A Survey of Two Signature Aggregation Techniques

Dan Boneh
dabo@cs.stanford.edu

Craig Gentry
cgentry@docomolabs-usa.com

Ben Lynn
blynn@cs.stanford.edu

Hovav Shacham
hovav@cs.stanford.edu

Abstract

We survey two recent signature constructions that support signature aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single signature. This single signature (and all n original messages) will convince any verifier that the n users signed the n original messages (i.e., for $i = 1, \ldots, n$ user i signed message number i). We survey two constructions. The first is based on the short signature scheme of Boneh, Lynn, and Shacham and supports general aggregation. The second, based on a multisignature scheme of Micali, Ohta, and Reyzin, is built from any trapdoor permutation but only supports sequential aggregation. Aggregate signatures are useful for reducing the size of certificate chains (by aggregating all signatures in the chain) and for reducing message size in secure routing protocols such as SBGP.

1 Introduction

Security systems often manage signatures on many different messages generated by many different users. For example, in a Public Key Infrastructure (PKI) of depth n, user signatures are accompanied by a chain of n certificates. The chain contains n signatures by n Certificate Authorities (CAs) on n distinct certificates. Similarly, in the Secure BGP protocol (SBGP) [16] each router receives a list of n signatures attesting to a certain path of length n in the network. A router signs its own segment in the path and forwards the resulting list of $n+1$ signatures to the next router. As a result, the number of signatures in routing messages is linear in the length of the path. Both systems would benefit from a method for compressing the list of signatures on distinct messages issued by distinct parties. For example, certificate chains could be shortened by compressing the n signatures in the chain into a single signature. Note that one would still need to store the data in all certificates in the chain — only the signatures in the chain are compressed.

An aggregate signature scheme enables us to achieve precisely this type of compression. In this paper we survey two mechanisms for signature aggregation: general aggregation and sequential aggregation. We assume each of n users has a public-private key pair (PK_i, SK_i). User i wishes to sign message M_i.

General aggregate signatures. In a general signature aggregation scheme each user i signs her message M_i to obtain a signature σ_i. Then anyone can use a public aggregation algorithm to take all n signatures $\sigma_1, \ldots, \sigma_n$ and compress them into a single signature σ. Moreover, the aggregation can be performed incrementally — signatures σ_1, σ_2 can be aggregated

into σ_{12} which can then be further aggregated with σ_3 to obtain σ_{123}, and so on. There is also an aggregate verification algorithm that takes $PK_1, \ldots, PK_n$, $M_1, \ldots, M_n$, and σ and decides whether the aggregate signature is valid. Thus, an aggregate signature provides non-repudiation at once on many different messages by many users. We refer to this mechanism as *general* aggregation since aggregation can be done by anyone and without the cooperation of the signers. In the next section we describe a general aggregate signature scheme due to Boneh, Gentry, Lynn, and Shacham [5]. The scheme uses bilinear maps from algebraic geometry.

Sequential aggregate signatures. In a sequential aggregation scheme, signature aggregation can only be done during the signing process. Each signer in turn sequentially adds her signature to the current aggregate. Thus, there is an explicit order imposed on the aggregate signature and the signers must communicate with each other during the aggregation process. Operationally, sequential aggregation works as follows: User 1 signs M_1 to obtain σ_1; user 2 then combines σ_1 and M_2 to obtain σ_2; and so on. The final signature σ_n binds user i to M_i for all $i = 1, \ldots, n$. In Section 3 we describe a sequential aggregate signature scheme based on homomorphic trapdoor permutations such as RSA. The scheme is based on a multisignature scheme due to Micali, Ohta, and Reyzin [19] and analyzed in [27].

Although general aggregation is more powerful than sequential aggregation, the fact that sequential aggregation can be built from standard primitives such as RSA has its benefits. Interestingly, either mechanism can be used for compressing signatures in a certificate chain.

Aggregate signatures are related to multisignatures [24, 23, 20, 3]. In multisignatures, a set of users all sign the *same message* and the result is a single signature. Recently, Micali, Ohta, and Reyzin [20], presented a clear security model and new constructions for multisignatures. Another efficient construction was presented by Boldyreva [3]. Multisignatures are insufficient for the applications we have in mind, such as certificate chains and SBGP. For these applications we must be able to combine signatures on distinct messages into an aggregate.

The application of aggregate signatures to compressing certificate chains is related to an open problem posed by Micali and Rivest [21]: Given a certificate chain and some special additional signatures, can intermediate links in the chain be cut out? Aggregate signatures allow the compression of certificate chains without any additional signatures, but a verifier must still be aware of all intermediate links in the chain.

2 General Aggregate Signatures

In a general aggregate signature scheme, signatures are generated by individual users. They can then be combined into an aggregate signature by some aggregating party. The aggregating party need not be one of the users, and need not be trusted by them. Every aggregate signature scheme is a generalization of an ordinary signature scheme. An aggregate signature is the same length as an ordinary signature in the underlying scheme.

The aggregation algorithm takes as input signatures $\sigma_1, \ldots, \sigma_n$ on respective messages $M_1, \ldots, M_n$ under respective public keys $PK_1, \ldots, PK_n$. (The assignment of indices is arbitrary.) It outputs a single aggregate signature σ.

The aggregate verification algorithm, given an aggregate signature σ, messages $M_1, \ldots, M_n$, and public keys $PK_1, \ldots, PK_n$, verifies that σ is a valid aggregate signature on the given messages under the given keys.

2.1 Bilinear Maps

We start by reviewing the mathematical underpinnings of general aggregate signatures: Gap Diffie-Hellman groups and bilinear groups. Gap Diffie-Hellman groups arise from a separation between Computational and Decision Diffie-Hellman. Bilinear groups arise from the presence of a bilinear map, a function with certain properties.

Consider a multiplicative cyclic group G of prime order p, with generator g. On this group, the familiar Diffie-Hellman problems proceed as follows.

Computational Diffie-Hellman (CDH). Given $g, g^a, h \in G$, compute $h^a \in G$.

Decision Diffie-Hellman (DDH). Given $g, g^a, h, h^b \in G$, decide whether a equals b. Tuples of this form — (g, g^a, h, h^a) — are termed Diffie-Hellman tuples.

Loosely stated, the CDH assumption is that it is computationally infeasible to solve random instances of the CDH problem; the DDH assumption is similarly defined.

GDH Groups. For many choices of group G, such as subgroups of $\mathbb{Z}_q^*$, both the CDH and DDH assumptions are believed to hold. As we will see, however, on certain elliptic-curve groups, the DDH problem is easy to solve, whereas CDH is believed hard [6, 22]. We term groups that have this property Gap Diffie-Hellman (GDH) groups. GDH is an instance of a family of gap problems discussed by Okamoto and Pointcheval [25].

Bilinear groups. Currently, the only known examples of GDH groups have additional structure, namely, a bilinear map. A bilinear map is a map $e : G \times G \rightarrow G_T$ — where G_T is another multiplicative cyclic group of prime order p — with the following properties:

- Computable: there exists an efficiently-computable algorithm for computing $e(u,v)$, for all $u, v \in G$.
- Bilinear: for all $u, v \in G$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u,v)^{ab}$.
- Non-degenerate: $e(g,g) \neq 1$.

A bilinear group is any group that possesses such a map e, and on which CDH is hard.

Joux and Nguyen [15] noted that a bilinear map e provides an algorithm for solving DDH. For a tuple (g, g^a, h, h^b) we have

$$a = b \bmod p \quad \Longleftrightarrow \quad e(h, g^a) = e(h^b, g) \ .$$

Consequently, if a group G is a bilinear group then G is also a GDH group. (The converse is probably not true.)

We now describe the elliptic curve groups mentioned above. Let $E/\mathbb{F}_q$ be an elliptic curve, and let G be a subgroup (of prime order p) of the curve's group of points $E(\mathbb{F}_q)$. On certain curves, the Weil and modified Tate pairings [14, 12, 13] yield a bilinear map $e : G \times G \rightarrow G_T$. The target group G_T is a subgroup of $\mathbb{F}_{q^\alpha}$, where α is a security multiplier that depends on the curve and on the group G.

The multiplier α provides a tradeoff between efficiency and security. The smaller the value of α, the faster is the computation of the bilinear map; the larger the value of α, the more difficult is the CDH problem on G. Current CDH algorithms on G require solving the discrete logarithm problem either in the generic group G (of order p) or in the finite field $\mathbb{F}_{q^\alpha}$ [17, 18]. We note that members of the MNT family of curves [22] have large subgroups with security multiplier $\alpha = 6$, which is suitable for our needs.

2.2 The BLS Signature Scheme

We now describe the BLS short signature scheme. The scheme works in any Gap Diffie-Hellman group G. It requires, in addition, a hash function from the message space onto the group G. The scheme is related to the undeniable signature scheme of Chaum and Pedersen [7].

Specifically, let $G = \langle g \rangle$ be a GDH group of prime order p, with a hash function $H : \{0,1\}^* \to G$, viewed as a random oracle [2]. Any string can be signed; a signature is a single element of G. The scheme comprises the three algorithms below.

Key Generation. Pick random $x \xleftarrow{R} \mathbb{Z}_p$ and compute $v \leftarrow g^x$. The public key is $v \in G$. The private key is $x \in \mathbb{Z}_p$.

Signing. Given a private key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G$.

Verification. Given a public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$ and verify that (g, v, h, σ) is a valid Diffie-Hellman tuple.

The intuition is: On a correct signature, $v = g^x$, and $\sigma = h^x$, so (g, v, h, σ) is a Diffie-Hellman tuple. This establishes the validity of the scheme. Its security against existential forgery under a chosen message attack can be shown based on the CDH assumption in G [6].

Signature length. Points on an elliptic curve group $G < E(\mathbb{F}_q)$ are usually represented as a pair (x, y) of elements of $\mathbb{F}_q$, but BLS remains valid and secure even if only the x-coordinate of every signature point $\sigma \in G$ is transmitted. Thus, on an MNT curve (with $\alpha = 6$) over a 170-bit field, BLS signatures are 170 bits long, and provide security comparable to that of 1024-bit RSA [26, 4] or 320-bit DSA [11]. In other words, BLS signatures are half the size of DSA with comparable security.

Because of their simple mathematical structure, BLS signatures are amenable to a variety of extensions, including threshold signatures, multisignatures, and blind signatures [3].

2.3 Bilinear Aggregate Signatures

We now describe the bilinear aggregate signature scheme [5]. Unlike the BLS signature scheme on which it is based, the bilinear aggregate signature scheme requires the group G to be a bilinear group — a general Gap Diffie-Hellman group is insufficient. As in the BLS scheme, any string can be signed. The scheme employs a random oracle hash function, but one that takes both a string and an element of G as input: $H : G \times \{0,1\}^* \to G$.

The bilinear aggregate signature scheme enables general aggregation. An arbitrary aggregating party unrelated to, and untrusted by, the original signers can combine pre-existing signatures into an aggregate. The system does not impose an order on the aggregated elements. Note that, when needed, an order can be imposed by prepending index numbers to the messages being signed.

For notational convenience, we number the users whose signatures are aggregated $1, 2, \ldots, n$ in the description below. This numbering is arbitrary. The number of signatures n in an aggregate is effectively unbounded (viz., polynomial in the security parameter).

The scheme includes the three usual algorithms for generating and verifying individual signatures, as well as two additional algorithms that provide the aggregation capability.

Key Generation. For a particular user, pick random $x \xleftarrow{R} \mathbb{Z}_p$, and compute $v \leftarrow g^x$. The user's public key is $v \in G$. The user's private key is $x \in \mathbb{Z}_p$.

Signing. For a particular user, given the public key v, the private key x, and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(v,M)$, where $h \in G$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G$.

Verification. Given a user's public key v, a message M, and a signature σ, compute $h \leftarrow H(v,M)$; accept if $e(\sigma,g) = e(h,v)$ holds.

Aggregation. Arbitrarily assign to each user whose signature will be aggregated an index i, ranging from 1 to n. Each user i provides a signature $\sigma_i \in G$ on a message $M_i \in \{0,1\}^*$ of her choice. Compute $\sigma \leftarrow \prod_{i=1}^{n} \sigma_i$. The aggregate signature is $\sigma \in G$.

Aggregate Verification. We are given an aggregate signature $\sigma \in G$ for a set of users, indexed as before, and are given the original messages $M_i \in \{0,1\}^*$ and public keys $v_i \in G$. To verify the aggregate signature σ, compute $h_i \leftarrow H(v_i, M_i)$ for $1 \le i \le n$, and accept if $e(\sigma,g) = \prod_{i=1}^{n} e(h_i, v_i)$ holds.

The test employed in the verification of individual signatures is the same DDH test used in BLS verification, but rewritten in bilinear-map notation. Note that a bilinear aggregate signature, like a BLS signature, is a single element of G. Unlike in BLS, the signing process signs both the message and the user's public key.

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

$$\begin{aligned} e(\sigma,g) &= e\left(\prod_i h_i^{x_i}, g\right) \\ &= \prod_i e(h_i, g)^{x_i} \\ &= \prod_i e(h_i, g^{x_i}) \\ &= \prod_i e(h_i, v_i) \ , \end{aligned}$$

which is the right-hand side, as required. This establishes the validity of the scheme; its security against forgery can be demonstrated. Even when the would-be forger possesses all but one of the private keys, he cannot frame the remaining honest user. See [5] for the exact security model and proof of security based on CDH in G.

Incremental Aggregation. Consider an aggregate signature σ on messages $M_1, \ldots, M_n$ under public keys $v_1, \ldots, v_n$. An additional signature σ_{n+1} (on a message M_{n+1} under public key v_{n+1}) can be folded into the aggregate: $\sigma' \leftarrow \sigma \cdot \sigma_{n+1}$. If some signature σ_j included in σ is known, it can be removed from the aggregate: $\sigma' \leftarrow \sigma/\sigma_j$. If, however, only the messages, public keys, and the aggregate signature σ are known, recovering the individual signatures $\sigma_1, \ldots, \sigma_n$ from the aggregate is hard. This hardness assumption, the basis for other signature constructions [5], was shown by Coron and Naccache to be equivalent to Computational Diffie-Hellman [10].

3 Sequential Aggregate Signatures

Sequential aggregate signatures are a variant of aggregate signatures. In a sequential aggregate signature scheme, signatures are not individually generated and then combined into an aggregate. Rather, a would-be signer transforms a sequential aggregate into another that includes a signature on a message of his choice. Signing and aggregation are a single operation. Sequential aggregate signatures are built in layers, like an onion; the first signature in the aggregate is the inmost. As with general aggregate signatures, the resulting sequential aggregate is the same length as an

ordinary signature. This behavior closely mirrors the sequential nature of certificate chains in a PKI.

For sequential aggregate signatures, aggregation and signing are performed in a single combined operation. The operation takes as input a private key SK, a message M_i to sign, and a sequential aggregate signature σ' on messages $M_1,\ldots,M_{i-1}$ under respective public keys $PK_1,\ldots,PK_{i-1}$, where M_1 is the inmost message. It adds a signature on M_i under SK to the aggregate, outputting a sequential aggregate σ on all i messages $M_1,\ldots,M_i$.

The aggregate verification algorithm, given a sequential aggregate signature σ, messages $M_1,\ldots,M_i$, and public keys $PK_1,\ldots,PK_i$, verifies that σ is a valid sequential aggregate (with M_1 inmost) on the given messages under the given keys.

3.1 Trapdoor Homomorphic Permutations

Sequential aggregate signatures are built from trapdoor homomorphic permutations. We first review trapdoor permutations and then describe the sequential aggregate scheme to which they give rise.

A permutation family Π is a collection of permutations of some domain D. Each permutation in Π has a description $s \in S$. Anyone given a description s can evaluate the corresponding permutation.

Loosely speaking, a permutation family is one-way if, given a permutation description s, it is infeasible to invert the corresponding permutation. A permutation family is trapdoor if each description s has some corresponding trapdoor $t \in T$ such that it is easy to invert the permutation corresponding to s with t, but infeasible without t. A trapdoor permutation family is necessarily one-way. (Here S and T are arbitrary sets.)

More formally, a trapdoor permutation family Π comprises three algorithms: *Generate*, *Evaluate*, and *Invert*. The randomized generation algorithm *Generate* outputs the description $s \in S$ of a permutation along with the corresponding trapdoor $t \in T$. The evaluation algorithm *Evaluate*, given the permutation description s and a value $x \in D$, outputs $a \in D$, the image of x under the permutation. The inversion algorithm *Invert*, given the permutation description s, the trapdoor t, and a value $a \in D$, outputs the preimage of a under the permutation.

We require that $\mathit{Evaluate}(s,\cdot)$ be a permutation of D for all $(s,t) \xleftarrow{R} \mathit{Generate}$, and that $\mathit{Invert}(s,t,\mathit{Evaluate}(s,x)) = x$ hold for all $(s,t) \xleftarrow{R} \mathit{Generate}$ and for all $x \in D$.

A trapdoor permutation is homomorphic if D is a group with some operation $*$ and if, for all (s,t) generated by *Generate*, the permutation $\pi : D \to D$ induced by $\mathit{Evaluate}(s,\cdot)$ is an automorphism on D. That is, if $a = \pi(x)$ and $b = \pi(y)$, then $a * b = \pi(x * y)$.

When it engenders no ambiguity, we consider the output of the generation algorithm *Generate* as a probability distribution Π on permutations, and write $(\pi,\pi^{-1}) \xleftarrow{R} \Pi$; here π is the permutation $\mathit{Evaluate}(s,\cdot)$, and π^{-1} is the inverse permutation $\mathit{Invert}(s,t,\cdot)$.

It can happen that each permutation $\mathit{Evaluate}(s,\cdot)$ is over a different domain D_s. For example, the RSA permutation family gives permutations over domains $\mathbb{Z}_N^*$, where each user has a distinct modulus N. We consider this further in Section 3.4. For now we assume that all permutations in the family are over the same domain D.

3.2 Full-domain signatures

We review the full-domain hash signature scheme. The scheme, introduced by Bellare and Rogaway [1] and further analyzed by Coron [8], works in any trapdoor permutation family.

Like the others discussed above, the full-domain hash signature scheme employs a random-oracle hash function $H : \{0,1\}^* \to D$. The hash function maps bit strings into the entire domain D (rather than some subset of D), a fact which gives the scheme its name.

Key Generation. For a particular user, pick random $(s,t) \xleftarrow{R} \mathit{Generate}$. The user's public key PK is s. The user's private key SK is (s,t).

Signing. For a particular user, given the private key (s,t) and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in D$, and $\sigma \leftarrow \mathit{Invert}(s,t,h)$. The signature is $\sigma \in D$.

Verification. Given a user's public key s, a message M, and a signature σ, compute $h \leftarrow H(M)$; accept if $h = \mathit{Evaluate}(s,\sigma)$ holds.

These algorithms can also be described using the simplified notation given above. A user signs a message by publishing $\sigma = \pi^{-1}(H(M))$; the signature is valid if $\pi(\sigma) = H(M)$ holds.

The signature scheme is secure against existential forgery under a chosen message attack if Π is a trapdoor permutation family [1]. If Π is homomorphic as well, then the security reduction can be made more efficient [8].

3.3 Sequential Aggregate Signatures

We now describe the trapdoor sequential aggregate signature scheme. The scheme is related to the full-domain hash signature scheme, but must be instantiated on a *homomorphic* trapdoor permutation. The scheme is based on a multisignature scheme due to Micali, Ohta, and Reyzin [19].

To simplify the presentation of the scheme, we introduce some notation for vectors. We write a vector as $\mathbf{x}$, its length as $|\mathbf{x}|$, and its elements as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\mathbf{x}|}$. We denote vector concatenation as $\mathbf{x} \| \mathbf{y}$ and appending an element to a vector as $\mathbf{x} \| z$. For a vector $\mathbf{x}$, $\mathbf{x}|_a^b$ is the sub-vector containing elements $\mathbf{x}_a, \mathbf{x}_{a+1}, \ldots, \mathbf{x}_b$. It is necessarily the case that $1 \le a \le b \le |\mathbf{x}|$.

Like the others, this scheme employs a full-domain random-oracle hash function H mapping inputs into D. A signer provides to H every public key and every message in the aggregate signature she is creating. Thus H is of the form $H : \bigcup_{j=1}^{\infty} \left[(S)^j \times (\{0,1\}^*)^j \right] \to D$.

Key Generation. For a particular user, pick random $(s,t) \xleftarrow{R} \mathit{Generate}$. The user's public key PK is s. The user's private key SK is (s,t).

Aggregate Signing. The input is a private key (s,t), a message $M \in \{0,1\}^*$ to be signed, and a sequential aggregate signature σ' on a vector of messages $\mathbf{M}$ under a vector of public keys $\mathbf{s}$. No key may appear twice in $\mathbf{s}$. Furthermore, the vectors $\mathbf{M}$ and $\mathbf{s}$ must have the same length. Let i equal $|\mathbf{M}|$. If i is 0, σ' must equal 1, the unit of D.
Compute $h \leftarrow H(\mathbf{s} \| s, \mathbf{M} \| M)$, where $h \in D$, and $\sigma \leftarrow \mathit{Invert}(s,t,h * \sigma')$. The sequential aggregate signature is $\sigma \in D$.

Aggregate Verification. The input is a sequential aggregate signature σ on messages $\mathbf{M}$ under public keys $\mathbf{s}$, where $|\mathbf{M}| = |\mathbf{s}| = i$. To verify, set $\sigma_i \leftarrow \sigma$. Then, for $j = i, \ldots, 1$, set $\sigma_{j-1} \leftarrow \mathit{Evaluate}(\mathbf{s}_j, \sigma_j) * H(\mathbf{s}|_1^j, \mathbf{M}|_1^j)^{-1}$. Accept if σ_0 equals 1.

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

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

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

The trapdoor sequential aggregate signature scheme is secure against forgery, assuming Π is a homomorphic trapdoor permutation family. Even when the would-be forger possesses all but one of the private keys, he cannot frame the remaining honest user. For the precise security model and proof of security see [27].

3.4 Aggregating with RSA

We consider the details of instantiating the sequential aggregate signature scheme presented above using the RSA permutation family.

The RSA function was introduced by Rivest, Shamir, and Adleman [26]. If $N = pq$ is the product of two large primes and $ed = 1 \bmod \phi(N)$, then $\pi(x) = x^e \bmod N$ is a permutation on $\mathbb{Z}_N^*$, and $\pi^{-1}(x) = x^d \bmod N$ is its inverse. Setting $s = (N, e)$ and $t = (d)$ gives a trapdoor permutation that is multiplicatively homomorphic.

A difficulty arises since two users cannot share the same modulus N. Thus the domains of the one-way permutations belonging to the aggregating users differ, making it difficult to treat RSA as a family of trapdoor permutations. We give two approaches that allow us to create sequential aggregate signatures from RSA nonetheless. The first method imposes more restrictions on the choices of signing keys than the second. Aggregate signatures created by the second method grow by one bit per signature.

Suppose the n users have moduli $N_1, \ldots, N_n$, with N_1 inmost. We assume that the moduli are approximately the same size, i.e., that $\lfloor \log_2 N_1 \rfloor = \lfloor \log_2 N_2 \rfloor = \cdots = \lfloor \log_2 N_n \rfloor$. Let N be the minimum of $N_1, \ldots, N_n$. The hash function H maps into the set $\{1, \ldots, N-1\}$; hashes not in $\mathbb{Z}_{N_i}^*$ for some i can be dealt with by iterating the hash, using the method given by Bellare and Rogaway [1, Section 4].

In the first method, the moduli are constrained so that $N_1 < N_2 < \cdots < N_n$. A sequential aggregate signature σ_i under the keys with moduli $N_1, \ldots, N_i$ is such that $\sigma_i < N_i < N_{i+1}$. Thus (except with negligibly small probability) σ_i is in the domain of the permutation with modulus N_{i+1}. Letting $\pi_i(x) = x^{e_i} \bmod N_i$, we can apply the sequential aggregate signature scheme of Section 3.3 otherwise unchanged.

In the second method, the moduli are not ordered and increasing. It can then happen that σ_i is larger than N_{i+1}. We deal with this by truncating σ_i so that it fits. Let ℓ equal $\lfloor \log_2 N \rfloor$. Then $2^\ell < N_1, \ldots, N_n < 2^{\ell+1}$. Now, if some i-element sequential aggregate signature σ_i is such that $\sigma_i \geq 2^\ell$, we emit the bit $b_i \leftarrow 1$ and continue aggregation using $\sigma_i' \leftarrow \sigma_i - 2^\ell$; otherwise we emit the bit $b_i \leftarrow 0$ and continue aggregation using $\sigma_i' \leftarrow \sigma_i$. The n-bit vector $b_1, \ldots, b_n$ can be appended to the sequential aggregate signature, which then grows by a single bit per aggregating user, or it can be omitted and recovered by an exhaustive search of the 2^n possibilities.

These two schemes are no longer full-domain hash signature schemes, but, since the moduli are all approximately the same size, Coron's partial-domain hash analysis [9] applies to either.

4 Conclusions

We surveyed two techniques for signature aggregation. Both methods provide the ability to compress multiple signatures by distinct signers on distinct messages into a single signature. The first method, based on bilinear maps, provides general aggregation, where anyone can combine signatures into an aggregate at any time, without the cooperation of the signers. The

second method, based on homomorphic trapdoor permutations such as RSA, provides only sequential aggregation where aggregation must be done during the signing process. General aggregation is more a powerful mechanism than sequential aggregation. For example, sequential aggregation can be built from general aggregation. Also, general aggregation seems easier to use.

We discussed two applications for signature aggregation: compressing certificate chains in a PKI and compressing messages in secure routing protocols. Both aggregation techniques are adequate for these applications.

References

[1] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. Denning, R. Pyle, R. Ganesan, R. Sandhu, and V. Ashby, editors, *Proceedings of CCS 1993*, pages 62–73. ACM, 1993.

[2] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In U. Maurer, editor, *Proceedings of Eurocrypt 1996*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, 1996.

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

[4] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–13, 1999.

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

[6] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–32. Springer-Verlag, 2001. Full paper: `http://crypto.stanford.edu/~dabo/pubs.html`.

[7] D. Chaum and T. Pedersen. Wallet databases with observers. In E. Brickell, editor, *Proceedings of Crypto 1992*, volume 740 of *LNCS*, pages 89–105. Springer-Verlag, 1992.

[8] J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 229–35. Springer-Verlag, 2000.

[9] J.-S. Coron. Security proof for partial-domain hash signature schemes. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 613–26. Springer-Verlag, 2002.

[10] J.-S. Coron and D. Naccache. Boneh et al.'s k-element aggregate extraction assumption is equivalent to the Diffie-Hellman assumption. In C. S. Laih, editor, *Proceedings of Asiacrypt 2003*, LNCS. Springer-Verlag, 2003. To appear.

[11] FIPS 186-2. Digital signature standard, 2000.

[12] G. Frey, M. Muller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Trans. Info. Th.*, 45(5):1717–9, 1999.

[13] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D. Kohel, editors, *Proceedings of ANTS V*, volume 2369 of *LNCS*, pages 324–37. Springer-Verlag, 2002.

[14] P. Gaudry, F. Hess, and N. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *J. Cryptology*, 15(1):19–46, 2002.

[15] A. Joux and K. Nguyen. Separating decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/`.

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

[17] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In Y. Desmedt, editor, *Proceedings of Crypto 1994*, volume 839 of *LNCS*, pages 271–81. Springer-Verlag, 1994.

[18] A. Menezes, T. Okamoto, and P. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Info. Th.*, 39(5):1639–46, 1993.

[19] S. Micali, K. Ohta, and L. Reyzin. Provable-subgroup signatures. Unpublished manuscript, 1999.

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

[21] S. Micali and R. Rivest. Transitive signature schemes. In *Proceedings of RSA 2002*, volume 2271 of *LNCS*, pages 236–43. Springer-Verlag, 2002.

[22] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84-A(5):1234–43, May 2001.

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

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

[25] T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic primitives. In K. Kim, editor, *Proceedings of PKC 2001*, volume 1992 of *LNCS*, pages 104–18. Springer-Verlag, 2001.

[26] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Commun. ACM*, 21:120–126, 1978.

[27] H. Shacham. Sequential aggregate signatures from trapdoor homomorphic permutations. Cryptology ePrint Archive, Report 2003/091, 2003. `http://eprint.iacr.org/`.