb{G}$ as

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{cdh}} \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}(g, g^a, h) = h^a : g, h \xleftarrow{\mathrm{R}} \mathbb{G}, a \xleftarrow{\mathrm{R}} \mathbb{Z}_p\right] \quad .$$

The probability is over the uniform random choice of g and h from $\mathbb{G}$, of a from $\mathbb{Z}_p$, and the coin tosses of $\mathcal{A}$. We say that an algorithm $\mathcal{A}$ (t, ϵ)-breaks Computational Diffie-Hellman on $\mathbb{G}$ if $\mathcal{A}$ runs in time at most t, and $\mathbf{Adv}_{\mathcal{A}}^{\mathrm{cdh}}$ is at least ϵ. The (t, ϵ)-Computational Diffie-Hellman assumption on $\mathbb{G}$ is that no adversary (t, ϵ)-breaks Computational Diffie-Hellman on $\mathbb{G}$.

Asymmetric Pairings and Short Representations. It is a simple (though tedious) matter to rewrite our schemes to employ an asymmetric pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Signatures will then include elements of $\mathbb{G}_1$, while public keys will include elements of G_2 and $\mathbb{G}_T$. This setting allows us to take advantage of curves due to Barreto and Naehrig [3]. With these curves, elements of $\mathbb{G}_1$ have a 160-bit representation at the 1024-bit security level.[6] In this case, security follows from the Computational co-Diffie-Hellman problem [9].

2.2 The Waters Signature Scheme

We describe the Waters signature scheme [28]. In our description the messages will be signatures on bitstrings of the form $\{0, 1\}^k$ for some fixed k. However, in practice one could apply a collision-resistant hash function $H_k : \{0, 1\}^* \to \{0, 1\}^k$ to sign messages of arbitrary length.

The scheme requires, besides the random generator $g \in \mathbb{G}$, $k + 1$ additional random generators $u', u_1, \ldots, u_k \in \mathbb{G}$. In the basic scheme, these can be generated at random as part of system setup and shared by all users. In some of the variants below, each user has generators $(u', u_1, \ldots, u_k)$ of her own, which must be included in her public key. We will draw attention to this in introducing the individual schemes.

The Waters signature scheme is a three-tuple of algorithms $\mathcal{W} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$. These behave as follows.

[6] By "1024-bit security," we mean parameters such that the conjectured complexity of computing discrete logarithms is roughly comparable to the complexity of factoring 1024-bit numbers. For a more refined analysis see Koblitz and Menezes [19].

W.Kg. Pick random $\alpha \xleftarrow{R} \mathbb{Z}_p$ and set $A \leftarrow e(g,g)^\alpha$. The public key *pk* is $A \in \mathbb{G}_T$. The private key *sk* is α.

W.Sig(sk, M)**.** Parse the user's private key *sk* as $\alpha \in \mathbb{Z}_p$ and the message M as a bitstring $(m_1, \ldots, m_k) \in \{0,1\}^k$. Pick a random $r \xleftarrow{R} \mathbb{Z}_p$ and compute

$$S_1 \leftarrow g^\alpha \cdot \big(u' \prod_{i=1}^{k} u_i^{m_i}\big)^r \qquad \text{and} \qquad S_2 \leftarrow g^r \ . \tag{1}$$

The signature is $\sigma = (S_1, S_2) \in \mathbb{G}^2$.

W.Vf(pk, M, σ)**.** Parse the user's public key *pk* as $A \in \mathbb{G}_T$, the message M as a bitstring $(m_1, \ldots, m_k) \in \{0,1\}^k$, and the signature σ as $(S_1, S_2) \in \mathbb{G}^2$. Verify that

$$e(S_1, g) \cdot e\Big(S_2,\, u' \prod_{i=1}^{k} u_i^{m_i}\Big)^{-1} \stackrel{?}{=} A \tag{2}$$

holds; if so, output `valid`; if not, output `invalid`.

This signature is existentially unforgeable under a chosen-message attack – the standard notion of signature security, due to Goldwasser, Micali, and Rivest [14] – if CDH is hard. We give a roundabout proof of this as Corollary 1.

3 Sequential Aggregate Signatures

In a sequential aggregate signature, as in an ordinary aggregate signature, a single short object – called the aggregate – takes the place of n signatures by n signers on n messages. Thus aggregate signatures are a generalization of multisignatures. Sequential aggregates differ from ordinary aggregates in that the aggregation operation is performed by each signer in turn, rather than by an unrelated party after the fact.

Aggregate signatures have many applications, as noted by Boneh et al. [8] and Lysyanskaya et al. [20]. Below, we consider two: Secure BGP route attestation and proxy signatures.

In BGP, routers generate and forward route attestations to other routers to advertise the routes which should be used to reach their networks. Secure BGP solves the problem of attestation forgery by having each router add its signature to a valid attestation before forwarding it to its neighbors. Because of the size of route attestations is limited, aggregate signatures are useful in reducing the overhead of multiple signatures along a path. Nicol, Smith, and Zhao [24] gave a detailed analysis of the application of aggregate signatures to the Secure BGP routing protocol [18]. Our sequential aggregate signature scheme is well suited for improving SBGP. Since all of the incoming route attestations need to be verified anyway, the fact that our signing algorithm requires a verification adds no overhead. Additionally, our signature scheme can have signatures that are smaller than those of Lysyanskaya et al. and verification will be faster than that of the Boneh et al. scheme.

A proxy signature scheme allows a user, called the *designator*, to delegate signing authority to another user, the *proxy signer*. This signature primitive, introduced by Mambo, Usada, and Okamoto [21], has been discussed and used in several practical applications. Boldyreva, Palacio, and Warinschi [7] show how to construct a secure proxy signature scheme from any aggregate (or sequential aggregate) signature scheme. Instantiating the Boldyreva-Palacio-Warinschi construction with our scheme, we obtain a practical proxy signature secure without random oracles.

3.1 Definitions

A sequential aggregate signature scheme includes three algorithms. The first, Kg, is used to generate public-private keypairs. The second, ASig, takes not only a private key and a message to sign, as does an ordinary signing algorithm, but also an aggregate-so-far by a set of l signers on l corresponding messages; it folds the new signature into the aggregate, yielding a new aggregate signature by $l+1$ signers on $l+1$ messages. The third algorithm, AVf, takes a purported aggregate signature, along with l public keys and l corresponding messages, and decides whether the the aggregate is valid.

The Sequential Aggregate Certified-Key Model. Because our aggregate signature behaves like a sequential aggregate signature from the signers' viewpoint, but like standard aggregate signature from the verifiers' viewpoint, we describe a security model for it that is a hybrid of the sequential aggregate chosen key model of Lysyanskaya et al. [20] and the aggregate chosen key model of Boneh et al. [8]. In both models, the adversary is given a single challenge key, along with an appropriate signing oracle for that key. His goal is to generate a sequential aggregate that frames the challenge user. The adversary is allowed to choose all the keys in that forged aggregate but the challenge key.

We prove our scheme in a more restricted model that requires that the adversary certify that the public keys it includes in signing oracle queries and in its forgery were properly generated. This we handle by having the adversary hand over the private keys before using the public keys. We could also extract the keys by rewinding or, if this is impossible, using the NIZKs proposed by Groth, Ostrovsky, and Sahai [15].

Formally, the advantage of a forger $\mathcal{A}$ in our model is the probability that the challenger outputs 1 in the following game:

Setup. Initialize the list of certified public keys $C \leftarrow \emptyset$. Choose $(pk, sk) \xleftarrow{\mathrm{R}} \mathsf{Kg}$. Run algorithm $\mathcal{A}$ with pk as input.

Certification Queries. Algorithm $\mathcal{A}$ provides a keypair (pk', sk') in order to certify pk'. Add pk' to C if sk' is its matching private key.

Signing Queries. Algorithm $\mathcal{A}$ requests a sequential aggregate signature, under the challenge key pk, on a message M. In addition, it supplies an aggregate-so-far σ' on messages $\boldsymbol{M}$ under keys $\boldsymbol{pk}$. Check that the signature σ' verifies; that each key in $\boldsymbol{pk}$ is in C; that pk does not appear in $\boldsymbol{pk}$;

and that $|\boldsymbol{pk}| < n$. If any of these fails to hold, answer `invalid`. Otherwise respond with $\sigma = \mathsf{ASig}(sk, M, \sigma', \boldsymbol{M}, \boldsymbol{pk})$.

Output. Eventually, $\mathcal{A}$ halts, outputting a forgery σ^* on messages $\boldsymbol{M}$ under keys $\boldsymbol{pk}$. This forgery must verify as valid under AVf; each key in $\boldsymbol{pk}$ (except the challenge key) must be in C; and $|\boldsymbol{pk}| \leq n$ must hold. In addition, the forgery must be nontrivial: the challenge key pk^* must appear in $\boldsymbol{pk}$, wlog at index 1 (since signature verification in our scheme has no inherent order), and the corresponding message $\boldsymbol{M}[1]$ must not have been queried by $\mathcal{A}$ of its sequential aggregate signing oracle. Output 1 if all these conditions hold, 0 otherwise.

We say that an aggregate signature scheme is $(t, q_C, q_S, n, \epsilon)$ secure if no t-time adversary making q_C certification queries and q_S signing queries can win the above game with advantage more than ϵ, where n is an upper bound on the length of the sequential aggregates involved.

3.2 Our Scheme

We start by giving some intuition for our scheme. Each signer in our scheme will have a unique public key from the Waters signature scheme

$$u', \boldsymbol{u} = (u_1, \ldots, u_k), A \leftarrow e(g, g)^{\alpha}.$$

While in the original signature scheme the private key consists only of g^{α}, in our aggregate signature scheme it is important that the private key holder will additionally choose and remember the discrete logs of $u', \boldsymbol{u} = (u_1, \ldots, u_k)$. In the Waters signature scheme, signatures are made of two group elements S_1 and S_2. At a high level, we can view S_2 as some randomness for the signature and S_1 as the signature on a message relative to that randomness.

An aggregate signature in our scheme also consists of group elements S'_1, S'_2. The second element S'_2 again consists of some "shared" randomness for the signature. When a signer wishes to add his signature on a message to an aggregate (S'_1, S'_2), he simply figures out what his S_1 component would be in the underlying signature scheme given S'_2 as the randomness. In order to perform this computation the signer must know the discrete log values of all of his public generators. He then then then multiplies this value into S'_1 and finally re-randomizes the signature.

We now formally describe the sequential aggregate obtained from the Waters signature.

Our sequential aggregate scheme is a three-tuple of algorithms $\mathcal{WSA} = (\mathsf{Kg}, \mathsf{ASig}, \mathsf{AVf})$. These behave as follows.

WSA.Kg. Pick random $\alpha, y' \xleftarrow{\text{R}} \mathbb{Z}_p$ and a random vector $\boldsymbol{y} = (y_1, \ldots, y_k) \xleftarrow{\text{R}} \mathbb{Z}_p^k$. Compute

$$u' \leftarrow g^{y'} \quad \text{and} \quad \boldsymbol{u} = (u_1, \ldots, u_k) \leftarrow (g^{y_1}, \ldots, g^{y_k}) \quad \text{and} \quad A \leftarrow e(g, g)^{\alpha} \ .$$

The user's private key is $sk = (\alpha, y', \boldsymbol{y}) \in \mathbb{Z}_p^{k+2}$. The public key is $pk = (A,\, u', \boldsymbol{u}) \in \mathbb{G}_{\mathrm{T}} \times \mathbb{G}^{k+1}$; it must be certified to ensure knowledge of the corresponding private key.

WSA.ASig$(sk, M,\, \sigma', \boldsymbol{M}, \boldsymbol{pk})$**.** The input is a private key sk, to be parsed as $(\alpha, y', y_1, \ldots, y_k) \in \mathbb{Z}_p^{k+2}$; a message M to sign, parsed as $(m_1, \ldots, m_k) \in \{0,1\}^k$; and an aggregate-so-far σ' on messages $\boldsymbol{M}$ under public keys $\boldsymbol{pk}$. Verify that σ' is valid by calling $\mathsf{AVf}(\sigma', \boldsymbol{M}, \boldsymbol{pk})$; if not, output `fail` and halt. Check that the public key corresponding to sk does not already appear in $\boldsymbol{pk}$; if it does, output `fail` and halt. (We revisit the issue of having one signer sign multiple messages below.)

Otherwise, parse σ' as $(S_1', S_2') \in \mathbb{G}^2$. Set $l \leftarrow |\boldsymbol{pk}|$. Now, for each i, $1 \le i \le l$, parse $\boldsymbol{M}[i]$ as $(m_{i,1}, \ldots, m_{i,k}) \in \{0,1\}^k$, and parse $\boldsymbol{pk}[i]$ as $(A_i,\, u_i', u_{i,1}, \ldots, u_{i,k}) \in \mathbb{G}_{\mathrm{T}} \times \mathbb{G}^{k+1}$. Compute

$$w_1 \leftarrow S_1' \cdot g^{\alpha} \cdot (S_2')^{(y' + \sum_{j=1}^{k} y_j m_j)} \qquad \text{and} \qquad w_2 \leftarrow S_2' \ . \tag{3}$$

The values (w_1, w_2) form a valid signature on $\boldsymbol{M} \| M$ under keys $\boldsymbol{pk} \| pk$, but this signature needs to be re-randomized: otherwise whoever created σ' could learn the user's private key g^{α}. Choose a random $\tilde{r} \in \mathbb{Z}_p$, and compute

$$S_1 \leftarrow w_1 \cdot \big(u' \prod_{j=1}^{k} u_j^{m_j}\big)^{\tilde{r}} \cdot \prod_{i=1}^{l} \big(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}}\big)^{\tilde{r}} \qquad \text{and} \qquad S_2 \leftarrow w_2 g^{\tilde{r}} \ . \tag{4}$$

It is easy to see that $\sigma = (S_1, S_2)$ is also a valid sequential aggregate signature on $\boldsymbol{M} \| M$ under keys $\boldsymbol{pk} \| pk$, with randomness $r + \tilde{r}$, where $w_2 = g^r$; output it and halt.

WSA.AVf$(\sigma, \boldsymbol{M}, \boldsymbol{pk})$**.** The input is a purported sequential aggregate σ on messages $\boldsymbol{M}$ under public keys $\boldsymbol{pk}$. Parse σ as $(S_1, S_2) \in \mathbb{G}$. If any key appears twice in $\boldsymbol{pk}$, if any key in $\boldsymbol{pk}$ has not been certified, or if $|\boldsymbol{pk}| \ne |\boldsymbol{M}|$, output `invalid` and halt.

Otherwise, set $l \leftarrow |\boldsymbol{pk}|$. If $l = 0$, output `valid` if $S_1 = S_2 = 1$, `invalid` otherwise.

Now, for each i, $1 \le i \le l$, parse $\boldsymbol{M}[i]$ as $(m_{i,1}, \ldots, m_{i,k}) \in \{0,1\}^k$, and parse $\boldsymbol{pk}[i]$ as $(A_i,\, u_i', u_{i,1}, \ldots, u_{i,k}) \in \mathbb{G}_{\mathrm{T}} \times \mathbb{G}^{k+1}$. Finally, verify that

$$e(S_1,\, g) \cdot e\Big(S_2,\, \prod_{i=1}^{l} \big(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}}\big)\Big)^{-1} \stackrel{?}{=} \prod_{i=1}^{l} A_i \tag{5}$$

holds; if so, output `valid`; if not, output `invalid`.

Signature Form. Consider a sequential aggregate signature on l messages $\boldsymbol{M}$ under l public keys $\boldsymbol{pk}$. For each i let $\boldsymbol{M}[i]$ be $(m_{i,1}, \ldots, m_{i,k})$ and let $\boldsymbol{pk}[i]$ be $(A_i,\, u_i', u_{i,1}, \ldots, u_{i,k})$ with corresponding private key $(\alpha_i,\, y_i', y_{i,1}, \ldots, y_{i,k})$. A well-formed sequential aggregate signature $\sigma = (S_1, S_2)$ in this case has the form

$$S_1 = \prod_{i=1}^{l} g^{\alpha_i} \cdot \prod_{i=1}^{l} \big(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}}\big)^{r} \qquad \text{and} \qquad S_2 = g^r \ .$$

Additionally, we consider $\sigma = (1, 1)$ to be a valid signature on an empty set of signers. Notice that (S_1, S_2) is the product of Waters signatures all sharing the same randomness r.

Even though in our description we did not allow a signer to sign twice in an aggregate signature, a simple trick allows for this. Suppose a signer wishes to add his signature on message M to a sequential aggregate signature that already contains his signature on another message M'. He need simply first remove his signature on M' from the aggregate, essentially by dividing it out of S_1, and multiply in a signature on $M' : M$, which is a message that attests to both M' and M.

Performance. Verification in our signatures is fast, taking approximately $k/2$ multiplications per signer in the aggregate, and only two pairings regardless of how many signers are included. In contrast, the aggregate signatures of Boneh et al. [8] take $l + 1$ pairings to verify when the aggregate includes l signers.

3.3 Proof of Security

Theorem 1. *The WSA sequential aggregate signature scheme is* $(t, q_C, q_S, n, \epsilon)$*-unforgeable if the W signature scheme is* (t', q', ϵ')*-unforgeable on* $\mathbb{G}$*, where*

$$t' = t + O(q_C + nq_S + n) \qquad \textit{and} \qquad q' = q_S \qquad \textit{and} \qquad \epsilon' = \epsilon \ .$$

Proof. Suppose that there exists an adversary $\mathcal{A}$ that succeeds with advantage ϵ. We build a simulator $\mathcal{B}$ to play the forgeability game against the W signature scheme. Given the challenge W-signature public key $pk^* = (A, u', u_1, \ldots, u_k)$, simulator $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ supplying it with the challenge key pk^*.

Certification Queries. Algorithm $\mathcal{A}$ wishes to certify some public key $pk = (A, u', u_1, \ldots, u_k)$, providing also its corresponding private key $sk = (\alpha, y', y_1, \ldots, y_k)$. Algorithm $\mathcal{B}$ checks that the private key is indeed the correct one and if so registers (pk, sk) in its list of certified keypairs.

Aggregate Signature Queries. Algorithm $\mathcal{A}$ requests a sequential aggregate signature, under the challenge key, on a message M. In addition, it supplies an aggregate-so-far σ' on messages $\boldsymbol{M}$ under keys $\boldsymbol{pk}$. The simulator first checks that the signature σ' verifies; that each key in $\boldsymbol{pk}$ has been certified; that the challenge key does not appear in $\boldsymbol{pk}$; and that $|\boldsymbol{pk}| < n$. If any of these conditions does not hold, $\mathcal{B}$ returns `fail`.
Otherwise, $\mathcal{B}$ queries its own signing oracle for key pk^*, obtaining a signature σ on message M, which we view as a sequential aggregate on messages (M) under keys (pk^*). The simulator now constructs the rest of the required aggregate by adding to σ, for each signer $\boldsymbol{pk}[i]$, the appropriate signature on message $\boldsymbol{M}[i]$ using algorithm ASig. It can do this because it knows – by means of the certification procedure – the private key corresponding to each public key in $\boldsymbol{pk}$. The result is an aggregate signature σ' on messages $\boldsymbol{M} \| M$ under keys $\boldsymbol{pk} \| pk^*$. This reconstruction method works because signatures

are re-randomized after each aggregate signing operation and because our signatures have no inherent verification order.

Output. Eventually, $\mathcal{A}$ halts, outputting a forgery, $\sigma^* = (S_1^*, S_2^*)$ on messages $\boldsymbol{M}$ under keys $\boldsymbol{pk}$. This forgery must verify as valid under AVf; each key in $\boldsymbol{pk}$ (except the challenge key) must have been certified; and $|\boldsymbol{pk}| \le n$ must hold. In addition, the forgery must be nontrivial: the challenge key pk^* must appear in $\boldsymbol{pk}$, wlog at index 1 (since signature verification in our scheme has no inherent order), and the corresponding message $\boldsymbol{M}[1]$ must not have been queried by $\mathcal{A}$ of its sequential aggregate signing oracle. If the adversary was not successful we can quit and disregard the attempt.

Now, for each i, $1 \le i \le l = |\boldsymbol{pk}| = |\boldsymbol{M}|$, parse $\boldsymbol{pk}[i]$ as $(A_i, u_i', u_{i,1}, \ldots, u_{i,k})$ and $\boldsymbol{M}[i]$ as $(m_{i,1}, \ldots, m_{i,k}) \in \{0,1\}^k$. Note that we have $pk^* = (A_1, u_1', u_{1,1}, \ldots, u_{1,k})$. Furthermore, for each i, $2 \le i \le l$, let $(\alpha_i, y_i', y_{i,1}, \ldots, y_{i,k})$ be the private key corresponding to $\boldsymbol{pk}[i]$. Algorithm $\mathcal{B}$ computes

$$S_1 \leftarrow S_1^* \cdot \prod_{i=2}^{l} \Big(g^{\alpha_i} \cdot (S_2^*)^{\left(y_i' + \sum_{j=1}^{k} y_{i,j} m_{i,j}\right)}\Big)^{-1} \qquad \text{and} \qquad S_2 \leftarrow S_2^* \ .$$

We now have

$$\begin{aligned} e(S_1, g) \cdot e\Big(S_2,\, u_1' \textstyle\prod_{j=1}^{k} u_{1,j}^{m_{1,j}}\Big)^{-1} \hspace{-4em}& \\ &= e(S_1^*, g) \cdot e\Big(S_2^*,\, u_1' \textstyle\prod_{j=1}^{k} u_{1,j}^{m_{1,j}}\Big)^{-1} \\ &\quad \times \prod_{i=2}^{l} e(g^{\alpha_i}, g)^{-1} \cdot \prod_{i=2}^{l} e\Big((S_2^*)^{\left(y_i' + \sum_{j=1}^{k} y_{i,j} m_{i,j}\right)},\, g\Big)^{-1} \\ &= e(S_1^*, g) \cdot e\Big(S_2^*,\, u_1' \textstyle\prod_{j=1}^{k} u_{1,j}^{m_{1,j}}\Big)^{-1} \\ &\quad \times \prod_{i=2}^{l} A_i^{-1} \cdot \prod_{i=2}^{l} e\Big(S_2^*,\, u_i' \textstyle\prod_{j=1}^{k} u_{i,j}^{m_{i,j}}\Big)^{-1} \\ &= e(S_1^*, g) \cdot \prod_{i=1}^{l} e\Big(S_2^*,\, u_i' \textstyle\prod_{j=1}^{k} u_{i,j}^{m_{i,j}}\Big)^{-1} \cdot \prod_{i=2}^{l} A_i^{-1} \\ &= \prod_{i=1}^{l} A_i \cdot \prod_{i=2}^{l} A_i^{-1} = A_1 = A \ . \end{aligned}$$

So (S_1, S_2) is a valid W signature on $M^* = \boldsymbol{M}[1] = (m_{1,1}, \ldots, m_{1,k})$ under key $\boldsymbol{pk}[1] = pk^*$. The last line follows from the sequential aggregate verification equation. Moreover, since $\mathcal{A}$ did not make an aggregate signing query at M^*, $\mathcal{B}$ did not make a signing query at M^*, so $\sigma = (S_1, S_2)$ is a nontrivial W signature forgery. Algorithm $\mathcal{B}$ returns it and halts.

Algorithm $\mathcal{B}$ is successful whenever $\mathcal{A}$ is. Algorithm $\mathcal{B}$ makes as many signing queries as $\mathcal{A}$ makes sequential aggregate signing queries. Algorithm $\mathcal{B}$'s running time is that of $\mathcal{A}$, plus the overhead in handling $\mathcal{A}$'s queries, and computing the final result. Each certification query can be handled in $O(1)$ time; each aggregate signing query can be handled in $O(n)$ time; and the final result can also be computed from $\mathcal{A}$'s forgery in $O(n)$ time.

4 Multisignatures

In a multisignature scheme, a single multisignature – the same size as one ordinary signature – stands for l signatures on a message M. Multisignatures were introduced by Itakura and Nakamura [17], and have been the subject of much research [26, 25, 6]. The first multisignatures in which signatures could be combined into a multisignature without interaction was proposed by Boldyreva [6], based on BLS signatures [9]. Below, we present another non-interactive multisignature scheme, based on the Waters signature, which is provably secure without random oracles.

Security Model. Micali, Ohta, and Reyzin [22] gave the first formal treatment of multisignatures. We prove security in a variant of the Micali-Ohta-Reyzin model due to Boldyreva [6]. In this model, the adversary is given a single challenge public key *pk*, and a signing oracle for that key. His goal is to output a forged multisignature σ^* on a message M^* under keys $pk_1, \ldots, pk_l$. Of these keys, pk_1 must be the challenge key *pk*. For the forgery to be nontrivial, the adversary must not have queried the signing oracle at M^*. The adversary is allowed to choose the remaining keys, but must prove knowledge of the private keys corresponding to them. For simplicity, Boldyreva handles this by having the adversary hand over the private keys; in a more complicated proof of knowledge, the keys could be extracted by rewinding, with the same result.

4.1 Our Scheme

We describe the multisignature obtained from the Waters signature. In this scheme, all users share the same random generators $u', u_1, \ldots, u_k$, which are included in the system parameters. Our scheme is a five-tuple of algorithms $\mathcal{WM} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf}, \mathsf{Comb}, \mathsf{MVf})$, which behave as follows.

WM.Kg, WM.Sig, WM.Vf. Same as W.Kg, W.Sig, and W.Vf, respectively.

WM.Comb$(\{pk_i, \sigma_i\}_{i=1}^{l}, M)$**.** For each user in the multisignature the algorithm takes as input a public key pk_i and a signature σ_i. All these signatures are on a single message M. For each i, parse user i's public key pk_i as $A_i \in \mathbb{G}_\mathrm{T}$ and her signature σ_i as $(S_1^{(i)}, S_2^{(i)}) \in \mathbb{G}^2$; parse the message M as a bitstring $(m_1, \ldots, m_k) \in \{0,1\}^k$. Verify each signature using Vf; if any is invalid, output `fail` and halt. Otherwise, compute

$$S_1 \leftarrow \prod_{i=1}^{l} S_1^{(i)} \qquad \text{and} \qquad S_2 \leftarrow \prod_{i=1}^{l} S_2^{(i)} \quad . \tag{6}$$

The multisignature is $\sigma = (S_1, S_2)$; output it and halt.

WM.MVf($\{pk_i\}_{i=1}^{l}$, M, σ)**.** For each user in the multisignature, the algorithm takes a public key pk_i. The algorithm also takes a purported multisignature σ on a message M. Parse user i's public key pk_i as $A_i \in \mathbb{G}_\mathrm{T}$, the message M as a bitstring $(m_1, \ldots, m_k) \in \{0,1\}^k$, and the multisignature σ as $(S_1, S_2) \in \mathbb{G}^2$. Verify that

$$e(S_1, g) \cdot e(S_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} \stackrel{?}{=} \prod_{i=1}^{l} A^{(i)} \tag{7}$$

holds; if so, output `valid`; if not, output `invalid`.

It is clear that if all signatures verify individually, the multisignature formed by their product also verifies according to (7). Note that we have

$$(S_1, S_2) = \left(g^{\sum_{i=1}^{l} \alpha^{(i)}} \cdot \left(u' \prod\nolimits_{j=1}^{k} u_j^{m_j}\right)^{\sum_{i=1}^{l} r^{(i)}},\ g^{\sum_{i=1}^{l} r^{(i)}} \right) ,$$

where $r^{(i)}$ is the randomness used by User i to generate her signature.

Proof of Security. The WM scheme is unforgeable if W signatures are unforgeable. The proof is given in Appendix A.

5 Verifiably Encrypted Signatures

A verifiably encrypted signature on some message attests to two facts:

– that the signer has produced an ordinary signature on that message; and
– that the ordinary signature can be recovered by the third party under whose key the signature is encrypted.

Such a primitive is useful for contract signing, in a protocol called optimistic fair exchange [1, 2]. Suppose both Alice and Bob wish to sign some contract. Neither is willing to produce a signature without being sure that the other will. But Alice can send Bob a verifiably encrypted signature on the contract. Bob can now send Alice his signature, knowing that if Alice does not respond with hers he can take Alice's verifiably encrypted signature and the transcript of his interaction with Alice to the third party – called the adjudicator – who will reveal Alice's signature.

Boneh et al. [8] introduced verifiably encrypted signatures, gave a security model for them, and constructed a scheme satisfying the definitions, based on the BLS short signature [9].

We describe the verifiably encrypted signature scheme obtained from the Waters signature scheme. Unlike the scheme of Boneh et al., ours is secure without random oracles.

Security Model. Boneh et al. specify two properties (besides correctness) that a verifiably encrypted signature scheme must satisfy: unforgeability and opacity. Both are defined in games. In each, the adversary is given a signer's public key pk and an adjudicator's public key apk. He is allowed to make verifiably encrypted signing queries of the form $\mathsf{ESig}(sk, apk, \cdot)$ and adjudication queries of the form $\mathsf{Adj}(ask, pk, \cdot, \cdot)$. In the unforgeability game, his goal is to output (M^*, η^*) such that he didn't query his signing oracle at M^*; in the opacity game his goal is to output (M^*, σ^*) such that he didn't query his *adjudication* oracle at M^*. An adversary can thus win the opacity game either by creating a forgery for the underlying signature scheme directly or by recovering the ordinary signature from an encrypted signature without the adjudicator's help.

5.1 Our Scheme

Our scheme is a seven-tuple of algorithms $\mathcal{WVES} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf}, \mathsf{AKg}, \mathsf{ESig}, \mathsf{EVf}, \mathsf{Adj})$ that behave as follows.

WVES.Kg, WVES.Sig, WVES.Vf These are the same as W.Kg, W.Sig, and W.Vf, respectively.

WVES.AKg. Pick $\beta \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, and set $v \leftarrow g^\beta$. The adjudicator's public key is $apk = v$; the adjudicator's private key is $ask = \beta$.

WVES.ESig(sk, apk, M) Parse the user's private key sk as $\alpha \in \mathbb{Z}_p$ and the adjudicator's public key apk as $v \in \mathbb{G}$. To sign the message $M = (m_1, \dots, m_k)$, compute a signature $(S_1, S_2) \xleftarrow{\mathrm{R}} \mathsf{Sig}(sk, M)$. Pick a random $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, and compute

$$K_1 \leftarrow S_1 \cdot v^s \qquad \text{and} \qquad K_2 \leftarrow S_2 \qquad \text{and} \qquad K_3 \leftarrow g^s \ .$$

The verifiably encrypted signature η is the tuple (K_1, K_2, K_3).

WVES.EVf(pk, apk, M, η)**.** Parse the user's public key pk as $A \in \mathbb{G}_\mathrm{T}$, the adjudicator's public key apk as $v \in \mathbb{G}$, and the verifiably encrypted signature η as $(K_1, K_2, K_3) \in \mathbb{G}^3$. Accept if the following equation holds:

$$e(K_1, g) \cdot e(K_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} \cdot e(K_3, v)^{-1} \stackrel{?}{=} A \ , \tag{8}$$

where $M = (m_1, \dots, m_k)$.

WVES.Adj(ask, pk, M, η)**.** Parse the adjudicator's private key ask as $\beta \in \mathbb{Z}_p$. Parse the user's public key pk as $A \in \mathbb{G}_\mathrm{T}$, and check that it has been certified. Parse the message M as $(m_1, \dots, m_k) \in \{0,1\}^k$. Verify (using EVf) that the verifiably encrypted signature η is valid, and parse it as $(K_1, K_2, K_3) \in \mathbb{G}^3$. Compute

$$S_1 \leftarrow K_1 \cdot K_3^{-\beta} \qquad \text{and} \qquad S_2 \leftarrow K_2 \ ;$$

re-randomize (S_1, S_2) by choosing $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and computing

$$S_1' \leftarrow S_1 \cdot \big(u' \prod_{i=1}^{k} u_i^{m_i}\big)^s \qquad \text{and} \qquad S_2' \leftarrow S_2 \cdot g^s \ ;$$

and output the signature (S'_1, S'_2).

It is easy to see that this scheme is valid, since if all parties are honest we have, for a verifiably encrypted signature (K_1, K_2, K_3),

$$
\begin{aligned}
&e(K_1, g) \cdot e(K_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} \cdot e(K_3, v)^{-1} \\
&\quad = \big(e(S_1, g) \cdot e(v^s, g)\big) \cdot e(S_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} \cdot e(g^s, v)^{-1} \\
&\quad = e(S_1, g) \cdot e(S_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} = A \ ,
\end{aligned}
$$

as required; and if (K_1, K_2, K_3) is a valid verifiably encrypted signature then

$$
\begin{aligned}
e(S_1, g) \cdot e(S_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} &= \big(e(K_1, g) \cdot e(K_3^{-\beta}, g)\big) \cdot e(K_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} \\
&= e(K_1, g) \cdot e(K_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} \cdot e(K_3, v)^{-1} = A \ ,
\end{aligned}
$$

so the adjudicated signature is indeed a valid one.

Proofs of Security. The WVES scheme is unforgeable if W signatures are unforgeable, and opaque if CDH is hard on $\mathbb{G}$. The proofs are given in Appendix B.

5.2 VES from General Assumptions

Recent work has shown that group signatures [4] and ring signatures [5] can be built from general assumptions using Non-Interactive Zero Knowledge (NIZK) proofs. We note that verifiably encrypted signatures can also be realized from general assumptions. Roughly, the signer signs a message, encrypts the signature to the adjudicator and then attaches a NIZK proof that this was performed correctly.

6 Comparison to Previous Work

In this section, we compare the schemes we have presented to previous schemes in the literature. For the comparison, we instantiate pairing-based schemes using Barreto-Naehrig curves [3] with 160-bit point representation. Note that BLS-based constructions must compute, for signing and verification, a hash function onto $\mathbb{G}$. This is an expensive operation [9, Sect. 3.2].

Table 1. Comparison of aggregate signature schemes. Signatures are by l signers; k is the output length of a collision resistant hash function; "R.O." denotes if the security proof uses random oracles.

Scheme	R.O.	Sequential	Key Model	Size	Verification	Signing
BGLS	YES	NO	Chosen	160 bits	$l+1$ pairings	1 exp.
LMRS-1	YES	YES	Chosen	1024 bits	$2l$ exp.	verify + 1 exp.
LMRS-2	YES	YES	Registered	1024 bits	$4l$ mult.	verify + 1 exp.
Ours	NO	YES	Registered	320 bits	2 pairings, $lk/2$ mult.	verify + 1 exp.

Sequential Aggregate Signatures. We compare our sequential aggregate signature scheme to the aggregate scheme of Boneh et al. [8] (BGLS) and to the sequential aggregate signature scheme of Lysyanskaya et al. [20] (LMRS).

We instantiate the LMRS scheme using the RSA-based permutation family with common domain devised by Hayashi, Okamoto, and Tanaka [16]. With this permutation family LMRS signatures do not grow by 1 bit with each signature, as is the case with the RSA-based instantiation given by Lysyanskaya et al. [20]; but evaluating the permutation requires two applications of the underlying RSA function. Lysyanskaya et al. give two variants of their scheme. One places constraints on the format of the RSA keys, thereby avoiding key certification; we call this variant LMRS-1. The other uses ordinary RSA keys and can have public exponent $e = 3$ for fast verification, but requires key certification, like our scheme; we call this variant LMRS-2.

We present the comparisons in Table 1. The size column gives signature length at the 1024-bit security level. The Verification and Signing columns give the computational costs of those operations; l is the number of signatures in an aggregate, and k is the output length of a collision-resistant hash function.

One drawback of our scheme is that a user's public key will be quite large. If we use a 160-bit collision resistant hash function, then keys will be approximately 160 group elements and take around 10KB to store. While it is desirable to achieve smaller public keys, this will be acceptable in many settings such as SBGP where achieving the signature size is a much more important consideration than the public key size. Additionally, Naccache [23] and Chatterjee and Sarkar [11] independently proposed ways to achieve shorter public keys in the Waters signature scheme. Using these methods we can also achieve considerably shorter public keys.

Multisignatures. We compare our multisignature scheme to the Boldyreva multisignature [6]. We present the comparisons in Table 2. The size column gives signature length at the 1024-bit security level. The Verification and Signing columns give the computational costs of those operations; l is the number of signatures in a multisignature, and k is the output length of a collision-resistant hash function.

Verifiably Encrypted Signatures. We compare our verifiably encrypted signature scheme to that of Boneh et al. [8] (BGLS). We present the comparisons in

Table 2. Comparison of multisignature schemes. Multisignatures are by l signers; k is the output length of a collision resistant hash function; "R.O." denotes if the security proof uses random oracles.

Scheme	R.O.	Key Model	Size	Verification	Signing
Boldyreva	YES	Registered	160 bits	2 pairings	1 exp.
Ours	NO	Registered	320 bits	2 pairings, $k/2$ mult.	1 exp.

Table 3. Comparison of verifiably encrypted signature schemes. We let k be the output length of a collision resistant hash function. "R.O." specifies whether the security proof uses random oracles.

Scheme	R.O.	Key Model	Size	Verification	Generation
BGLS	YES	Registered	320 bits	3 pairings	3 exp.
Ours	NO	Registered	480 bits	3 pairings, $k/2$ mult.	4 exp.

Table 3. The size column gives signature length at the 1024-bit security level. The Verification and Generation columns give the computational costs of those operations; k is the output length of a collision-resistant hash function.

7 Conclusions and Open Problems

In this paper we gave the first aggregate signature scheme which is provably secure without random oracles; the first multisignature scheme which is provably secure without random oracles; and the first verifiably encrypted signature scheme which is provably secure without random oracles. All our constructions derive from the recent signature scheme due to Waters [28]. All our constructions are quite practical.

Signatures in our aggregate signature scheme are sequentially constructed, but knowledge of the order in which messages are signed is not necessary for verification. Additionally, our scheme gives shorter signatures than in the LMRS sequential aggregate signature scheme [20] and has a more efficient verification algorithm than the BGLS aggregate signature scheme [8]. That this gives some interesting tradeoffs for practical applications such as secure routing and proxy signatures.

Some interesting problems remain open for random-oracle–free aggregate signatures:

1. To find a scheme which supports full aggregation, in which aggregate signature do not need to be sequentially constructed. While many applications only require sequential aggregation, having a more general capability is desirable.
2. To find a sequential aggregate signature scheme provably secure in the chosen-key model.
3. To find a sequential aggregate signature scheme with shorter user keys. The size of public keys in our system reflects the size of keys in the underlying Waters signature scheme. Naccache [23] and Chatterjee and Sarkar [11]

have proposed ways to shorten the public keys of the Waters IBE/signature scheme by trading off parameter size with tightness in the security reduction. It would be better to have a solution in which the public key is just a few group elements.

The last two are particularly important for certificate chain compression, proposed by Boneh et al. [8] as an application for aggregate signatures. If keys need to be registered with an authority then a chaining application is impractical, and having large public keys negates any benefit from reducing the signature size in a certificate chain, since the keys must be included in the certificates.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE J. Selected Areas in Comm.*, 18(4):593–610, Apr. 2000.

[2] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with offline TTP. In P. Karger and L. Gong, editors, *Proceedings of IEEE Security & Privacy*, pages 77–85, May 1998.

[3] P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Proceedings of SAC 2005*, volume 3897 of *LNCS*, pages 319–31. Springer-Verlag, 2006.

[4] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 614–29. Springer-Verlag, May 2003.

[5] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In S. Halevi and T. Rabin, editors, *Proceedings of TCC 2006*, volume 3876 of *LNCS*, pages 60–79. Springer-Verlag, Mar. 2006.

[6] A. Boldyreva. Threshold signature, multisignature and blind signature schemes based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer-Verlag, Jan. 2003.

[7] A. Boldyreva, A. Palacio, and B. Warinschi. Secure proxy signature schemes for delegation of signing rights. Cryptology ePrint Archive, Report 2003/096, 2003. `http://eprint.iacr.org/`.

[8] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 416–32. Springer-Verlag, May 2003.

[9] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, Sept. 2004. Extended abstract in *Proceedings of Asiacrypt 2001*.

[10] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–94, July 2004.

[11] S. Chatterjee and P. Sarkar. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In D. Won and S. Kim, editors, *Proceedings of ICISC 2005*, LNCS. Springer-Verlag, Dec. 2005. To appear.

[12] J.-S. Coron and D. Naccache. Boneh et al.'s k-element aggregate extraction assumption is equivalent to the Diffie-Hellman assumption. In C. S. Laih, editor, *Proceedings of Asiacrypt 2003*, volume 2894 of *LNCS*, pages 392–7. Springer-Verlag, Dec. 2003.

[13] S. Galbraith. Pairings. In I. F. Blake, G. Seroussi, and N. Smart, editors, *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Notes*, chapter IX, pages 183–213. Cambridge University Press, 2005.

[14] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, 1988.

[15] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for NP. In S. Vaudenay, editor, *Proceedings of Eurocrypt 2006*, LNCS. Springer-Verlag, May 2006. This volume.

[16] R. Hayashi, T. Okamoto, and K. Tanaka. An RSA family of trap-door permutations with a common domain and its applications. In F. Bao, R. H. Deng, and J. Zhou, editors, *Proceedings of PKC 2004*, volume 2947 of *LNCS*, pages 291–304. Springer-Verlag, Mar. 2004.

[17] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC J. Res. & Dev.*, 71:1–8, Oct. 1983.

[18] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (Secure-BGP). *IEEE J. Selected Areas in Comm.*, 18(4):582–92, April 2000.

[19] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In N. Smart, editor, *Proceedings of Cryptography and Coding 2005*, volume 3796 of *LNCS*, pages 13–36. Springer-Verlag, Dec. 2005.

[20] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In C. Cachin and J. Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 74–90. Springer-Verlag, May 2004.

[21] M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In L. Gong and J. Stearn, editors, *Proceedings of CCS 1996*, pages 48–57. ACM Press, Mar. 1996.

[22] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures (extended abstract). In P. Samarati, editor, *Proceedings of CCS 2001*, pages 245–54. ACM Press, Nov. 2001.

[23] D. Naccache. Secure and practical identity-based encryption. Cryptology ePrint Archive, Report 2005/369, 2005. `http://eprint.iacr.org/`.

[24] D. Nicol, S. Smith, and M. Zhao. Evaluation of efficient security for BGP route announcements using parallel simulation. *Simulation Modelling Practice and Theory*, 12:187–216, 2004.

[25] K. Ohta and T. Okamoto. Multisignature schemes secure against active insider attacks. *IEICE Trans. Fundamentals*, E82-A(1):21–31, 1999.

[26] T. Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Computer Systems*, 6(4):432–41, November 1988.

[27] K. Paterson. Cryptography from pairings. In I. F. Blake, G. Seroussi, and N. Smart, editors, *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Notes*, chapter X, pages 215–51. Cambridge University Press, 2005.

[28] B. Waters. Efficient identity-based encryption without random oracles. In R. Cramer, editor, *Proceedings of Eurocrypt 2005*, volume 3494 of *LNCS*, pages 114–27. Springer-Verlag, May 2005.

A WM Proof of Security

Theorem 2. *The WM multisignature scheme is* (t, q, ϵ)*-unforgeable if the W signature scheme is* (t', q', ϵ')*-unforgeable, where*

$$t' = t + O(q) \qquad \text{and} \qquad q' = q \qquad \text{and} \qquad \epsilon' = \epsilon \ .$$

Proof. Suppose $\mathcal{A}$ is an adversary that can forge multisignatures, and (t, q, ϵ)-breaks the WM scheme. We show how to construct an algorithm $\mathcal{B}$ that (t', q, ϵ)-breaks the W scheme. Algorithm $\mathcal{B}$ is given a W public key $A = e(g, g)^{\alpha}$. It interacts with $\mathcal{A}$ as follows.

Setup. Simulator $\mathcal{B}$ invokes $\mathcal{A}$, providing to it the public key A.

Signature queries. Algorithm $\mathcal{A}$ requests a signature on some message M under the challenge key A. Algorithm $\mathcal{B}$ requests a signature on M in turn from its own signing oracle, and returns the result to the adversary.

Output. Finally, $\mathcal{A}$ halts, having output a signature (S_1^*, S_2^*) on some message M^*, along with public keys $A^{(1)}, \ldots, A^{(l)}$ for some l, where $A^{(1)}$ equals A, the challenge key. It must not previously have requested a signature on M^*. In addition, it outputs the private keys $\alpha^{(2)}, \ldots, \alpha^{(l)}$ for all keys except the challenge key. Algorithm $\mathcal{B}$ sets $S \leftarrow S_1^* / \prod_{i=2}^{l} g^{\alpha^{(i)}}$. Then we have

$$\begin{aligned} e(S, g) \cdot e(S_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} &= e(S_1, g) \cdot e(S_2, u' \prod_{i=1}^{k} u_i^{m_i})^{-1} \cdot \prod_{i=2}^{l} e(g, g)^{-\alpha^{(i)}} \\ &= \prod_{i=1}^{l} A^{(i)} \cdot \prod_{i=2}^{l} A^{-(i)} = A^{(1)} = A \ , \end{aligned}$$

so (S, S_2) is a valid W signature on M^* under the challenge key A. Since $\mathcal{A}$ did not make a signing query to the challenger at M^*, neither did $\mathcal{B}$ make a signing query to its own signing oracle at M^*, and the forgery is thus nontrivial. Algorithm $\mathcal{B}$ outputs (S, S_2) and halts.

Thus $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does. Algorithm $\mathcal{B}$ makes exactly as many signing queries as $\mathcal{A}$ does. Its running time is the same as $\mathcal{A}$'s, plus the time required for setup and output – both $O(1)$ – and to handle $\mathcal{A}$'s signing queries – $O(1)$ for each of at most q queries.

B WVES Proofs of Security

B.1 Unforgeability

Theorem 3. *The WVES verifiably encrypted signature scheme is* (t, q_S, q_A, ϵ)*-unforgeable if the W signature scheme is* (t', q', ϵ')*-unforgeable, where*

$$t' = t + O(q_S + q_A) \qquad \text{and} \qquad q' = q_S \qquad \text{and} \qquad \epsilon' = \epsilon \ .$$

Proof. We show how to turn a verifiably-encrypted signature forger $\mathcal{A}$ into a forger $\mathcal{B}$ for the underlying Waters signature scheme.

Algorithm $\mathcal{B}$ is given a Waters signature public key $A = e(g,g)^\alpha$. It picks $\beta \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, sets $v \leftarrow g^\beta$, and provides the adversary $\mathcal{A}$ with A and v.

When $\mathcal{A}$ requests a verifiably encrypted signature on some message M, the challenger $\mathcal{B}$ requests a signature on M from its own signing oracle, obtaining a signature (S_1, S_2). It picks $s \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ and computes

$$K_1 \leftarrow S_1 \cdot v^s \qquad \text{and} \qquad K_2 \leftarrow S_2 \qquad \text{and} \qquad K_3 \leftarrow g^s \ .$$

The tuple (K_1, K_2, K_3) is a valid verifiably encrypted signature on M. Algorithm $\mathcal{B}$ provides $\mathcal{A}$ with it. (Here $\mathcal{B}$ is simply evaluating ESig, except that it uses its signing oracle instead of evaluating Sig directly.)

When algorithm $\mathcal{A}$ requests adjudication of a verifiably encrypted signature (K_1, K_2, K_3) on some message M under the challenge key A, $\mathcal{B}$ responds with $\mathsf{Adj}\big(\beta, A, M, \ (K_1, K_2, K_3)\big)$. Note that $\mathcal{B}$ knows the adjudicator's private key β.

Finally, $\mathcal{A}$ outputs a forged verifiably-encrypted signature (K_1^*, K_2^*, K_3^*) on some message $M^* = (m_1^*, \ldots, m_k^*)$. Algorithm $\mathcal{A}$ must never have made a verifiably encrypted signing query at M^*.

The challenger $\mathcal{B}$ computes

$$S_1^* \leftarrow K_1^* \cdot (K_3^*)^{-\beta} \qquad \text{and} \qquad S_2^* \leftarrow K_2^*.$$

Then we have

$$\begin{aligned} & e(S_1^*, g) \cdot e\big(S_2^*, \, u' \prod_{i=1}^k u_i^{m_i^*}\big)^{-1} \\ & \quad = \Big[e(K_1^*, g) \cdot e\big(K_2^*, \, u' \prod_{i=1}^k u_i^{m_i^*}\big)^{-1} \Big] \cdot e\big((K_3^*)^{-\beta}, g\big) \\ & \quad = e(K_1^*, g) \cdot e\big(K_2^*, \, u' \prod_{i=1}^k u_i^{m_i^*}\big)^{-1} \cdot e(K_3^*, v)^{-1} = A \ , \end{aligned}$$

and (S_1^*, S_2^*) is therefore a valid Waters signature on M^*. The last equality follows from equation (8). Because $\mathcal{A}$ did not make a verifiably encrypted signing query at M^*, neither did $\mathcal{B}$ make a signing query at M^*, and the forgery is thus nontrivial. The challenger $\mathcal{B}$ outputs (S_1^*, S_2^*) and halts.

Algorithm $\mathcal{B}$ thus succeeds whenever $\mathcal{A}$ does. Its running time overhead is $O(1)$ for each of $\mathcal{A}$'s verifiably encrypted signing and adjudication queries, and for computing the final output.

B.2 Opacity

For convenience, we prove opacity by reduction from the aggregate extraction assumption: given $(g^\alpha, g^\beta, g^\gamma, g^\delta, g^{\alpha\gamma+\beta\delta})$, computing $g^{\alpha\gamma}$ is hard. Coron and Naccache [12] showed that this assumption, introduced by Boneh et al. [8], is equivalent to CDH.

Theorem 4 (Coron–Naccache [12]). *The aggregate extraction and Computational Diffie-Hellman problems are Karp reducible to each other with $O(1)$ computation.*[7]

Theorem 5. *The WVES verifiably encrypted signature scheme is (t, q_S, q_A, ϵ)-opaque if aggregate extraction is (t', ϵ')-hard on $\mathbb{G}$, where*

$$t' = t + O(q_S + q_A) \qquad \textit{and} \qquad q' = q_S \qquad \textit{and} \qquad \epsilon' = 4kq_A\epsilon \ .$$

Proof. Given an algorithm $\mathcal{A}$ that breaks the opacity of the scheme, we show how to construct an algorithm $\mathcal{B}$ that breaks the aggregate extraction assumption.

The challenger $\mathcal{B}$ is given values g^α, g^β, g^γ, and g^δ, along with $g^{\alpha\gamma+\beta\delta}$; its goal is to produce $g^{\alpha\gamma}$. It sets $v \leftarrow g^\beta$, $g_1 \leftarrow g^\alpha$, and $g_2 \leftarrow g^\gamma$. It computes $A \leftarrow e(g_1, g_2) = e(g, g)^{\alpha\gamma}$.

Let $\lambda = 2q_A$. Algorithm $\mathcal{B}$ picks $\kappa \stackrel{\text{R}}{\leftarrow} \{0, \ldots, k\}$, $x', x_1, \ldots, x_k \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_\lambda = \{0, \ldots, \lambda - 1\}$ and $y', y_1, \ldots, y_k \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$ and sets

$$u' \leftarrow g_2^{x'-\kappa\lambda} g^{y'} \qquad \text{and} \qquad u_i \leftarrow g_2^{x_i} g^{y_i} \quad \text{for } i = 1, \ldots, k \ .$$

It then interacts with $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{B}$ gives to $\mathcal{A}$ the system parameters $(g, u', u_1, \ldots u_k)$, the signer's public key A, and the adjudicator's public key v. Note that the private signing key is $\alpha\gamma$.

Verifiably Encrypted Signing Queries. $\mathcal{A}$ requests a verifiably-encrypted signature on $M = (m_1, \ldots, m_k) \in \{0,1\}^k$ under challenge key A and adjudicator key v. Define $F = -\kappa\lambda + x' + \sum_{i=1}^k x_i m_i$ and $J = y' + \sum_{i=1}^k y_i m_i$. If $F \neq 0 \bmod p$ algorithm $\mathcal{B}$ proceeds as follows. It picks $r \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$ and sets

$$S_1 \leftarrow g_1^{-J/F} \big(u' \prod_{i=1}^k u_i^{m_i}\big)^r \qquad \text{and} \qquad S_2 \leftarrow g_1^{-1/F} g^r \ .$$

This is a valid W signature with randomness $\tilde{r} = r - \alpha/F$: observing that $u' \prod_{i=1}^k u_i^{m_i} = g_2^F g^J$, we see that

$$S_1 = g_1^{-J/F} \big(u' \prod_{i=1}^k u_i^{m_i}\big)^r = g_2^\alpha (g_2^F g^J)^{-\alpha/F} (g_2^F g^J)^r = g^{\alpha\gamma} \big(u' \prod_{i=1}^k u_i^{m_i}\big)^{\tilde{r}} \ ,$$

where for the second equality we have multiplied and divided by g_2^α. Algorithm $\mathcal{B}$ then encrypts (S_1, S_2) by choosing $s \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$ and setting

$$K_1 \leftarrow S_1 \cdot v^s \qquad \text{and} \qquad K_2 \leftarrow S_2 \qquad \text{and} \qquad K_3 \leftarrow g^s \ .$$

If $F = 0$, however, $\mathcal{B}$ picks $r, s \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$ and sets

[7] Strictly speaking, the amount of work is poly-logarithmic in the security parameter since the group element representations grow. The number of algebraic operations is constant.

$$K_1 \leftarrow (g^{\alpha\gamma+\gamma\delta})\cdot(g^{\gamma})^s\cdot\big(u'\prod_{i=1}^{k} u_i^{m_i}\big)^r \quad\text{and}\quad K_2 \leftarrow g^r \quad\text{and}\quad K_3 \leftarrow (g^{\delta})\cdot g^s \ .$$

This is a W signature with randomness r, encrypted with randomness $\delta + s$. In either case, $\mathcal{B}$ returns to $\mathcal{A}$ the verifiably encrypted signature (K_1, K_2, K_3).

Adjudication Queries. Suppose $\mathcal{A}$ requests adjudication on (K_1, K_2, K_3) for message $M = (m_1, \ldots, m_k)$. Algorithm $\mathcal{B}$ first verifies that (K_1, K_2, K_3) is valid and rejects it otherwise. Define $F = -\kappa\lambda + x' + \sum_{i=1}^{k} x_i m_i$ and $J = y' + \sum_{i=1}^{k} y_i m_i$ as before. If $F = 0 \bmod p$, $\mathcal{B}$ declares failure and halts. Otherwise, it picks $r \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and computes

$$S_1 \leftarrow g_1^{-J/F}\big(u'\prod_{i=1}^{k} u_i^{m_i}\big)^r \qquad\text{and}\qquad S_2 \leftarrow g_1^{-1/F} g^r$$

as above, returning (S_1, S_2) to $\mathcal{A}$.

(Note that $\mathcal{A}$ must previously have made a verifiably encrypted signing query at M, since otherwise we could use it to break the unforgeability of WVES.)

Output. Finally, algorithm $\mathcal{A}$ outputs a signature (S_1^*, S_2^*) on a message $M^* = (m_1^*, \ldots, m_k^*)$; it must not have queried its adjudication oracle at M^*. Define $F^* = -\kappa\lambda + x' + \sum_{i=1}^{k} x_i m_i^*$ and $J^* = y' + \sum_{i=1}^{k} y_i m_i^*$. If $F^* \neq 0 \bmod p$, $\mathcal{B}$ declares failure and exits. Otherwise, we have $u' \prod_{i=1}^{k} u_i^{m_i^*} = g^{J^*}$, so that

$$\begin{aligned} e(g_1, g_2) = A &= e(S_1^*, g)\cdot e\Big(S_2^*,\, u'\prod_{i=1}^{k} u_i^{m_i^*}\Big)^{-1} \\ &= e(S_1^*, g)\cdot e\big(S_2^*, g^{J^*}\big)^{-1} = e\big(S_1^*(S_2^*)^{-J^*},\, g\big) \ , \end{aligned}$$

and $S_1^*(S_2^*)^{-J^*}$ equals $g^{\alpha\gamma}$, which is the solution to the aggregate extraction challenge; $\mathcal{B}$ outputs it and halts.

The probability that $\mathcal{B}$ doesn't abort in any adjudication query is at least $1 - 1/\lambda$; since there are at most $q_A = \lambda/2$ such queries, $\mathcal{B}$ manages to answer all queries without aborting with probability at least $1/2$. Having done so, $\mathcal{B}$ then receives a forgery such that $F^* = 0 \bmod p$ with probability at least $1/(\kappa\lambda) \geq 1/(2kq_A)$. Thus $\mathcal{B}$ succeeds with probability at least $\epsilon/(4kq_A)$. (For more detailed probability analysis, see Waters' original proof [28].) Algorithm $\mathcal{B}$'s run-time overhead is $O(1)$ to answer each of $\mathcal{A}$'s queries and to compute the final output.

Security of the Waters Signature. The reduction above did not require that $\mathcal{A}$ had requested a verifiably encrypted signature at M^*. It is easy to convert an algorithm $\mathcal{A}'$ that forges the underlying W signature to a WVES opacity breaker of this sort: simulate a W signing oracle by a call to the verifiably encrypted signing oracle followed by a call to the adjudication oracle. Combining this insight with Theorems 5 and 4 immediately gives the following corollary:

Corollary 1 (Waters [28]). *The Waters signature scheme is (t, q, ϵ)-unforgeable if Computational Diffie-Hellman is $(t + O(q),\, 4kq\epsilon)$-hard on $\mathbb{G}$. Here q is the number of signing queries.*