J. Cryptol. (2013) 26: 340–373
DOI: 10.1007/s00145-012-9126-5

Journal of
CRYPTOLOGY

Sequential Aggregate Signatures, Multisignatures, and Verifiably Encrypted Signatures Without Random Oracles

Steve Lu*, Rafail Ostrovsky†, and Amit Sahai‡
Los Angeles, CA, USA
stevelu@math.ucla.edu; rafail@cs.ucla.edu; sahai@cs.ucla.edu

Hovav Shacham§
La Jolla, CA, USA
hovav@cs.ucsd.edu

Brent Waters¶
Austin, TX, USA
bwaters@cs.utexas.edu

Communicated by Keneth G. Paterson

Received 8 July 2009
Online publication 22 June 2012

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.'s sequential aggregates and can be verified more efficiently than Boneh et al.'s aggregates. We also consider applications to secure routing and proxy signatures.

Key words. Waters signature, Bilinear map, Secure BGP.

* S. Lu was supported in part by NSF Grant DMS-0502315.

† R. Ostrovsky was 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.

‡ A. Sahai was 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.

§ H. Shacham was supported by a MURI Grant administered by the Air Force Office of Scientific Research. Work done while at the Weizmann Institute of Science, supported by a Koshland Scholars Program Fellowship.

¶ B. Waters was supported by DHS and DOI Contract No. NBCHF040146. Views expressed in this paper do not necessarily reflect those of DHS and DOI.

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 [13] have 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 space requirements for a sequence of signatures can have an impact on practical application performance.

Boneh et al. [11] presented the first aggregate signature scheme, which was based on the BLS short signature due to Boneh, Lynn, and Shacham [12] in groups with efficiently computable bilinear maps. Subsequently, Lysyanskaya et al. [28] 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 [7] 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, as people can tell, contains 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. [11] gave the first verifiably encrypted signature scheme, based on BLS signatures.

All our constructions derive from novel adaptations of the signature scheme of Waters [44], which follows from his identity-based encryption (IBE) scheme. In particular, we are (to our knowledge) the first to observe that the Waters signature remains secure when the generators u' and $u_1, \ldots, u_k$ are chosen by a user and their discrete logarithms (to some base) known to her, and when these generators are shared by all users and their discrete logarithms are known to a central authority. Our sequential aggregate signature makes use of the former setting; our multisignature, of the latter.

Previous Publication A preliminary version of this work appeared in the proceedings of Eurocrypt 2006 [27]. This is the full version.

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 [44] 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. [17,38].) Consider the following setting:

- $\mathbb{G}$ and $\mathbb{G}_T$ 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 $\mathbb{G}$;
- $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 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

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

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 $\mathcal{A}$.[1] 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 $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Signatures will then include elements of $\mathbb{G}_1$, while public keys will include elements of $\mathbb{G}_2$ 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.[2] In this case, security follows from the Computational co-Diffie–Hellman problem [12].

[1] Here and below, the symbol "$\stackrel{\mathrm{R}}{\leftarrow}$" stands for uniform random selection from a set or sampling from the output of an algorithm when run with uniform random coins.

[2] 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 [26].

Group Membership Tests There exist efficient algorithms for verifying that a bitstring represents an algebraic object that is an element of the groups $\mathbb{G}$ or $\mathbb{G}_T$. Throughout this paper we assume that all algorithms test that their inputs are in the correct groups before operating on them; signature verification algorithms, in particular, must reject a signature if its components are not in the appropriate groups.

2.2. *The Waters Signature Scheme*

We describe the Waters signature scheme [44]. 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 $H_k\colon \{0,1\}^* \to \{0,1\}^k$ 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 $\mathsf{W} = (\mathsf{W.Kg}, \mathsf{W.Sig}, \mathsf{W.Vf})$. These behave as follows.

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

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

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

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

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 [20]—if CDH is hard. We give a roundabout proof of this as Corollary 5.5.

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 ag-

gregates 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. [11] and Lysyanskaya et al. [28]. Below, we consider two: Secure BGP route attestation and proxy signatures.

BGP, the Border Gateway Protocol, is the protocol by which the core routers that make up the Internet backbone agree on how packets should be routed among them. (The latest version of BGP is specified in RFC 4271 [40].) 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. Since 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 [35] gave a detailed analysis of the application of aggregate signatures to the Secure BGP routing protocol [25]. 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, called the proxy signer. This signature primitive, introduced by Mambo, Usuda, and Okamoto [29], has been discussed and used in several practical applications. Boldyreva, Palacio, and Warinschi [8] 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 and Security Model*

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 aggregate is valid.

More formally, the key generation algorithm Kg, is a randomized algorithm whose output is a public–private keypair (pk,sk). The aggregate signing algorithm, ASig, is a randomized algorithm that takes a private key sk, a message $M \in \{0,1\}^*$ to sign, an aggregate-so-far σ', and two l-element vectors: one, $\mathbf{M}$, of messages, the other, $\mathbf{pk}$, of public keys. A system parameter, n, serves as an upper bound on the length of aggregate signatures and therefore on l. The aggregate-so-far should be a valid sequential aggregate signature on the messages in $\mathbf{M}$, each under the corresponding public key in $\mathbf{pk}$, and the signer's public key pk should not appear in $\mathbf{pk}$. The signing algorithm outputs a new aggregate-so-far, σ', on messages $\mathbf{M}\|M$ under public keys $\mathbf{pk}\|pk$. Finally, the aggregate verification algorithm, AVf, takes an aggregate signature σ, a vector of messages $\mathbf{M}$, and a vector of public keys $\mathbf{pk}$. It verifies that both vectors are the same

length l; that l is at most n, the maximum number of signatures in an aggregate; that no public key appears more than once in **pk**; and that σ is a valid sequential aggregate signature on the messages in **M** under the respective public keys in **pk**. If all these checks pass, the algorithm returns 1; otherwise it returns 0.

The Sequential Aggregate Certified-Key Model Since 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. [28] and the aggregate chosen key model of Boneh et al. [11]. 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.[3]

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. More realistically, a system incorporating our signature scheme could require users to engage with the authority in an interactive proof of knowledge of their private keys, and in the proof of security use rewinding to extract the keys; or else require non-interactive proofs that a committed private key is the correct one (using, e.g., the non-interactive zero knowledge proofs proposed by Groth, Ostrovsky, and Sahai [21]), and in the proof of security set the common reference string to allow extraction.

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 $(pk,sk) \xleftarrow{R} \mathsf{Kg}$. 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.[4]

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 **M** under keys **pk**. Check that the signature σ' verifies; that each key in **pk** is in C; that pk does not appear in **pk**; and that $|\mathbf{pk}| < n$. Here n is an upper bound on the length of a sequential aggregate, a game parameter. If any of these fails to hold, answer `invalid`. Otherwise respond with $\sigma = \mathsf{ASig}(sk, M, \sigma', \mathbf{M}, \mathbf{pk})$.

Output. Eventually, $\mathcal{A}$ halts, outputting a forgery σ^* on messages **M** under keys **pk**. This forgery must verify as valid under AVf; each key in **pk** (except the challenge key)

[3] The original security model of Boneh et al. included the additional restriction that the messages included in an aggregate all be distinct. As observed by Shacham [43] and formalized by Bellare, Namprempre, and Neven [5], this restriction is in fact unnecessary provided that signatures are computed over $H(pk\|M)$ rather than $H(M)$.

[4] A private key matches a public key if (pk, sk) is in the image of Kg. For some signature schemes (including the sequential aggregate signature that we propose—see below), it is possible to check this relation directly. For others, the requirement of private key disclosure should be implemented by having the adversary disclose the coins it provided to the key generation algorithm.

must be in C; and $|\mathbf{pk}| \leq n$ must hold. In addition, the forgery must be nontrivial: the challenge key pk must appear in $\mathbf{pk}$, wlog at index 1 (since signature verification in our scheme has no inherent order), and the corresponding message $\mathbf{M}[1]$ 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

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

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 $u', \mathbf{u} = (u_1, \ldots, u_k)$. 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 multiplies this value into S_1' and finally re-randomizes the signature.

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

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

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

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

The user's private key is $sk = (\alpha, y', \mathbf{y}) \in \mathbb{Z}_p^{k+2}$. The public key is $pk = (A, u', \mathbf{u}) \in \mathbb{G}_T \times \mathbb{G}^{k+1}$; it must be certified to ensure knowledge of the corresponding private key.

WSA.ASig$(sk, M, \sigma', \mathbf{M}, \mathbf{pk})$**.** 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 $\mathbf{M}$ under public keys $\mathbf{pk}$. Verify that σ' is valid by

calling $\mathsf{WSA.AVf}(\sigma', \mathbf{M}, \mathbf{pk})$; if not, output $\perp$ and halt. Check that the public key corresponding to sk does not already appear in $\mathbf{pk}$; if it does, output $\perp$ 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 $l \leftarrow |\mathbf{pk}|$. Now, for each i, $1 \le i \le l$, parse $\mathbf{M}[i]$ as $(m_{i,1}, \ldots, m_{i,k}) \in \{0,1\}^k$, and parse $\mathbf{pk}[i]$ as $(A_i, u_i', u_{i,1}, \ldots, u_{i,k}) \in \mathbb{G}_{\mathrm{T}} \times \mathbb{G}^{k+1}$. Compute

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

The values (w_1, w_2) form a valid signature on $\mathbf{M} \| M$ under keys $\mathbf{pk} \| pk$, 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

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

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

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

Otherwise, set $l \leftarrow |\mathbf{pk}|$. If $l = 0$, output `valid` if $S_1 = S_2 = 1$, `invalid` otherwise. Now, for each i, $1 \le i \le l$, parse $\mathbf{M}[i]$ as $(m_{i,1}, \ldots, m_{i,k}) \in \{0,1\}^k$, and parse $\mathbf{pk}[i]$ 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

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

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

Signature Form Consider a sequential aggregate signature on l messages $\mathbf{M}$ under l public keys $\mathbf{pk}$. For each i let $\mathbf{M}[i]$ be $(m_{i,1}, \ldots, m_{i,k})$ and let $\mathbf{pk}[i]$ 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

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

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.

Multiple Messages from One Signer 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.

To see why this trick works, suppose the signer's public key is $pk = (A, u', u_1, \ldots, u_k)$ and her private key is $sk = (\alpha, y', \mathbf{y})$, and consider an $(l + 1)$-element sequential aggregate signature $(S1, S2)$ under keys $\mathbf{pk} \| pk$ on messages $\mathbf{M} \| M$, where $M = (m_1, \ldots, m_k)$. If this is a valid sequential aggregate signature, then the sequential aggregate verification equation guarantees that we have

$$e(S_1, g) \cdot e\left(S_2, \left(u' \prod_{j=1}^{k} u_j^{m_j}\right) \cdot \prod_{i=1}^{l} \left(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}}\right)\right)^{-1} = A \cdot \prod_{i=1}^{l} A_i.$$

Now let $S1' \leftarrow (S_2)^{(y' + \sum_{j=1}^{k} y_j m_j)}$; then we can easily see that (S_1', S_2) is a valid Waters signature on M per (2), so

$$e\big(S_1', g\big) \cdot e\left(S_2, \left(u' \prod_{j=1}^{k} u_j^{m_j}\right)\right)^{-1} = A.$$

Now set $S_1'' \leftarrow S_1 / S_1'$. Dividing the former displayed equation by the latter shows that

$$e\big(S_1'', g\big) \cdot e\left(S_2, \prod_{i=1}^{l} \left(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}}\right)\right)^{-1} = \prod_{i=1}^{l} A_i,$$

i.e., that (S_1'', S_2) is a valid l-element sequential aggregate signature under keys $\mathbf{pk}$ on messages $\mathbf{M}$; and we can re-randomize (S_1'', S_2) to obtain a uniformly distributed sequential aggregate under the same keys on the same messages. The signer can now add a signature on the message $M : M'$ to this aggregate just as before.

The proof of security below can be modified to take this into account. For an aggregate (S_1, S_2), suppose the adversary, algorithm $\mathcal{A}$, asks its challenger to replace the signature under the challenge key from one on message M to one on $M : M'$. The reduction algorithm $\mathcal{B}$ queries its Waters signature oracle on $M : M'$, obtaining a sequential aggregate on messages $(M : M')$ under keys (pk); using its knowledge of the certified private keys, it then constructs the rest of the required aggregate by adding to σ, for each signer $\mathbf{pk}[i]$, the appropriate signature on message $\mathbf{M}[i]$ using algorithm WSA.ASig. Note also that the adversary must previously have queried its signing oracle at M, since otherwise (S_1, S_2) would already constitute a nontrivial forgery.

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. [11] take $l + 1$ pairings to verify when the aggregate includes l signers.

3.3. *Proof of Security*

Theorem 3.1. *The WSA sequential aggregate signature scheme is* $(t, q_C, q_S, n, \epsilon)$*-unforgeable if the Waters signature scheme is* (t', q', ϵ')*-unforgeable on* $\mathbb{G}$*, where*

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

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 Waters signature scheme. Given the challenge Waters 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', providing also its corresponding private key sk'. 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.[5]

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 $\mathbf{M}$ under keys $\mathbf{pk}$. The simulator first checks that the signature σ' verifies; that each key in $\mathbf{pk}$ has been certified; that the challenge key does not appear in $\mathbf{pk}$; and that $|\mathbf{pk}| < n$. If any of these conditions does not hold, $\mathcal{B}$ returns $\perp$.

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 $\mathbf{pk}[i]$, the appropriate signature on message $\mathbf{M}[i]$ using algorithm WSA.ASig. It can do this because it knows—by means of the certification procedure—the private key corresponding to each public key in $\mathbf{pk}$. The result is an aggregate signature σ' on messages $\mathbf{M}\|M$ under keys $\mathbf{pk}\|pk$. 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 $\mathbf{M}$ under keys $\mathbf{pk}$. This forgery must verify as valid under WSA. AVf; each key in $\mathbf{pk}$ (except the challenge key) must have been certified; and $|\mathbf{pk}| \leq n$ must hold. In addition, the forgery must be nontrivial: the challenge key pk must appear in $\mathbf{pk}$, wlog at index 1 (since signature verification in our scheme has no inherent order), and the corresponding message $\mathbf{M}[1]$ 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, $1 \leq i \leq l = |\mathbf{pk}| = |\mathbf{M}|$, parse $\mathbf{pk}[i]$ as $(A_i, u_i', u_{i,1}, \ldots, u_{i,k})$ and $\mathbf{M}[i]$ 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, $2 \leq i \leq l$, let $(\alpha_i, y_i', y_{i,1}, \ldots, y_{i,k})$ be the private key corre-

[5] As noted above, for our signature scheme we can verify that a private key $sk = (\alpha, y', \mathbf{y})$ matches a public key $pk = (A, u', u_1, \ldots, u_k)$ and, indeed, that (pk, sk) is in the image of WSA.Kg, by checking that $A \stackrel{?}{=} e(g,g)^{\alpha}$, that $u' \stackrel{?}{=} g^{y'}$, and that $u_i \stackrel{?}{=} g^{y_i}$ for $1 \leq i \leq k$.

sponding to $\mathbf{pk}[i]$. Algorithm $\mathcal{B}$ computes

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

We now have

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

So (S_1, S_2) is a valid Waters signature on $M^* = \mathbf{M}[1] = (m_{1,1}, \ldots, m_{1,k})$ under key $\mathbf{pk}[1] = pk$. 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 Waters 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 $\mathcal{A}$'s, 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. □

3.4. *A More Efficient Variant in the Random Oracle Model*

Our scheme as described in Sect. 3.2 implicitly uses the Waters hash $H(m_1, \ldots, m_k) = u' \prod_{i=1}^{k} u_i^{m_i}$. It is also possible to instantiate it with the Boneh–Boyen hash $H(M) =$

$u^{H_0(M)}h$, where H_0 maps $\{0,1\}^*$ to $\mathbb{Z}_p$ and is treated as a random oracle. (This derives from Boneh and Boyen's suggested conversion, in the random oracle model, of their selective-ID IBE to a fully secure one [9, Theorem 7.2], to which we then apply the Naor transform recorded by Boneh and Franklin [10] to obtain a signature.)

In this variant, each user picks $x, y, \alpha \xleftarrow{R} \mathbb{Z}_p$ and publishes $u = g^x$, $h = g^y$, and $A = e(g,g)^\alpha$. Public key sizes are thus much smaller than in our Waters-hash–based scheme.

Compared to the scheme of Boneh et al. [11], whose proof of security is also in the random oracle model, our variant scheme is sequential, guarantees security in a weaker (certified-key) security model, and has somewhat longer public keys and signatures. On the other hand, verification in our variant scheme requires only a constant number of pairings rather than $l + 1$ for an l-user aggregate as in BGLS.

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 [24], and have been the subject of much research [7,36,37]. The first multisignatures in which signatures could be combined into a multisignature without interaction were proposed by Boldyreva [7], based on BLS signatures [12]. Below, we present another non-interactive multisignature scheme, based on the Waters signature scheme, which is provably secure without random oracles.

4.1. *Definitions*

A multisignature scheme includes five algorithms. Three of these, Kg, Sig, and Vf, are analogous to those in ordinary signature schemes. The randomized key-generation algorithm Kg outputs a public–private keypair (pk, sk). The randomized signing algorithm Sig takes a private key sk and a message $M \in \{0,1\}^*$ and outputs a signature σ. The verification algorithm Vf takes a public key pk, a message M, and a signature σ, and outputs a bit: 1 if the signature is valid, 0 otherwise.

The two remaining algorithms provide the multisignature functionality. The first, Comb, combines l ordinary signatures, all on a common message M but each under a different key, into a single multisignature that stands for all the input signatures. More formally, Comb is a randomized algorithm that takes the l public key–signature pairs $\{pk_i, \sigma_i\}_{i=1}^l$ along with the message $M \in \{0,1\}^*$ and outputs a multisignature σ or, if combining the signatures failed, $\perp$. We stress that the combination algorithm requires the public keys of all the users, not just the signatures themselves.

The second algorithm, MVf, performs multisignature verification. It takes the l public keys $\{pk_i\}_{i=1}^l$; the common message M; and the multisignature σ that purportedly stands for signatures on M under each of the keys, and outputs a bit: 1 if the multisignature is valid, 0 otherwise.

We add the restriction that neither the combination algorithm nor the multisignature verification algorithm allows a single signer's key to appear more than once in the key list $\{pk_i\}_{i=1}^l$.

A multisignature scheme, instantiated using these algorithms, is correct if all properly-generated signatures and multisignatures verify. More formally, for all signer keypairs (pk, sk) and (pk_i, sk_i) output by Kg, all messages M, and all $l \geq 1$, the following hold with probability 1:

$$\mathsf{Vf}\big(pk, M, \mathsf{Sig}(sk, M)\big) = 1,$$

$$\mathsf{MVf}\big(\{pk_i\}_{i=1}^{l}, M, \mathsf{Comb}\big(\{\, pk_i, \mathsf{Sig}(sk_i, M)\}_{i=1}^{l}, M\big)\big) = 1.$$

4.2. *Security Model*

Micali, Ohta, and Reyzin [31] gave the first formal treatment of multisignatures. We prove security in a variant of the Micali–Ohta–Reyzin model due to Boldyreva [7]. 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. Furthermore, an extractable knowledge-of-secret-key protocol is not required; the multisignature we propose can be proved secure if users provide a lighter-weight "proof of possession" of their private keys. Indeed, Ristenpart and Yilek [41, Sect. 4.2] provide an efficient non-interactive proof-of-possession scheme for our Waters-signature based multisignature; the proof of possession is essentially a Waters signature on the user's public key using a second set of Waters hash exponents shared by all users. Ristenpart and Yilek's Theorem 4.2 shows that our multisignature, augmented by their proof of possession, is secure without requiring adversaries to disclose their secret keys.

More formally, the advantage of an adversary in forging a multisignature is the probability that the challenger outputs 1 in the following game:

Setup. The challenger generates a challenge keypair $(pk, sk) \xleftarrow{R} \mathsf{Kg}$. It runs the adversary, providing to it the public key pk.

Signature queries. The adversary can request ordinary signatures under the challenge key and a message M of his choice. The challenger computes a signature σ as $\mathsf{Sig}(sk, M)$ and returns σ to the adversary.

Note that no multisigning oracle is provided to the adversary. Combining individual signatures into a multisignature requires no knowledge of secret information, and the adversary is expected to carry this step out on its own.

Output. Finally, the adversary halts, having output a multisignature forgery σ^* on some message M^* under public keys $pk_1^*, \ldots, pk_l^*$; l, the number of keys, is up to the adversary. One of the keys, say at index i^*, must be the challenge key pk: $pk_{i^*}^* = pk$. For multisignatures in which the key order is not important (such as our proposed scheme, below) we may assume, without loss of generality, that $i^* = 1$.

In addition, to implement key certification, we require the adversary to emit the private keys $\{sk_i^*\}_{i \neq i^*}$ for all keys except the challenge key.[6]
If no key appears more than once in the key list $\{pk_i^*\}$; if the challenge key appears in the key list $\{pk_i^*\}$ at some index i^*; if each of the other public keys matches the corresponding private key; if the adversary did not query the signing oracle at M^*; and if the multisignature verifies (i.e., $\mathsf{MVf}(\{pk_i^*\}_{i=1}^l, M^*, \sigma^*) = 1$), the challenger outputs 1; otherwise, the challenger outputs 0.

A multisignature scheme is (t, q_S, ϵ)-unforgeable if no t-time adversary making q_S signing queries can win the above game with probability more than ϵ.

4.3. *Our Scheme*

We describe the multisignature obtained from the Waters signature scheme. 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 $\mathsf{WM} = (\mathsf{WM.Kg}, \mathsf{WM.\ Sig}, \mathsf{WM.Vf}, \mathsf{WM.Comb}, \mathsf{WM.MVf})$, 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 $A_i \in \mathbb{G}_T$ and her signature σ_i as $(S_1^{(i)}, S_2^{(i)}) \in \mathbb{G}^2$; parse the message M as a bitstring $(m_1, \ldots, m_k) \in \{0,1\}^k$. Check that no public key occurs twice in $\{pk_i\}$, and verify each signature using WM.Vf; if some key is repeated or any signature is invalid, output $\perp$ and halt. Otherwise, compute

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

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

WM.MVf$(\{pk_i\}_{i=1}^l, M, \sigma)$**.** 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 $A_i \in \mathbb{G}_T$, 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 no key occurs twice in $\{pk_i\}$ and that

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

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

[6] A private key matches a public key if (pk, sk) is in the image of Kg. For some signature schemes (including the multisignatures we propose—see below), it is possible to check this relation directly. For others, the requirement of private key disclosure should be implemented by having the adversary disclose the coins it provided to the key generation algorithm.

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

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

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

Incremental Combination of Multisignatures In the discussion above, we considered signature combination in a multisignature as a one-time operation: The signature combination algorithm provides no way to add additional signatures to an existing multisignature or combine two multisignatures (on the same message) into one. In fact, our scheme does allow such incremental signature combination, as do other multisignatures, including the Boldyreva's multisignature [7], as well as the BGLS aggregate signature [11]. See Mykletun, Narasimha, and Tsudik [32] for a discussion. In the case of multisignatures, this is simply an interface issue. The security model and the proof of security need not change.

4.4. *Proof of Security*

Theorem 4.1. *The WM multisignature scheme is* (t, q, ϵ)*-unforgeable if the Waters signature scheme is* (t', q', ϵ')*-unforgeable, where*

$$t' = t + O(q) \quad \textit{and} \quad q' = q \quad \textit{and} \quad \epsilon' = \epsilon.$$

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 Waters signature scheme. Algorithm $\mathcal{B}$ is given a Waters 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 $\mathcal{B}$ checks that $A^{(1)}$ equals A, the challenge key; that each private key matches the corresponding public key, by verifying that $A^{(i)} \stackrel{?}{=} e(g, g)^{\alpha^{(i)}}$ holds for $2 \le i \le l$; that no key that occurs appears twice in $\{A^{(i)}\}$; and (using WM.MVf) that (S_1^*, S_2^*) is a valid signature on M^* under keys $\{A^{(i)}\}$, and that $\mathcal{A}$ did not query its signing oracle at M^*. If any of these conditions is not satisfied, $\mathcal{A}$ has failed to provide a valid forgery, and $\mathcal{B}$ declares failure as well.

Otherwise, algorithm $\mathcal{B}$ sets $S \leftarrow S_1^* / \prod_{i=2}^{l} g^{\alpha^{(i)}}$. Then we have

$$e(S, g) \cdot e\left(S_2, u' \prod_{i=1}^{k} u_i^{m_i}\right)^{-1} = e(S_1, g) \cdot e\left(S_2, u' \prod_{i=1}^{k} u_i^{m_i}\right)^{-1} \cdot \prod_{i=2}^{l} e(g, g)^{-\alpha^{(i)}}$$

$$= \prod_{i=1}^{l} A^{(i)} \cdot \prod_{i=2}^{l} A^{-(i)} = A^{(1)} = A,$$

so (S, S_2) is a valid Waters 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. □

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. [11] introduced verifiably encrypted signatures, gave a security model for them, and constructed a scheme satisfying the definitions, based on BLS signatures.

The security model proposed by Boneh et al. has been revisited by Hess [23] and by Rückert and Schröder [42]. We prove security in a variant of Boneh et al.'s model that takes into account the "extractability" requirement of Rückert and Schröder.

Below, we recall the definition of verifiably encrypted signatures, formally specify the security model we use, and 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.

5.1. *Definitions*

A verifiably encrypted signature includes seven algorithms. Three of these, Kg, Sig, and Vf, are analogous to those in ordinary signature schemes. The randomized key-generation algorithm Kg outputs a public–private keypair (pk, sk). The randomized

signing algorithm Sig takes a private key sk and a message $M \in \{0,1\}^*$ and outputs a signature σ. The verification algorithm Vf takes a public key pk, a message M, and a signature σ, and outputs a bit: 1 if the signature is valid, 0 otherwise.

The remaining four algorithms provide the verifiably encrypted signature functionality, and bring into the system the trusted third party, which is called the adjudicator. The randomized algorithm AKg generates the adjudicator's public–private keypair, (apk, ask). The adjudicator's public key is made available to users who use it to generate verifiably encrypted signatures. The randomized algorithm for this, ESig, takes a user's private key sk, an adjudicator's public key apk, and a message $M \in \{0,1\}^*$, and outputs a verifiably encrypted signature η. Other users then use the verification algorithm EVf to check the validity of η. This algorithm takes the signer's public key pk, the adjudicator's public key apk, the message M, and the purported verifiably encrypted signature η, and outputs a bit: 1 if the verifiably encrypted signature is valid, 0 otherwise. Finally, the adjudication algorithm, Adj, is used by the adjudicator to recover the (ordinary) signature encrypted in η. The algorithm takes the adjudicator's private key ask, the signer's public key pk, the message M, and the purported encrypted signature η, and outputs either the underlying ordinary signature σ or $\perp$ if extraction failed.

A verifiably encrypted signature, instantiated using these algorithms, is correct if all properly-generated signatures verify, and properly-generated verifiably encrypted signatures verify (as verifiably encrypted signatures) and, when extracted, lead to ordinary signatures that verify (as ordinary signatures). More formally, for all signer keypairs (pk, sk) output by Kg, for all adjudicator keypairs (apk, ask) output by AKg, and for all M, the following hold with probability 1:

$$\mathsf{Vf}\big(pk, M, \mathsf{Sig}(sk, M)\big) = 1,$$

$$\mathsf{EVf}\big(pk, apk, M, \mathsf{ESig}(sk, apk, M)\big) = 1,$$

$$\mathsf{Vf}\big(pk, M, \mathsf{Adj}\big(ask, pk, M, \mathsf{ESig}(sk, apk, M)\big)\big) = 1.$$

5.2. *Security Model*

Informally, we would like a secure verifiably encrypted signature scheme to satisfy, in addition to the security required of ordinary signature schemes, the following property: Any verifiably signature that verifies as valid (using EVf) can be adjudicated, and the resulting signature verifies as valid (using Vf).

In the paper introducing verifiably encrypted signatures, Boneh et al. [11] specified two formal security properties, besides correctness, intended to capture this informal desideratum: 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 a valid message and encrypted signature pair (M^*, η^*) such that he did not query his signing oracle at M^*; in the opacity game his goal is to output a valid message and *ordinary* signature pair (M^*, σ^*) such that he did not 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.

Rückert and Schröder [42] observe that, to capture the informal desideratum above, the original security model of Boneh et al. is missing an important third property, which they called extractability: every verifiably encrypted signature η that verifies as valid on some message M under some user's public key pk and adjudicator public key apk should yield a valid signature when adjudicated: i.e., if $\mathsf{EVf}(pk, apk, M, \eta)$ accepts then so does $\mathsf{Vf}(pk, M, \sigma)$, where $\sigma = \mathsf{Adj}(ask, pk, M, \eta)$. This is a stronger requirement than the correctness requirement for the verifiably encrypted signature scheme, since extractability must hold even when η and pk are maliciously generated.

Further enhancements to the security model for verifiably encrypted signatures have been proposed that we do not address here. Hess [23] observed that the original scheme of Boneh et al. is vulnerable to related-key attacks on adjudication in a multi-user setting, and proposed that signatures be computed over $H(pk\|M)$ rather than $H(M)$. An analogous attack in this model and an analogous defense apply to our scheme. Rückert and Schröder [42] further consider an attack in which a corrupt adjudicator frames an honest user by forging a verifiably encrypted signature from that user; they argue that our WVES verifiably encrypted signature and the verifiably encrypted signature of Boneh et al. are secure against such framing attacks.

5.2.1. *Oracles for the Security Games*

We now formally define the games for unforgeability, extractability, and opacity. In all three games, the challenger generates an adjudicator keypair $(apk, ask) \xleftarrow{R} \mathsf{AKg}$; in the unforgeability and opacity games, the challenger also generates a signer keypair $(pk, sk) \xleftarrow{R} \mathsf{Kg}$. The public keys from these keypairs are supplied to the adversary. The adversary in each of the games is given access to certain of the following oracles:

Verifiably encrypted signing queries for the challenge user. The adversary provides a message $M \in \{0,1\}^*$. The challenger computes $\sigma \xleftarrow{R} \mathsf{ESig}(sk, apk, M)$ and responds to the adversary with σ.

Adjudication queries for the challenge user. The adversary provides a message M and a verifiably encrypted signature η. The challenger first checks that η is a valid verifiably encrypted signature on M under the challenge user key pk and challenge adjudicator key apk, by checking that $\mathsf{EVf}(pk, apk, M, \eta) = 1$. If η is invalid, the challenger responds with $\perp$. Otherwise, it computes $\sigma \leftarrow \mathsf{Adj}(ask, pk, M, \eta)$ and responds to the adversary with σ.

Adjudication queries for an adversarially chosen user. In this variant of the previous oracle, used in the extractability game (in which the challenger does not provide the adversary with a challenge user key), the adversary provides a message M, a user public key pk', and a verifiably encrypted signature η. The challenger first checks that η is a valid verifiably encrypted signature on M under the provided user key pk' and the challenge adjudicator key apk, by checking that $\mathsf{EVf}(pk', apk, M, \eta) = 1$. If η is invalid, the challenger responds with $\perp$. Otherwise, it computes $\sigma \leftarrow \mathsf{Adj}(ask, pk', M, \eta)$ and responds to the adversary with σ.

Note that no ordinary signing oracle is provided for the challenge user key pk: The adversary can obtain ordinary signatures under this key by calling the verifiably encrypted signing oracle and the adjudication oracle in succession.

5.2.2. *Unforgeability*

Formally, the forging advantage of an algorithm $\mathcal{A}$ against a verifiably encrypted signature scheme is the probability that the challenger outputs 1 in the following game:

Setup. The challenger generates an adjudicator keypair $(apk, ask) \xleftarrow{R} \mathsf{AKg}$ and a signer keypair $(pk, sk) \xleftarrow{R} \mathsf{Kg}$ and runs algorithm $\mathcal{A}$ with pk and apk as input.
Verifiably encrypted signing queries for the challenge user. As specified above.
Adjudication queries for the challenge user. As specified above.
Output. Eventually, $\mathcal{A}$ halts, outputting a verifiably encrypted forgery η^* on a message M^* under challenge keys pk and apk. If $\mathsf{EVf}(pk, apk, M^*, \eta^*) = 1$ and the adversary never queried either of its oracles at M^*, the challenger outputs 1; otherwise, the challenger outputs 0.[7]

A verifiably encrypted signature scheme is (t, q_S, q_A, ϵ)-unforgeable if no t-time adversary making q_S verifiably encrypted signing queries and q_A adjudication queries can win the above game with probability more than ϵ.

5.2.3. *Extractability*

Formally, the extraction advantage of an algorithm $\mathcal{A}$ against a verifiably encrypted signature scheme is the probability that the challenger outputs 1 in the following game:

Setup. The challenger generates an adjudicator keypair $(apk, ask) \xleftarrow{R} \mathsf{AKg}$ and runs algorithm $\mathcal{A}$ with apk as input.
Adjudication queries for an adversarially chosen user. As specified above.
Output. Eventually, $\mathcal{A}$ halts, outputting three values: a user public key pk^*; a message M^*; and a verifiably encrypted forgery η^*. If η^* is a valid verifiably encrypted signature on M^* under user key pk^* and the challenge adjudication key apk (i.e., if $\mathsf{EVf}(pk^*, apk, M^*, \eta^*) = 1$) but either the signature cannot be adjudicated (i.e., $\mathsf{Adj}(ask, pk^*, M^*, \eta^*) = \perp$) or, when adjudicated, is not a valid ordinary signature (i.e., $\mathsf{Vf}(pk^*, M^*, \mathsf{Adj}(ask, pk^*, M^*, \eta^*)) = 0$), the challenger outputs 1; otherwise, the challenger outputs 0.

A verifiably encrypted signature scheme is $(t, q_{A'}, \epsilon)$-extractable if no t-time adversary making $q_{A'}$ adjudication queries for an adversarially chosen user can win the above game with probability more than ϵ. A verifiably encrypted signature scheme is unconditionally extractable if no adversary can win the above game at all, regardless of how long it runs or how many adjudication queries for an adversarially chosen user it makes.

[7] Note that the important nontriviality restriction is that the adversary never queried the verifiably encrypted signing oracle at M^*. There is no reason for it to have queried its adjudication oracle at M^*: If the verifiably encrypted signature provided to that oracle is valid, the adversary could instead output it, winning the game; if the provided signature is invalid, the oracle returns $\perp$; and the adversary can run the EVf algorithm itself to check whether or not the signature is valid.

5.2.4. *Opacity*

Finally, the opacity advantage of an algorithm $\mathcal{A}$ against a verifiably encrypted signature scheme is the probability that the challenger outputs 1 in the following game:

Setup. The challenger generates an adjudicator keypair $(apk, ask) \xleftarrow{R} \mathsf{AKg}$ and a signer keypair $(pk, sk) \xleftarrow{R} \mathsf{Kg}$ and runs algorithm $\mathcal{A}$ with pk and apk as input.

Verifiably encrypted signing queries for the challenge user. As specified above.

Adjudication queries for the challenge user. As specified above.

Output. Eventually, $\mathcal{A}$ halts, outputting an *ordinary* forged signature σ^* on a message M^* under challenge key pk. If $\mathsf{Vf}(pk, M^*, \eta^*) = 1$ and the adversary never queried its adjudication oracle at M^*, the challenger outputs 1; otherwise, the challenger outputs 0.

A verifiably encrypted signature scheme is (t, q_S, q_A, ϵ)-opaque if no t-time adversary making q_S verifiably encrypted signing queries and q_A adjudication queries can win the above game with probability more than ϵ.

Note that the opacity game also captures the unforgeability of the underlying ordinary signature scheme. To transform an algorithm that wins the existential unforgeability game (per Goldwasser, Micali, and Rivest [20]) against the underlying signature scheme to one that wins the opacity game with the same probability, one simulates that algorithm's signing oracle by using the verifiably encrypted signing oracle followed by the adjudication oracle; it is easy to see that the resulting forgery is nontrivial.

5.3. *Our Scheme*

Our scheme is a seven-tuple of algorithms $\mathsf{WVES} = (\mathsf{WVES.Kg}, \mathsf{WVES.Sig}, \mathsf{WVES.Vf}, \mathsf{WVES.AKg}, \mathsf{WVES.ESig}, \mathsf{WVES.EVf}, \mathsf{WVES.Adj})$ 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 $\beta \xleftarrow{R} \mathbb{Z}_p$, 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 $M = (m_1, \ldots, m_k)$, compute a signature $(S_1, S_2) \xleftarrow{R} \mathsf{WVES.Sig}(sk, M)$. Pick a random $s \xleftarrow{R} \mathbb{Z}_p$, and compute

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

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 $A \in \mathbb{G}_T$, 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:

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

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

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 $A \in \mathbb{G}_{\mathrm{T}}$. Parse the message M as $(m_1, \ldots, m_k) \in \{0,1\}^k$. Verify (using WVES. EVf) that the verifiably encrypted signature η is valid, and parse it as $(K_1, K_2, K_3) \in \mathbb{G}^3$. Compute

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

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

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

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),

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

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

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

so the adjudicated signature is indeed a valid one.

5.4. *Proofs of Security*

5.4.1. *Unforgeability*

Theorem 5.1. *The WVES verifiably encrypted signature scheme is* (t, q_S, q_A, ϵ)*-unforgeable if the Waters signature scheme is* (t', q', ϵ')*-unforgeable, where*

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

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 $\beta \xleftarrow{R} \mathbb{Z}_p$, 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 $s \xleftarrow{R} \mathbb{Z}_p$ and computes

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

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 WVES.ESig, except that it uses its signing oracle instead of evaluating WVES. 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 WVES.Adj$(\beta, A, M, (K_1, K_2, K_3))$. 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

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

Then we have

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

and (S_1^*, S_2^*) is therefore a valid Waters signature on M^*. The last equality follows from Equation (8). Since $\mathcal{A}$ 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. □

Rückert and Schröder observe that our WVES verifiably encrypted signature has a property they call "key independence" and prove that all key-independent verifiably encrypted signature schemes are unforgeable if the underlying signature is unforgeable [42]. This gives an another proof for the unforgeability of WVES.

5.4.2. *Extractability*

Theorem 5.2. *The WVES verifiably encrypted signature scheme is unconditionally extractable.*

Proof. We show that every valid verifiably encrypted signature can always be adjudicated, and that the resulting ordinary signature always verifies as valid. Winning in the extractability game requires the adversary to produce a verifiably encrypted signature that cannot be adjudicated to a valid signature, so what we show in fact proves that no adversary can win the extractability game, and that the WVES scheme is unconditionally extractable.

Specifically, we show that, for all adjudicator keypairs (apk, ask), all signer keys A, all messages M, and all verifiably encrypted signatures η, whenever $\mathsf{WVES.EVf}(pk, apk, M, \eta)$ accepts, so does $\mathsf{WVES.Vf}(pk, M, \sigma)$, where $\sigma = \mathsf{WVES.Adj}(ask, pk, M, \eta)$.

Let $u', u_1, \ldots, u_k \in G$ be the shared random generators included in the system parameters. Let the adjudicator's private key be $ask = \beta \in \mathbb{Z}_p$, and let his public key be $apk = v = g^\beta \in \mathbb{G}$.

Consider a public key $pk = A \in \mathbb{G}_T$, a message $M = (m_1, \ldots, m_k) \in \{0,1\}^k$, and a verifiably encrypted signature $\eta = (K_1, K_2, K_3) \in \mathbb{G}^3$. The adversary may have generated these maliciously.

By the definition of WVES.EVf, the condition that $\mathsf{WVES.EVf}(pk, apk, M, \eta) = 1$ means that

$$e(K_1, g) \cdot e\left(K_2, u' \prod_{i=1}^{k} u_i^{m_i}\right)^{-1} \cdot e(K_3, v)^{-1} = A.$$

Given the verifiably encrypted signature η, WVES.Adj chooses $s \xleftarrow{R} \mathbb{Z}_p$ and sets

$$S_1' \leftarrow K_1 \cdot K_3^{-\beta} \cdot \left(u' \prod_{i=1}^{k} u_i^{m_i}\right)^s \quad \text{and} \quad S_2' \leftarrow K_2 \cdot g^s.$$

We will show that the pair (S_1', S_2') is a valid signature on M under key pk, as required. Indeed,

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

where the last equality follows from (8), the verifiably encrypted signature verification equation for (K_1, K_2, K_3). But we have just derived (2), the verification equation for Waters signatures, showing that (S_1', S_2') is a valid Waters signature under public key A. □

5.4.3. *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 [16] showed that this assumption, introduced by Boneh et al. [11], is equivalent to CDH.

Theorem 5.3 (Coron–Naccache [16]). *The aggregate extraction and Computational Diffie–Hellman problems are Karp reducible to each other with $O(1)$ computation.*[8]

Theorem 5.4. *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*

$$t' = t + O(q_S + q_A) \quad \text{and} \quad \epsilon' = 8q_A(k+1)\epsilon.$$

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.

[8] 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.

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 $v \leftarrow g^\beta$, and $A \leftarrow e(g,g)^{\alpha\gamma}$.

Let λ be a parameter to be optimized later and small enough that $k\lambda \ll p$. Algorithm $\mathcal{B}$ picks $\kappa \xleftarrow{R} \{0,\ldots,k\}$, $x', x_1, \ldots, x_k \xleftarrow{R} \mathbb{Z}_\lambda = \{0,\ldots,\lambda-1\}$ and $y', y_1, \ldots, y_k \xleftarrow{R} \mathbb{Z}_p$ and sets

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

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, $g^{\alpha\gamma}$, is not known to $\mathcal{B}$.

Verifiably encrypted signing queries. Algorithm $\mathcal{A}$ requests a verifiably encrypted signature on $M = (m_1,\ldots,m_k) \in \{0,1\}^k$ under challenge key A and adjudicator key v. Algorithm $\mathcal{B}$ returns to $\mathcal{A}$ a verifiably encrypted signature (K_1, K_2, K_3), which it computes by choosing $r, s \xleftarrow{R} \mathbb{Z}_p$ and setting

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

This is a Waters signature with randomness r, encrypted with randomness $\delta + s$; with r and s uniformly chosen (K_1, K_2, K_3) are distributed exactly as in the real system.

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 $J = y' + \sum_{i=1}^k y_i m_i$; observe that $u' \prod_{i=1}^k u_i^{m_i} = (g^\gamma)^F g^J$. If $F \equiv 0 \mod \lambda$ algorithm $\mathcal{B}$ declares failure and halts. Otherwise, it proceeds as follows. It picks $r \xleftarrow{R} \mathbb{Z}_p$ and sets

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

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

$$S_1 = \left(g^\alpha\right)^{-J/F} \left(u' \prod_{i=1}^k u_i^{m_i}\right)^r = g^{\alpha\gamma} \left(\left(g^\gamma\right)^F g^J\right)^{-\alpha/F} \left(\left(g^\gamma\right)^F g^J\right)^r$$

$$= g^{\alpha\gamma} \left(u' \prod_{i=1}^k u_i^{m_i}\right)^{\tilde{r}},$$

where for the second equality we have multiplied and divided by $g^{\alpha\gamma}$. (This is the "Boneh–Boyen trick" [9].) Algorithm $\mathcal{B}$ returns (S_1, S_2) to $\mathcal{A}$ as the answer to the adjudication query.

Note that $\mathcal{B}$ does not decrypt the encrypted signature (K_1, K_2, K_3) given by $\mathcal{A}$ as input to the adjudication oracle; instead, it generates a randomly-distributed valid Waters signature on the same message. We must therefore argue that the distribution of $\mathcal{B}$'s responses is the same as it would be if $\mathcal{B}$ performed the adjudication using WVES.Adj and the adjudicator's private key. Theorem 5.2 shows that, whenever (K_1, K_2, K_3) is a valid encrypted signature on M, the output of WVES.Adj, computed by choosing $s \xleftarrow{R} \mathbb{Z}_p$ and setting

$$S_1' \leftarrow K_1 \cdot K_3^{-\beta} \cdot \left(u' \prod_{i=1}^{k} u_i^{m_i}\right)^s \quad \text{and} \quad S_2' \leftarrow K_2 \cdot g^s,$$

satisfies the Waters signature verification equation (2):

$$e\big(S_1', g\big) \cdot e\left(S_2', u' \prod_{i=1}^{k} u_i^{m_i}\right)^{-1} = A. \tag{9}$$

Let $S_2' = g^{\tilde{s}}$ for some $\tilde{s} \in \mathbb{Z}_p$. (Here we are relying on the group membership test applied to K_1, K_2, and K_3 to be sure that S_2', computed as above, is in fact in $\mathbb{G} = \langle g \rangle$.) Then we can rewrite (9) as

$$e\big(S_1', g\big) = e\left(g^{\alpha\gamma} \left(u' \prod_{i=1}^{k} u_i^{m_i}\right)^{\tilde{s}}, g\right).$$

Since $e(\cdot, g)$ is an isomorphism from $\mathbb{G}$ to $\mathbb{G}_T$ (in fact, it is the Menezes–Okamoto–Vanstone map [30]), this shows that $S_1' = g^{\alpha\gamma}(u' \prod_{i=1}^{k} u_i^{m_i})^{\tilde{s}}$, which means that S_1' is a Waters signature on M, as output by W.Sig, with randomness $\tilde{s}$. Further, there is a one-to-one correspondence between choices of r, $\tilde{r}$, $\tilde{s}$, and s. Thus the three distributions—(1) the output of $\mathcal{B}$ as above using the Boneh–Boyen trick; (2) the output were $\mathcal{B}$ following WVES.Adj; and (3) the output of W.Sig on M—are all identical.

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^*, which means that M^* differs in at least one coordinate from each message $M^{(i)}$ on which $\mathcal{A}$ queried the adjudication oracle. 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 $F^* \neq 0$, $\mathcal{B}$ declares failure and exits. Otherwise, we have $u' \prod_{i=1}^{k} u_i^{m_i^*} = g^{J^*}$, so that, by the Waters signature verification equation,

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

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

To complete the proof, we must bound the probability that $\mathcal{B}$ aborts.

Consider a third algorithm, $\mathcal{B}'$, which is given the secret values α, β, γ, and δ from the aggregate extraction challenge. Algorithm $\mathcal{B}'$ interacts with $\mathcal{A}$ as algorithm $\mathcal{B}$ does, except for the following differences:

1. Whenever $\mathcal{B}$ would have aborted, $\mathcal{B}'$ instead uses its secret knowledge to compute and return the answer; and
2. Once $\mathcal{A}$ terminates, $\mathcal{B}'$ outputs 0 if $\mathcal{A}$'s queries or Waters signature forgery would have caused $\mathcal{B}$ to abort; 1 otherwise.

(Equivalently, $\mathcal{B}'$ could use its secret knowledge to answer all of $\mathcal{A}$'s queries; these answers would be distributed identically to those $\mathcal{B}$ gives, as we have argued above.) Clearly $\mathcal{A}$ cannot distinguish whether it is interacting with $\mathcal{B}$ or $\mathcal{B}'$ so long as neither aborts. We will give an upper bound for the probability that $\mathcal{B}'$ outputs 0, which will also bound the probability that $\mathcal{B}$ aborts when interacting with $\mathcal{A}$.

Because algorithm $\mathcal{B}'$ uses the values κ, x', and $x_1, \ldots, x_k$ only after $\mathcal{A}$ has halted, to decide whether the interaction would have caused $\mathcal{B}$ to abort, it is clear that $\mathcal{A}$ can learn no information about these values. Moreover, it is only these values that determine whether $\mathcal{B}$ aborts (equivalently, whether $\mathcal{B}'$ outputs 0).

We will, equivalently, give a lower bound on the probability that $\mathcal{B}'$ does not abort regardless of the adversary's sequences of message queries. Different sequences of message queries lead to different success probabilities; the usual argument that relies on full independence between aborts for different messages does not apply. (This is why, in the identity-based encryption context, Waters' original analysis required an artificial abort stage [44]; see Bellare and Ristenpart [6] for further discussion and an improved analysis.) Instead, we will give a lower bound on the success probability that holds for any message sequence. Specifically, we will consider the following combinatorial problem:

> Let M^* stand for $(m_1^*, \ldots, m_k^*)$ and define $F^* = -\kappa\lambda + x' + \sum_{i=1}^k x_i m_i^*$; for $1 \le j \le q_A$, let $M^{(j)}$ stand for $(m_1^{(j)}, \ldots, m_k^{(j)})$ and define $F^{(j)} = -\kappa\lambda + x' + \sum_{i=1}^k x_i m_i^{(j)}$. What is the minimum probability for any choice of M^* and $\{M^{(j)}\}_{1 \le j \le q_A}$, all distinct, that $F^* = 0$ and, for $1 \le j \le q_A$, $F^{(j)} \not\equiv 0 \bmod \lambda$, where the probability is over the random choice of κ from $\{0, \ldots, k\}$ and x' and $x_1, \ldots, x_k$ from $\{0, \lambda - 1\}$?

Let $\mathcal{E}^{(j)}$ be the event that $F^{(j)} \equiv 0 \mod \lambda$ for each j, and let $\mathcal{E}^*$ be the event that $F^* \equiv 0 \mod \lambda$. Observe that $\Pr[\mathcal{E}^{(j)}] = \Pr[\mathcal{E}^*] = 1/\lambda$ since each x' and $x_1, \ldots, x_k$ is independently random in $\{0, \ldots, \lambda - 1\}$; furthermore, the events are pairwise independent because the messages are all distinct, so one event in any pair will involve an

x_i value not included in the other. Following Waters' analysis, we have

$$\begin{aligned}
\Pr\left[\bigwedge_{j=1}^{q_A} \neg\mathcal{E}^{(j)} \wedge \mathcal{E}^*\right] &= \Pr\left[\bigwedge_j \neg\mathcal{E}^{(j)}\right] \cdot \Pr\left[\mathcal{E}^* \,\middle|\, \bigwedge_j \neg\mathcal{E}^{(j)}\right] \\
&= \Pr\left[\bigwedge_j \neg\mathcal{E}^{(j)}\right] \cdot \frac{\Pr[\mathcal{E}^*]}{\Pr[\bigwedge_j \neg\mathcal{E}^{(j)}]} \cdot \Pr\left[\bigwedge_j \neg\mathcal{E}^{(j)} \,\middle|\, \mathcal{E}^*\right] \\
&\geq \Pr\left[\bigwedge_j \neg\mathcal{E}^{(j)}\right] \cdot \Pr[\mathcal{E}^*] \cdot \Pr\left[\bigwedge_j \neg\mathcal{E}^{(j)} \,\middle|\, \mathcal{E}^*\right] \\
&= \left(1 - \Pr\left[\bigvee_j \mathcal{E}^{(j)}\right]\right) \cdot \Pr[\mathcal{E}^*] \cdot \left(1 - \Pr\left[\bigvee_j \mathcal{E}^{(j)} \,\middle|\, \mathcal{E}^*\right]\right) \\
&\geq \left(1 - \sum_j \Pr[\mathcal{E}^{(j)}]\right) \cdot \Pr[\mathcal{E}^*] \cdot \left(1 - \sum_j \Pr[\mathcal{E}^{(j)} \mid \mathcal{E}^*]\right) \\
&= \left(1 - \frac{q_A}{\lambda}\right)\left(\frac{1}{\lambda}\right)\left(1 - \frac{q_A}{\lambda}\right) \geq \left(\frac{1}{\lambda}\right)\left(1 - \frac{2q_A}{\lambda}\right).
\end{aligned}$$

Here the second line follows from an application of Bayes' law, the fifth from two applications of the union bound.

Next, observe that $\Pr[F^* = 0 \mid \bigwedge_{j=1}^{q_A} \neg\mathcal{E}^{(j)} \wedge \mathcal{E}^*] = 1/(k+1)$, since event $\mathcal{E}^*$ guarantees that $x' + \sum_{i=1}^{k} x_i m_i^*$ is a multiple of λ between 0 and $k\lambda$; this is canceled out by the $-\kappa\lambda$ term when κ has the right value, which occurs with probability $1/(k+1)$ since κ is independent of the events $\mathcal{E}^*$ and $\{\mathcal{E}^{(i)}\}$.

Putting this all together and fixing the parameter λ as $\lambda = 4q_A$, we obtain

$$\Pr\left[F^* = 0 \wedge \bigwedge_{j=1}^{q_A} F^{(j)} \not\equiv 0 \bmod \lambda\right] \geq \left(\frac{1}{k+1}\right)\left(\frac{1}{\lambda}\right)\left(1 - \frac{2q_A}{\lambda}\right) = \frac{1}{8q_A(k+1)}.$$

This shows that if $\mathcal{A}$ succeeds in breaking the opacity of the WVES verifiably encrypted signature with probability ϵ, $\mathcal{B}$ succeeds in breaking the aggregate extraction assumption with probability at least $\epsilon/(8q_A(k+1))$. Algorithm $\mathcal{B}$'s run-time overhead is $O(1)$ to answer each of $\mathcal{A}$'s queries and to compute the final output. □

5.5. *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 Waters signature to a WVES opacity breaker of this sort: simulate a Waters signing oracle by a call to the verifiably encrypted signing oracle followed by a call to the adjudication oracle. Combining this insight with Theorems 5.4 and 5.3 immediately gives the following corollary:

Corollary 5.5 (Waters [44]). *The Waters signature scheme is* (t, q, ϵ)*-unforgeable if Computational Diffie–Hellman is* $(t + O(q), 8q(k + 1)\epsilon)$*-hard on* $\mathbb{G}$*. Here* q *is the number of signing queries.*

Identity-Based Verifiably Encrypted Signatures Bellare, Namprempre, and Neven [4] describe a "folklore" construction for identity-based signatures in which an identity-based signature on a message M under an identity ID consists of an ordinary signature on M under a public key and a certificate from the central authority tying that public key to the identity ID. As Galindo, Herranz, and Kiltz [18] observe, this construction yields an identity-based verifiably encrypted signature if the signature on M is replaced with a verifiably encrypted signature on M. Thus, combining our WVES verifiably encrypted signature (at level two) with the ordinary Waters signature (at level one, for certificates) yields an identity-based verifiably encrypted signature secure under CDH without random oracles.

Alternatively, one could construct an identity-based verifiably encrypted signature directly using a variant of our construction. Waters' IBE [44] can be viewed as an instance of the Boneh–Boyen paradigm for hierarchical IBE [9]. As observed by Gentry and Silverberg [19], a two-level HIBE gives an identity-based signature using a transform analogous to the Naor transform from IBE to signatures [10]. Thus, in particular, the Waters 2-HIBE yields an identity-based signature; this was formalized by Paterson and Schuldt [39]. This Waters identity-based signature can be verifiably encrypted just as we do with the ordinary Waters signature in our WVES construction. The resulting scheme is an identity-based verifiably encrypted signature secure under CDH without random oracles. Verifiably encrypted signatures in the direct construction do not include a certificate for a Waters public key and are thus substantially shorter than those in the generic construction described above.

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 [12, Sect. 3.2].

Sequential Aggregate Signatures We compare our sequential aggregate signature scheme to the aggregate scheme of Boneh et al. [11] (BGLS), to the sequential aggregate signature scheme of Lysyanskaya et al. [28] (LMRS), and to Neven's sequential aggregate signed data scheme, an improved version of LMRS [34].

We instantiate the LMRS scheme using the RSA-based permutation family with common domain devised by Hayashi, Okamoto, and Tanaka [22]. 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. [28], 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

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. Neven's scheme supports message recovery, which can reduce the effective signature overhead.

Scheme	R.O.	Sequential	Key model	Sig. size	Key size	Verification	Signing
BGLS	YES	NO	Chosen	160 bits	1920 bits	$l+1$ pair.	1 exp.
LMRS-1	YES	YES	Chosen	1024 bits	2048 bits	$2l$ exp.	verify + 1 exp.
LMRS-2	YES	YES	Registered	1024 bits	1024 bits	$4l$ mult.	verify + 1 exp.
Neven	YES	YES	Chosen	1184 bits	1024 bits	$2l$ mult.	verify + 1 exp.
Ours	NO	YES	Registered	320 bits	311040 bits	2 pair., $lk/2$ mult.	verify + 1 exp.

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. Neven's scheme improves on that of Lysyanskaya et al.'s by removing the requirement that keys correspond to certified permutations, which allows signers to use $e = 3$ without key certification, and by allowing aggregation to proceed without requiring that keys be scoped, that aggregate signatures grow by a bit with each additional signature (as in the original Lysyanskaya et al.'s construction), or requiring two RSA function applications per signature (as in Lysyanskaya et al.'s construction instantiated with the Hayashi–Okamoto–Tanaka permutation). The downside to Neven's scheme is somewhat longer signatures, though Neven's signatures support message recovery, which can reduce the signature overhead to as little as 160 bits.

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. For the pairing-based schemes, we assume Barreto–Naehrig [3] curves without compression of $\mathbb{G}_2$ element representation.

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 38 KB 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. (The routers participating in BGP are all known to each other and can distribute their public keys ahead of any SBGP conversation.) Additionally, Naccache [33] and Chatterjee and Sarkar [15] independently proposed ways to achieve shorter public keys in the Waters signature scheme. Using these methods we can also achieve considerably shorter public keys. Finally, there are shorter representations for elements of $\mathbb{G}_2$ than the naive choice we consider; see Chatterjee and Menezes [14] for a discussion.

Multisignatures We compare our multisignature scheme to the Boldyreva's multisignature [7]. 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. Note that, unlike with our sequential aggregate signature scheme, all users of our multisignature scheme must share a single set of hash generators u' and $u_1, \ldots, u_k$, making public keys small.

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	Sig. size	Key size	Verification	Signing
Boldyreva	YES	Registered	160 bits	1920 bits	2 pair.	1 exp.
Ours	NO	Registered	320 bits	1920 bits	2 pair., $k/2$ 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	Sig. size	Key size	Verification	Generation
BGLS	YES	Registered	320 bits	1920 bits	3 pair.	3 exp.
Ours	NO	Registered	480 bits	1920 bits	3 pair., $k/2$ mult.	4 exp.

While Neven gives a non-interactive multi-signed data scheme as a variant of his sequential aggregate signed data scheme [34], this scheme does not appear to give a constant-size multisignature when message recovery is not used. Accordingly, we omit this scheme in our comparison. If message recovery is used and the messages signed are sufficiently long, however, Neven's scheme compares favorably to Boldyreva's and ours in signature overhead. Furthermore, unlike Boldyreva's scheme and ours, Neven's is secure in the chosen-key model, even with small public exponent RSA keys.

Verifiably Encrypted Signatures We compare our verifiably encrypted signature scheme to that of Boneh et al. [11] (BGLS). We present the comparisons in 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. Note that, unlike with our sequential aggregate signature scheme, all users of our verifiably encrypted signature scheme can share a single set of hash generators u' and $u_1, \ldots, u_k$, making public keys small.

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 [44]. 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 [28] and has a more efficient verification algorithm than the BGLS aggregate signature scheme [11]. That gives some interesting trade-offs 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 signatures 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 [33] and Chatterjee and Sarkar [15] 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. [11] 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.

Acknowledgements

We thank Dan Boneh, Andrew Bortz, Xavier Boyen, Ilya Mironov, and the anonymous Eurocrypt 2006 and *Journal of Cryptology* reviewers for their helpful comments.

References

[1] N. Asokan, V. Shoup, M. Waidner, Optimistic fair exchange of digital signatures. *IEEE J. Sel. Areas Commun.* **18**(4), 593–610 (2000)

[2] F. Bao, R. Deng, W. Mao, Efficient and practical fair exchange protocols with offline TTP, in *Proceedings of IEEE Security & Privacy*, ed. by P. Karger, L. Gong (1998), pp. 77–85

[3] P. Barreto, M. Naehrig, Pairing-friendly elliptic curves of prime order, in *Proceedings of SAC 2005*, ed. by B. Preneel, S. Tavares. LNCS, vol. 3897 (Springer, Berlin, 2006), pp. 319–331

[4] M. Bellare, C. Namprempre, G. Neven, Security proofs for identity-based identification and signature schemes, in *Proceedings of Eurocrypt 2004*, ed. by C. Cachin, J. Camenisch. LNCS, vol. 3027 (Springer, Berlin, 2004), pp. 268–286

[5] M. Bellare, C. Namprempre, G. Neven, Unrestricted aggregate signatures, in *Proceedings of ICALP 2007*, ed. by L. Arge, C. Cachin, T. Jurdziński, A. Tarlecki. LNCS, vol. 4596 (Springer, Berlin, 2007), pp. 411–422

[6] M. Bellare, T. Ristenpart, Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme, in *Proceedings of Eurocrypt 2009*, ed. by A. Joux. LNCS, vol. 5479 (Springer, Berlin, 2009), pp. 407–424

[7] A. Boldyreva, Threshold signature, multisignature and blind signature schemes based on the gap-Diffie–Hellman-group signature scheme, in *Proceedings of PKC 2003*, ed. by Y. Desmedt. LNCS, vol. 2567 (Springer, Berlin, 2003), pp. 31–46

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

[9] D. Boneh, X. Boyen, Efficient selective-ID secure identity based encryption without random oracles, in *Proceedings of Eurocrypt 2004*, ed. by C. Cachin, J. Camenisch. LNCS, vol. 3027 (Springer, Berlin, 2004), pp. 223–238

[10] D. Boneh, M. Franklin, Identity-based encryption from the Weil pairing. *SIAM J. Comput.* **32**(3), 586–615 (2003). Extended abstract in *Proceedings of Crypto 2001*

[11] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in *Proceedings of Eurocrypt 2003*, ed. by E. Biham. LNCS, vol. 2656 (Springer, Berlin, 2003), pp. 416–432

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

[13] R. Canetti, O. Goldreich, S. Halevi, The random oracle methodology, revisited. *J. ACM* **51**(4), 557–594 (2004)

[14] S. Chatterjee, A. Menezes, On cryptographic protocols employing asymmetric pairings—the role of ψ revisited. Cryptology ePrint Archive, Report 2009/480 (2009). http://eprint.iacr.org/

[15] S. Chatterjee, P. Sarkar, Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model, in *Proceedings of ICISC 2005*, ed. by D. Won, S. Kim. LNCS, vol. 3935 (Springer, Berlin, 2005), pp. 424–440

[16] J.-S. Coron, D. Naccache, Boneh et al.'s k-element aggregate extraction assumption is equivalent to the Diffie–Hellman assumption, in *Proceedings of Asiacrypt 2003*, ed. by C.S. Laih. LNCS, vol. 2894 (Springer, Berlin, 2003), pp. 392–397

[17] S. Galbraith, Pairings, in *Advances in Elliptic Curve Cryptography*, ed. by I.F. Blake, G. Seroussi, N. Smart. London Mathematical Society Lecture Notes, vol. 317 (Cambridge University Press, Cambridge, 2005), pp. 183–213. Chapter IX

[18] D. Galindo, J. Herranz, E. Kiltz, On the generic construction of identity-based signatures with additional properties, in *Proceedings of Asiacrypt 2006*, ed. by X. Lai, K. Chen. LNCS, vol. 4284 (Springer, Berlin, 2006), pp. 178–193

[19] C. Gentry, A. Silverberg, Hierarchical ID-based cryptography, in *Proceedings of Asiacrypt 2002*, ed. by Y. Zheng. LNCS, vol. 2501 (Springer, Berlin, 2002), pp. 548–566

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

[21] J. Groth, R. Ostrovsky, A. Sahai, Perfect non-interactive zero knowledge for NP, in *Proceedings of Eurocrypt 2006*, ed. by S. Vaudenay. LNCS, vol. 4004 (Springer, Berlin, 2006), pp. 339–358

[22] R. Hayashi, T. Okamoto, K. Tanaka, An RSA family of trap-door permutations with a common domain and its applications, in *Proceedings of PKC 2004*, ed. by F. Bao, R.H. Deng, J. Zhou. LNCS, vol. 2947 (Springer, Berlin, 2004), pp. 291–304

[23] F. Hess, On the security of the verifiably encrypted signature scheme of Boneh, Gentry, Lynn and Shacham. *Inf. Process. Lett.* **89**(3), 111–114 (2004)

[24] K. Itakura, K. Nakamura, A public-key cryptosystem suitable for digital multisignatures. *NEC J. Res. Dev.* **71**, 1–8 (1983)

[25] S. Kent, C. Lynn, K. Seo, Secure border gateway protocol (secure-BGP). *IEEE J. Sel. Areas Commun.* **18**(4), 582–592 (2000)

[26] N. Koblitz, A. Menezes, Pairing-based cryptography at high security levels, in *Proceedings of Cryptography and Coding 2005*, ed. by N. Smart. LNCS, vol. 3796 (Springer, Berlin, 2005), pp. 13–36

[27] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, B. Waters, Sequential aggregate signatures and multisignatures without random oracles, in *Proceedings of Eurocrypt 2006*, ed. by S. Vaudenay. LNCS, vol. 4004 (Springer, Berlin, 2006), pp. 465–485

[28] A. Lysyanskaya, S. Micali, L. Reyzin, H. Shacham, Sequential aggregate signatures from trapdoor permutations, in *Proceedings of Eurocrypt 2004*, ed. by C. Cachin, J. Camenisch. LNCS, vol. 3027 (Springer, Berlin, 2004), pp. 74–90

[29] M. Mambo, K. Usuda, E. Okamoto, Proxy signatures for delegating signing operation, in *Proceedings of CCS 1996*, ed. by L. Gong, J. Stearn (ACM, New York, 1996), pp. 48–57

[30] A. Menezes, T. Okamoto, P. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inf. Theory* **39**(5), 1639–1646 (1993)

[31] S. Micali, K. Ohta, L. Reyzin, Accountable-subgroup multisignatures (extended abstract), in *Proceedings of CCS 2001*, ed. by P. Samarati (ACM, New York, 2001), pp. 245–254

[32] E. Mykletun, M. Narasimha, G. Tsudik, Signature bouquets: Immutability for aggregated/condensed signatures, in *Proceedings of ESORICS 2004*, ed. by P. Ryan, P. Samarati. LNCS, vol. 3193 (Springer, Berlin, 2004), pp. 160–176

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

[34] G. Neven, Efficient sequential aggregate signed data, in *Proceedings of Eurocrypt 2008*, ed. by N. Smart. LNCS, vol. 4965 (Springer, Berlin, 2008), pp. 52–69

[35] D. Nicol, S. Smith, M. Zhao, Evaluation of efficient security for BGP route announcements using parallel simulation. *Simul. Model. Pract. Theory* **12**, 187–216 (2004)

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

[37] T. Okamoto, A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Comput. Syst.* **6**(4), 432–441 (1988)

[38] K. Paterson, Cryptography from pairings, in *Advances in Elliptic Curve Cryptography*, ed. by I.F. Blake, G. Seroussi, N. Smart. London Mathematical Society Lecture Notes, vol. 317 (Cambridge University Press, Cambridge, 2005), pp. 215–251. Chapter X

[39] K. Paterson, J. Schuldt, Efficient identity-based signatures secure in the standard model, in *Proceedings of ACISP 2006*, ed. by L. Batten, R. Safavi-Naini. LNCS, vol. 4058 (Springer, Berlin, 2006), pp. 207–222

[40] Y. Rekhter, T. Li, S. Hares, A Border Gateway Protocol 4 (BGP-4). RFC 4271 (draft standard), Jan. 2006

[41] T. Ristenpart, S. Yilek, The power of proofs-of-possession: securing multiparty signatures against rogue-key attacks, in *Proceedings of Eurocrypt 2007*, ed. by M. Naor. LNCS, vol. 4515 (Springer, Berlin, 2007), pp. 228–245

[42] M. Rückert, D. Schröder, Security of verifiably encrypted signatures and a construction without random oracles, in *Proceedings of Pairing 2009*, ed. by H. Shacham, B. Waters. LNCS, vol. 5671 (Springer, Berlin, 2009), pp. 17–34

[43] H. Shacham, *New paradigms in signature schemes*. Ph.D. thesis, Stanford University, 2005

[44] B. Waters, Efficient identity-based encryption without random oracles, in *Proceedings of Eurocrypt 2005*, ed. by R. Cramer. LNCS, vol. 3494 (Springer, Berlin, 2005), pp. 114–127