Aggregate and Verifiably Encrypted Signatures from Bilinear Maps

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

An aggregate signature scheme is a digital signature that supports aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single short signature. This single signature (and the n original messages) will convince the verifier that the n users did indeed sign the n original messages (i.e., user i signed message M_i for $i = 1, \ldots, n$). In this paper we introduce the concept of an aggregate signature, present security models for such signatures, and give several applications for aggregate signatures. We construct an efficient aggregate signature from a recent short signature scheme based on bilinear maps due to Boneh, Lynn, and Shacham. 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. We also show that aggregate signatures give rise to verifiably encrypted signatures. Such signatures enable the verifier to test that a given ciphertext C is the encryption of a signature on a given message M. Verifiably encrypted signatures are used in contract-signing protocols. Finally, we show that similar ideas can be used to extend the short signature scheme to give simple ring signatures.

1 Introduction

Many real-world applications involve signatures on many different messages generated by many different users. For example, in a Public Key Infrastructure (PKI) of depth n, each user is given a chain of n certificates. The chain contains n signatures by n Certificate Authorities (CAs) on n distinct certificates. Similarly, in the Secure BGP protocol (SBGP) [17] 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 applications would benefit from a method for compressing the list of signatures on distinct messages issued by distinct parties. Specifically, X.509 certificate chains could be shortened by compressing the n signatures in the chain into a single signature.

An aggregate signature scheme enables us to achieve precisely this type of compression. Suppose each of n users has a public-private key pair (PK_i, SK_i). User u_i signs message M_i to obtain a signature σ_i. Then there is a public aggregation algorithm that takes as input all of $\sigma_1, \ldots, \sigma_n$ and outputs a short compressed signature σ. Anyone can aggregate the n signatures. Moreover, the aggregation can be performed incrementally. That is, signatures σ_1, σ_2 can be aggregated into σ_{12} which can then be further aggregated with σ_3 to obtain σ_{123}. When aggregating signatures in a certificate chain, each CA can incrementally aggregate its own signature into the chain. There is also an aggregate verification algorithm that takes $PK_1, \ldots, PK_n$, $M_1, \ldots, M_n$, and σ and decides whether the aggregate signature is valid. Intuitively, the security requirement is that the aggregate signature σ is declared valid only if the aggregator who created σ was given all of $\sigma_1, \ldots, \sigma_n$. Precise security definitions are given in Sect. 3.2. Thus, an aggregate signature provides non-repudiation at once on many different messages by many users.

We construct an aggregate signature scheme based on a recent short signature due to Boneh, Lynn, and Shacham (BLS) [6]. This signature scheme works in any group where the Decision Diffie-Hellman problem (DDH) is easy, but the Computational Diffie-Hellman problem (CDH) is hard. We refer to such groups as gap groups [6, 25]. Recently there have been a number of constructions using such gap groups [6, 18, 7, 4]. Surprisingly, general gap groups are insufficient for constructing efficient aggregate signatures. Instead, our construction uses a pair of groups G_1, G_T and a bilinear map $e : G_1 \times G_1 \to G_T$ where CDH is hard in G_1. Joux and Nguyen [16] showed that the map e can be used to solve DDH in G_1, and so G_1 is a gap group. It is the extra structure provided by the bilinear map that enables us to construct an efficient aggregate signature scheme. We do not know how to build efficient aggregate signatures from general gap groups. Thus, our construction is an example where the bilinear map provides extra functionality beyond a simple algorithm for solving DDH. Bilinear maps were previously used for three-way Diffie-Hellman [15], Identity-Based Encryption (IBE) [5], and Hierarchical IBE [14, 12].

Aggregate signatures are related to multisignatures [19, 24, 23, 4]. In multisignatures, a set of users all sign the *same message* and the result is a single signature. Recently, Micali et al. [19] defined a security model for multisignatures and gave some constructions and applications. Multisignatures are insufficient for the applications we have in mind, such as certificate chains and SBGP. For these applications we must be able to aggregate signatures on distinct messages. We note that recently Boldyreva [4] showed that general gap groups are sufficient for constructing multisignatures from BLS signatures. As noted above, to obtain aggregate signatures, one needs the extra structure provided by bilinear maps.

Our application of aggregate signatures to compressing certificate chains is related to an open problem posed by Micali and Rivest [20]: Given a certificate chain and some special additional signatures, can intermediate links in the chain be cut out? Aggregate signatures allow the compression of certificate chains without any additional signatures, but a verifier must still be aware of all intermediate links in the chain. We note that batch RSA [8] also provides some signature compression, but only for signatures produced by a single signer.

As a further application for aggregate signatures we show in Sect. 4 that certain aggregate signature schemes give rise to simple verifiably encrypted signatures. These signatures enable user Alice to give Bob a signature on a message M encrypted using a third party's public key and Bob to verify that the encrypted signature is valid. Verifiably encrypted signatures are used in optimistic contract signing protocols [1, 2] to enable fair exchange. Previous constructions [1, 26] require zero knowledge proofs to verify an encrypted signature. The verifiably encrypted signatures in Section 4 are short and can be validated efficiently. We note that the resulting contract signing protocol is not abuse-free in the sense of [9].

As a third application of these ideas we construct in Sect. 5 a simple ring signature [27] using bilinear maps. As above, the construction using a bilinear map is simpler and more efficient than constructions that only make use of gap groups.

2 Signature Schemes Based on Co-Gap Diffie-Hellman

We first review a few concepts related to bilinear maps and Gap Diffie-Hellman signatures [6]. Throughout the paper we use the following notation:

1. G_1 and G_2 are two (multiplicative) cyclic groups of prime order p;
2. g_1 is a generator of G_1 and g_2 is a generator of G_2;
3. ψ is a computable isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$; and
4. e is a computable bilinear map $e : G_1 \times G_2 \to G_T$ as described below.

The isomorphism ψ is mostly needed for the proofs of security. To keep the discussion general, we simply assume that ψ exists and is efficiently computable. When G_1, G_2 are subgroups of the group of points of an elliptic curve $E/\mathbb{F}_q$, the trace map on the curve can be used as this isomorphism (we assume $G_1 \subseteq E(\mathbb{F}_q)$ and $G_2 \subseteq E(\mathbb{F}_{q^r})$).

Throughout the paper, we consider bilinear maps $e : G_1 \times G_2 \to G_T$ where all groups G_1, G_2, G_T are multiplicative and of prime order p. One could set $G_1 = G_2$. However, we allow for the more general case where $G_1 \neq G_2$ so that our constructions can make use of certain families of non-supersingular elliptic curves defined by Miyaji et al. [21]. These curves give rise to very short signatures [6]. This will lead in turn to short aggregate signatures, ring signatures, etc. To handle the case $G_1 \neq G_2$ we define the co-CDH and co-DDH problems [6]. When $G_1 = G_2$, these problems reduce to the standard CDH and DDH problems. Hence, for the remainder of the paper, although we handle arbitrary G_1, G_2, for simplicity, the reader may assume $G_1 = G_2, g_1 = g_2$, and $\psi = I$, the identity map.
With this setup we obtain natural generalizations of the CDH and DDH problems:

Computational Co-Diffie-Hellman. Given $g_2, g_2^a \in G_2$ and $h \in G_1$ compute $h^a \in G_1$.

Decision Co-Diffie-Hellman. Given $g_2, g_2^a \in G_2$ and $h, h^b \in G_1$ output `yes` if $a = b$ and `no` otherwise. When the answer is `yes` we say that (g_2, g_2^a, h, h^a) is a co-Diffie-Hellman tuple.

When $G_1 = G_2$ and $g_1 = g_2$, these problems reduce to the standard CDH and DDH. Next we define co-GDH gap groups to be group pairs G_1 and G_2 on which co-DDH is easy but co-CDH is hard.

Definition 2.1. Two groups (G_1, G_2) are a decision group pair for co-Diffie-Hellman if the group action on G_1, the group action on G_2, and the map ψ from G_2 to G_1 can be computed in one time unit, and Decision co-Diffie-Hellman on (G_1, G_2) can be solved in one time unit.

Definition 2.2. The advantage of an algorithm $\mathcal{A}$ in solving the Computational co-Diffie-Hellman problem in groups G_1 and G_2 is

$$\mathsf{Adv\ co\text{-}CDH}_{\mathcal{A}} \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}(g_2, g_2^a, h) = h^a : a \stackrel{R}{\leftarrow} \mathbb{Z}_p, h \stackrel{R}{\leftarrow} G_1\right] .$$

The probability is taken over the choice of a, h, and $\mathcal{A}$'s coin tosses. An algorithm $\mathcal{A}$ (t, ϵ)-breaks Computational co-Diffie-Hellman on G_1 and G_2 if $\mathcal{A}$ runs in time at most t, and $\mathsf{Adv\ co\text{-}CDH}_{\mathcal{A}}$ is at least ϵ. Two Groups (G_1, G_2) are a (t, ϵ)-co-GDH group pair if they are a decision group pair for co-Diffie-Hellman and no algorithm (t, ϵ)-breaks Computational co-Diffie-Hellman on them.

2.1 Bilinear Maps

Let G_1 and G_2 be two groups as above, with an additional group G_T such that $|G_1| = |G_2| = |G_T|$. A bilinear map is a map $e : G_1 \times G_2 \to G_T$ with the following properties:

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

2. Non-degenerate: $e(g_1, g_2) \neq 1$.

These properties imply two more: for any $u_1, u_2 \in G_1, v \in G_2$, $e(u_1 u_2, v) = e(u_1, v) \cdot e(u_2, v)$; and for any $u, v \in G_2$, $e(\psi(u), v) = e(\psi(v), u)$.

Definition 2.3. Two groups (G_1, G_2) are a bilinear group pair if the group action on either can be computed in one time unit, the map ψ from G_2 to G_1 can be computed in one time unit, a bilinear map $e : G_1 \times G_2 \to G_T$ exists, and e is computable in one time unit.

Definition 2.4. Two groups (G_1, G_2) are a (t, ϵ)-bilinear group pair for co-Diffie-Hellman if they are a bilinear group pair and no algorithm (t, ϵ)-breaks Computational co-Diffie-Hellman on them.

Joux and Nguyen [16] showed that an efficiently-computable bilinear map e provides an algorithm for solving the decision co-Diffie-Hellman problem. For a tuple (g_2, g_2^a, h, h^b) we have

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

Consequently, if two groups (G_1, G_2) are a (t, ϵ)-bilinear group pair for co-Diffie-Hellman, then they are also a $(t/2, \epsilon)$-co-GDH group pair. The converse is probably not true.

2.2 The Co-GDH Signature Scheme

We review the signature scheme of [6], which can be based on any gap group. It comprises three algorithms, *KeyGen*, *Sign*, and *Verify*, and uses a full-domain hash function $H : \{0,1\}^* \to G_1$, viewed as a random oracle [3].

Key Generation. Pick random $x \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The public key is $v \in G_2$. The secret key is $x \in \mathbb{Z}_p$.

Signing. Given a secret key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

Verification. Given a public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$ and verify that (g_2, v, h, σ) is a valid co-Diffie-Hellman tuple.

A co-GDH signature is a single element of G_1. On certain elliptic curves these signatures are very short: they are half the size of DSA signatures with similar security. Theorem 1 of [6] proves the existential unforgeability of the scheme under a chosen message attack [13] in the random oracle model assuming (G_1, G_2) is a co-gap group pair for Diffie-Hellman.

3 Aggregate Signatures

We define aggregate signatures and describe an aggregate signature scheme based on co-GDH signatures. Unlike the co-GDH scheme, aggregate signatures require the existence of a bilinear map. We define security models and provide proofs of security for aggregate signatures.

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

3.1 Bilinear Aggregate Signatures

We describe a bilinear aggregate signature scheme based on the co-GDH scheme presented above. Individual signatures in the aggregate signature scheme are created and verified precisely as are signatures in the co-GDH scheme (Sect. 2.2). Aggregate verification makes use of a bilinear map on G_1 and G_2.

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

The scheme comprises five algorithms: *KeyGen*, *Sign*, *Verify*, *Aggregate*, and *AggregateVerify*. The first three are as in ordinary signature schemes; the last two provide the aggregation capability. The scheme employs a full-domain hash function $H : \{0,1\}^* \to G_1$, viewed as a random oracle.

Key Generation. For a particular user, pick random $x \xleftarrow{R} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key is $v \in G_2$. The user's secret key is $x \in \mathbb{Z}_p$.

Signing. For a particular user, given the secret key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

Verification. Given user's public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$; accept if $e(\sigma, g_2) = e(h, v)$ holds.

Aggregation. For the aggregating subset of users $U \subseteq \mathbb{U}$, assign to each user an index i, ranging from 1 to $k = |U|$. Each user $u_i \in U$ provides a signature $\sigma_i \in G_1$ on a message $M_i \in \{0,1\}^*$ of his choice. The messages M_i must all be distinct. Compute $\sigma \leftarrow \prod_{i=1}^k \sigma_i$. The aggregate signature is $\sigma \in G_1$.

Aggregate Verification. We are given an aggregate signature $\sigma \in G_1$ for an aggregating subset of users U, indexed as before, and are given the original messages $M_i \in \{0,1\}^*$ and public keys $v_i \in G_2$ for all users $u_i \in U$. To verify the aggregate signature σ,

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

A bilinear aggregate signature, like a co-GDH signature, is a single element of G_1. Note that aggregation can be done incrementally.

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

$$e(\sigma, g_2) = e(\prod_i h_i^{x_i}, g_2) = \prod_i e(h_i, g_2)^{x_i} = \prod_i e(h_i, g_2^{x_i}) = \prod_i e(h_i, v_i) \ ,$$

which is the right-hand side, as required. It remains to prove the security of the scheme.

3.2 Aggregate Signature Security

Informally, the security of aggregate signature schemes is equivalent to the nonexistence of an adversary capable, within the confines of a certain game, of existentially forging an aggregate signature. Existential forgery here means that the adversary attempts to forge an aggregate signature, on messages of his choice, by some set of users.

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

Setup. The aggregate forger $\mathcal{A}$ is provided with a public key PK_1, generated at random.

Queries. Proceeding adaptively, $\mathcal{A}$ requests signatures with PK_1 on messages of his choice.

Response. Finally, $\mathcal{A}$ outputs $k-1$ additional public keys $PK_2, \ldots, PK_k$. Here k is at most N, a game parameter. These keys, along with the initial key PK_1, will be included in $\mathcal{A}$'s forged aggregate. $\mathcal{A}$ also outputs messages $M_1, \ldots, M_k$; and, finally, an aggregate signature σ by the k users, each on his corresponding message.

The forger wins if the aggregate signature σ is a valid aggregate on messages $M_1, \ldots, M_k$ under keys $PK_1, \ldots, PK_k$, and σ is nontrivial, i.e., $\mathcal{A}$ did not request a signature on M_1 under PK_1. The probability is over the coin tosses of the key-generation algorithm and of $\mathcal{A}$.

Definition 3.1. An aggregate forger $\mathcal{A}$ $(t, q_H, q_S, N, \epsilon)$-breaks an N-user aggregate signature scheme in the aggregate chosen-key model if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the hash function and at most q_S queries to the signing oracle; $\mathsf{Adv\,AggSig}_{\mathcal{A}}$ is at least ϵ; and the forged aggregate signature is by at most N users. An aggregate signature scheme is $(t, q_H, q_S, N, \epsilon)$-secure against existential forgery in the aggregate chosen-key model if no forger $(t, q_H, q_S, N, \epsilon)$-breaks it.

A potential attack on aggregate signatures. The adversary's ability in the chosen-key model to generate keys suggests the following attack, previously considered in the context of multisignatures [19, 4]. Alice publishes her public key v_A. Bob generates a private key x'_B and a public key $v'_B = g_2^{x'_B}$, but publishes as his public key $v_B = v'_B / v_A$, a value whose discrete log he does not know. Then $H(M)^{x'_B}$ verifies as an aggregate signature on M by both Alice and Bob. Note that in this forgery Alice and Bob both sign the same message M.

One countermeasure is to require the adversary to prove knowledge of the discrete logarithms (to base g_2) of his published public keys. For example, Boldyreva, in her multisignature scheme [4], requires, in effect, that the adversary disclose the corresponding private keys $x_2, \ldots, x_k$. Micali et al. [19] discuss a series of more sophisticated approaches based on zero-knowledge proofs, again with the effect that the adversary is constrained in his key selection. These defenses apply equally well to our aggregate signature scheme. For aggregate signatures, though, there is a simpler defense.

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

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

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

Theorem 3.2. *Let (G_1, G_2) be a (t', ϵ')-bilinear group pair for co-Diffie-Hellman, with each group of order p, with respective generators g_1 and g_2, with an isomorphism ψ computable from G_2 to G_1, and with a bilinear map $e : G_1 \times G_2 \to G_T$. Then the bilinear aggregate signature scheme on (G_1, G_2) is $(t, q_H, q_S, N, \epsilon)$-secure against existential forgery in the aggregate chosen-key model for all t and ϵ satisfying*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2]. \tag{1}$$

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

Claim 3.3. *The probability that algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}$'s aggregate signature queries is at least* $(1 - 1/(q_S + N))^{q_S}$. *Hence,* $\Pr[\mathcal{E}_1] \ge (1 - 1/(q_S + N))^{q_S}$.

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

Claim 3.4. *If algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}$'s queries then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Hence,* $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.

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

Claim 3.5. *The probability that algorithm $\mathcal{C}$ does not abort after $\mathcal{A}$ outputs a valid and nontrivial forgery is at least* $(1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$.
Hence, $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq (1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$.

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

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

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

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

To complete the proof of Theorem 3.2, we use the bounds from the claims above in equation (1). Algorithm $\mathcal{C}$ produces the correct answer with probability at least

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

as required.

Algorithm $\mathcal{C}$'s running time is the same as $\mathcal{A}$'s running time plus the time is takes to respond to $(q_H + q_S)$ hash queries and q_S signature queries, and the time to transform $\mathcal{A}$'s final forgery into the co-CDH solution. Each query requires an exponentiation in G_1. The output phase requires at most N additional hash computations, two inversions, two exponentiations, and $N + 1$ multiplications. We assume that exponentiation and inversion in G_1 take time c_{G_1}. Hence, the total running time is at most $t + c_{G_1}(q_H + 2q_S + N + 4) + N + 1 \leq t'$ as required. This completes the proof of Theorem 3.2. $\square$

Aggregate verification time. Let σ be an aggregate of the n signatures $\sigma_1, \ldots, \sigma_n$. The time to verify the aggregate signature σ is linear in n. In the special case when all n signatures are issued by the same public key v, aggregate verification is faster. One need only verify that $e(\sigma, g_2) = e(v, \prod_{i=1}^{n} H(M_i))$ holds, where $M_1, \ldots, M_n$ are the signed messages.

4 Verifiably Encrypted Signatures

Next, we show an application of aggregate signatures to verifiably encrypted signatures. Verifiably encrypted signatures are used in applications such as online contract signing [1, 2]. Suppose Alice wants to show Bob that she has signed a message, but does not want Bob to possess her signature of that message. (Alice will give her signature to Bob only when a certain event has occurred, e.g., Bob has given Alice his signature on the same message.) Alice can achieve this by encrypting her signature using the public key of a trusted third party, and sending this to Bob along with a proof that she has given him a valid encryption of her signature. Bob can verify that Alice has signed the message, but cannot deduce any information about her signature. Later in the protocol, if Alice is unable or unwilling to reveal her signature, Bob can ask the third party to reveal Alice's signature. We note that the resulting contract signing protocol is not abuse-free in the sense of [9].

We show that a variant of the bilinear aggregate signature scheme allows the creation of very efficient verifiably encrypted signatures.

4.1 Verifiably Encrypted Signature Security

A verifiably encrypted signature scheme comprises seven algorithms. Three, *KeyGen*, *Sign*, and *Verify*, are analogous to those in ordinary signature schemes. The others, *AdjKeyGen*, *VESigCreate*, *VESigVerify*, and *Adjudicate*, provide the verifiably encrypted signature capability. The algorithms are described below. We refer to the trusted third party as the adjudicator.

Key Generation, Signing, Verification. As in standard signature schemes.

Adjudicator Key. Generate a public-private key pair (APK, ASK) for the adjudicator.

VESig Creation. Given a secret key SK, a message M, and an adjudicator's public key APK, compute (probabilistically) a verifiably encrypted signature ω on M.

VESig Verification. Given a public key PK, a message M, an adjudicator's public key APK, and a verifiably encrypted signature ω, verify that ω is a valid verifiably encrypted signature on M under key PK.

Adjudication. Given an adjudicator's keypair (APK, ASK), a certified public key PK, and a verifiably encrypted signature ω on some message M, extract and output σ, an ordinary signature on M under PK.

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

Validity requires that verifiably encrypted signatures verify, and that adjudicated verifiably encrypted signatures verify as ordinary signatures, i.e., that $\mathit{VESigVerify}(M, \mathit{VESigCreate}(M))$ and $\mathit{Verify}(M, \mathit{Adjudicate}(\mathit{VESigCreate}(M))$ hold for all M and for all properly-generated keypairs and adjudicator keypairs. (The keys provided to the algorithms are here omitted for brevity.)

Unforgeability requires that it be difficult to forge a valid verifiably encrypted signature. The advantage in existentially forging a verifiably encrypted signature of an algorithm $\mathcal{F}$, given access to a verifiably-encrypted-signature creation oracle S and an adjudication oracle A, along with a hash oracle, is

$$\mathsf{Adv\,VSigF}_{\mathcal{F}} \stackrel{\mathrm{def}}{=} \Pr\left[\begin{array}{l} \mathit{VESigVerify}(PK, APK, M, \omega) = \texttt{valid} : \\ \qquad (PK, SK) \stackrel{\mathrm{R}}{\leftarrow} \mathit{KeyGen}, \\ \qquad (APK, ASK) \stackrel{\mathrm{R}}{\leftarrow} \mathit{AdjKeyGen}, \\ \qquad (M, \omega) \stackrel{\mathrm{R}}{\leftarrow} \mathcal{F}^{S,A}(PK, APK) \end{array}\right].$$

The probability is taken over the coin tosses of the key-generation algorithms, of the oracles, and of the forger. The forger is additionally constrained in that its forgery on M must be nontrivial: It must not previously have queried either oracle at M. Note that an ordinary signing oracle is not provided; it can be simulated by a call to S followed by a call to A.

Definition 4.1. A verifiably encrypted signature forger $\mathcal{F}$ $(t, q_H, q_S, q_A, \epsilon)$-forges a verifiably encrypted signature if: Algorithm $\mathcal{F}$ runs in time at most t; $\mathcal{F}$ makes at most q_H queries to the hash function, at most q_S queries to the verifiably-encrypted-signature creation oracle S, at most q_A queries to the adjudication oracle A; and $\mathsf{Adv\,VSigF}_{\mathcal{F}}$ is at least ϵ. A verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against existential forgery if no forger $(t, q_H, q_S, q_A, \epsilon)$-breaks it.

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

$$\mathsf{Adv\,VSigE}_{\mathcal{E}} \stackrel{\mathrm{def}}{=} \Pr\left[\begin{array}{l} \mathit{Verify}(PK, M, \sigma) = \texttt{valid} : \\ \qquad (PK, SK) \stackrel{\mathrm{R}}{\leftarrow} \mathit{KeyGen}, \\ \qquad (APK, ASK) \stackrel{\mathrm{R}}{\leftarrow} \mathit{AdjKeyGen}, \\ \qquad (M, \sigma) \stackrel{\mathrm{R}}{\leftarrow} \mathcal{E}^{S,A}(PK, APK) \end{array}\right].$$

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

Definition 4.2. An algorithm $\mathcal{E}$ $(t, q_H, q_S, q_A, \epsilon)$-extracts a verifiably encrypted signature if $\mathcal{E}$ runs in time at most t, makes at most q_H queries to the hash function, at most q_S queries to the verifiably-encrypted-signature creation oracle S, at most q_A queries to the adjudication oracle, and $\mathsf{Adv\,VSigE}_{\mathcal{E}}$ is at least ϵ. A verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against extraction if no algorithm $(t, q_H, q_S, q_A, \epsilon)$-extracts it.

4.2 Aggregate Extraction

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

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

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

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

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

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

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

Definition 4.3. An algorithm $\mathcal{E}$ (t, k, ϵ)-extracts a subaggregate from an k-element bilinear aggregate signature if $\mathcal{E}$ runs in time at most t and $\mathsf{Adv}\, k\text{-}\mathsf{Extr}_{\mathcal{E}}$ is at least ϵ. An instantiation of the bilinear aggregate signature scheme is (t, k, ϵ)-secure against aggregate extraction if no algorithm (t, k, ϵ)-extracts it.

We will be particularly concerned with the case $k = 2$. In this case, the aggregate extraction problem reduces to this one: given g_2^a, g_2^b, g_1^u, g_1^v, and g_1^{au+bv}, calculate g_1^{au}. (If the extractor outputs g_1^{bv} instead, we may recover g_1^{au} as g_1^{au+bv}/g_1^{bv}.)

4.3 Verifiably Encrypted Signatures via Aggregation

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

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

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

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

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

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

4.4 The Bilinear Verifiably-Encrypted Signature Scheme

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

Key Generation. *KeyGen* and *AdjKeyGen* are the same as *KeyGen* in the co-GDH signature scheme.

Signing, Verification. *Sign* and *Verify* are the same as in the co-GDH signature scheme.

VESig Creation. Given a secret key $x \in \mathbb{Z}_p$ a message $M \in \{0,1\}^*$, and an adjudicator's public key $v' \in G_2$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. Select r at random from $\mathbb{Z}_p$ and set $\mu \leftarrow \psi(g_2)^r$ and $\sigma' \leftarrow \psi(v')^r$. Aggregate σ and σ' as $\omega \leftarrow \sigma\sigma' \in G_1$. The verifiably encrypted signature is the pair (ω, μ). (This can also be viewed as ElGamal encryption of σ under the adjudicator's key.)

VESig Verification. Given a public key v, a message M, an adjudicator's public key v', and a verifiably encrypted signature (ω, μ), set $h \leftarrow H(M)$; accept if $e(\omega, g_2) = e(h, v) \cdot e(\mu, v')$ holds.

Adjudication. Given an adjudicator's public key v' and corresponding private key $x' \in \mathbb{Z}_p$, a certified public key v, and a verifiably encrypted signature (ω, μ) on some message M, ensure that the verifiably encrypted signature is valid; then output $\sigma = \omega/\mu^{x'}$.

If the adjudicator does not first validate a purported verifiably encrypted signature, a malicious user can trick him into signing arbitrary messages under his adjudication key. Similarly, the adjudicator should only adjudicate for certified public keys v; we assume that the CA, in issuing a certificate on v, verifies that the user knows the private key for v.

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

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

Theorem 4.4. *Let G_1 and G_2 be cyclic groups of prime order p, with respective generators g_1 and g_2, with a computable bilinear map $e : G_1 \times G_2 \to G_T$. Suppose that the co-GDH signature scheme is $(t', q'_H, q'_S, \epsilon')$-secure against existential forgery on (G_1, G_2). Then the bilinear verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against existential forgery on (G_1, G_2), for all $q_H \le q'_H$, $q_S \le q'_S$, $\epsilon \ge \epsilon'$, and all t satisfying $t \le t' - 2c_{G_1}(q_S + q_A + 1)$, where exponentiation and inversion on G_1 take time c_{G_1}.*

Proof. Given a verifiably-encrypted-signature forger algorithm $\mathcal{V}$, we construct a forger algorithm $\mathcal{F}$ for the underlying co-GDH signature scheme.

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

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

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

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

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

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

Output. Finally, $\mathcal{V}$ halts, either declaring failure, in which case $\mathcal{F}$, too, declares failure and halts, or providing a valid and nontrivial verifiably encrypted signature (ω^*, μ^*) on a message M^*. $\mathcal{F}$ sets $\sigma^* \leftarrow \omega^*/(\mu^*)^{x'}$ which, by the validity property, is a valid co-GDH signature on M^* under key v.

That the forgery is nontrivial means that $\mathcal{V}$ did not query the verifiably encrypted signature oracle at M^*, from which it follows that $\mathcal{F}$ did not query its signing oracle at M^*. Thus (M^*, σ^*) is a nontrivial co-GDH forgery; algorithm $\mathcal{F}$ outputs it and halts.

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

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

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

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

Theorem 4.5. *Let G_1 and G_2 be cyclic groups of prime order p, with respective generators g_1 and g_2, with a computable isomorphism $\psi : G_2 \to G_1$ such that $\psi(g_2) = g_1$ and a computable bilinear map $e : G_1 \times G_2 \to G_T$. Suppose that the bilinear aggregate signature scheme on (G_1, G_2) is $(t', 2, \epsilon')$-secure against aggregate extraction. Then the bilinear verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against extraction on (G_1, G_2) for all t and ϵ satisfying*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2]. \tag{2}$$

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

Claim 4.6. *The probability that algorithm $\mathcal{A}$ does not abort as a result of $\mathcal{V}$'s adjudication queries is at least $1/e$. Hence, $\Pr[\mathcal{E}_1] \geq 1/e$.*

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

Claim 4.7. *If algorithm $\mathcal{A}$ does not abort as a result of $\mathcal{V}$'s adjudication queries then $\mathcal{V}$'s view is identical to its view in the real attack. Hence, $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.*

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

Claim 4.8. *The probability that algorithm $\mathcal{A}$ does not abort after $\mathcal{V}$ outputs a valid and nontrivial verifiably-encrypted signature extraction is at least $1/(q_A + 1)$ Hence,* $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(q_A + 1)$.

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

Using the bounds from the claims above in equation (2) shows that $\mathcal{A}$ produces the correct answer with probability at least $\epsilon/e(q_A + 1) \geq \epsilon'$ as required.

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

4.5 Observations on Verifiably Encrypted Signatures

We note some extensions of the verifiably encrypted signature scheme discussed above. Some of these rely for security on the k-element aggregate extraction assumption with $k > 2$.

- Anyone can convert an ordinary unencrypted signature to a verifiably encrypted signature. The same applies to unencrypted aggregate signatures.
- An adjudicator's private key can be shared amongst n parties using k-of-n threshold cryptography [11, 10], so that k parties are needed to adjudicate a verifiably encrypted signature.
- A message-signature pair in the co-GDH signature scheme is of the same form as an identity–private-key pair in the Boneh-Franklin Identity-Based Encryption Scheme [5]. Thus the verifiably encrypted signature scheme can potentially be modified to yield a verifiably encrypted encryption scheme for IBE private keys. Verifiably encrypted private keys have many applications [26].

5 Ring Signatures

Rivest, Shamir and Tauman define ring signature schemes and construct some using RSA and Rabin cryptosystems [27]. Naor defines the closely-related notion of deniable ring authentication and proposes such a scheme that relies only on the existence of a strong encryption function [22]. We shall see that co-GDH signatures give rise to natural ring signatures.

5.1 Ring Signatures

Consider a set U of users. Each user $u \in U$ has a signing keypair (PK_u, SK_u). A ring signature on U is a signature that is constructed using all the public keys of the users in U, and a single private key of any user in U. A ring signature has the property that a verifier is convinced that the signature was produced using one of the private keys of U, but is not able to determine which one. This property is called *signer-ambiguity* [27]. Applications for ring signatures include authenticated (yet repudiable) communication and leaking secrets [27].

Zhang and Kim [28] devised a bilinear ring signature in an identity-based setting. Our scheme differs from theirs, as our goal is to extend co-GDH signatures to obtain efficient ring signatures; the system parameters and key generation algorithm in our system are identical to those of the co-GDH scheme.

5.2 Bilinear Ring Signatures

The ring signature scheme comprises three algorithms: *KeyGen*, *RingSign*, and *RingVerify*. Recall g_1, g_2 are generators of groups G_1, G_2 respectively, and $e : G_1 \times G_2 \to G_T$ is a bilinear map, and a computable isomorphism $\psi : G_2 \to G_1$ exists, with $\psi(g_2) = g_1$. Again we use a full-domain hash function $H : \{0,1\}^* \to G_1$. The security analysis views H as a random oracle.

Key Generation. For a particular user, pick random $x \xleftarrow{R} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key is $v \in G_2$. The user's secret key is $x \in \mathbb{Z}_p$.

Ring Signing. Given public keys $v_1, \ldots, v_n \in G_2$, a message $M \in \{0,1\}^*$, and a private key x corresponding to one of the public keys v_s for some s, choose random $a_i \xleftarrow{R} \mathbb{Z}_p$ for all $i \neq s$. Compute $h \leftarrow H(M) \in G_1$ and set

$$\sigma_s \leftarrow \left(h / \psi\Big(\prod_{i \neq s} v_i^{a_i} \Big) \right)^{1/x}.$$

For all $i \neq s$ let $\sigma_i \leftarrow g_1^{a_i}$. Output the ring signature $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in G_1^n$.

Ring Verification. Given public keys $v_1, \ldots, v_n \in G_2$, a message $M \in \{0,1\}^*$, and a ring signature σ, compute $h \leftarrow H(M)$ and verify that $e(h, g_2) = \prod_{i=1}^n e(\sigma_i, v_i)$.

Using the bilinearity and nondegeneracy of the pairing e, it is easy to show that a signature produced by the *RingSign* algorithm will verify under the *RingVerify* algorithm.

5.3 Security

There are two aspects a security analysis for ring signatures we must consider. Firstly, signer ambiguity must be ensured. We show that the identity of the signer is unconditionally protected.

Theorem 5.1. *For any algorithm $\mathcal{A}$, any set of users U, and a random $u \in U$, the probability $\Pr[\mathcal{A}(\sigma) = u]$ is at most $1/|U|$, where σ is any ring signature on U generated with private key SK_u.*

Proof. The theorem follows from a simple probability argument: for any $h \in G_1$, and any s, $1 \le s \le n$, the distribution $\{g_1^{a_1}, \ldots, g_1^{a_n} : a_i \xleftarrow{R} \mathbb{Z}_p$ for $i \neq s, a_s$ chosen such that $\prod_{i=1}^n g_1^{a_i} = h\}$ is identical to the distribution $\{g_1^{a_1}, \ldots, g_1^{a_n} : \prod_{i=1}^n g_1^{a_i} = h\}$, since the value of any one of the a_i's is uniquely determined by the values of the other a_i's. □

Secondly, we need to examine the scheme's resistance to forgery. We adopt the security model of Rivest, Shamir and Tauman [27]. Consider the following game played between an adversary and a challenger. The adversary is given the public keys $v_1, \ldots, v_n$ of a set of users U, and is given oracle access to h and a ring-signing oracle. The adversary may work adaptively. The goal of the adversary is to output a valid ring signature on U of a message M subject to the condition that M has never been presented to the ring-signing oracle. An adversary A's advantage $\mathsf{Adv\,RingSig}_{\mathcal{A}}$ in existentially forging a bilinear ring signature is the probability, taken over the coin tosses of the key-generation algorithm and of the forger, that A succeeds in creating a valid ring signature in the above game.

Theorem 5.2. *Suppose $\mathcal{F}$ is a (t', ϵ')-algorithm that can produce a forgery of a ring signature on a set of users of size n. Then there exists an (t, ϵ)-algorithm that can solve the co-CDH problem where $t \le 2t' + 2c_{G_2}(2n + q_H + nq_S)$ and $\epsilon \ge ((\epsilon'/e)(1 + q_S))^2$, where $\mathcal{F}$ issues at most q_S ring-signature queries and at most q_H hash queries, and exponentiation and inversion on G_2 take time c_{G_2}.*

Proof. The co-CDH problem can be solved by first solving two random instances of the following problem: Given g_1^{ab}, g_2^a (and g_1, g_2), compute g_1^b. We shall construct an algorithm $\mathcal{A}$ that solves this problem. This is easy if $a = 0$. In what follows, we assume $a \neq 0$.

Initially $\mathcal{A}$ picks $x_2, \ldots, x_n$ at random from $\mathbb{Z}_p$ and sets $x_1 = 1$. It sets $v_i = (g_2^a)^{x_i}$. Algorithm $\mathcal{F}$ is given the public keys $v_1, \ldots, v_n$. Without loss of generality we may assume $\mathcal{F}$ submits distinct queries (as previous replies can be cached); that for every ring-signing query on a message M, $\mathcal{F}$ has previously issued a hash query for M; and that $\mathcal{F}$ issues a hash query on the message on which it attempts to forge a signature some time before giving its final output.

On a hash query, $\mathcal{A}$ flips a coin that shows 0 with probability p and 1 otherwise (p shall be determined later). Then $\mathcal{A}$ picks a random $r \xleftarrow{R} \mathbb{Z}_p$, and if the coins shows 0, $\mathcal{A}$ returns $(g_1^{ab})^r$, otherwise it returns $\psi(g_2^a)^r$.

Suppose $\mathcal{F}$ issues a ring sign query for a message M. By assumption, $\mathcal{A}$ has previously issued a hash query for M. If the coin $\mathcal{A}$ flipped for this h-query showed 0, then $\mathcal{A}$ fails and exits. Otherwise $\mathcal{A}$ had returned $H(M) = \psi(g_2^a)^r$ for some r. In this case $\mathcal{A}$ chooses random $a_2, \ldots, a_n \xleftarrow{R} \mathbb{Z}_p$, computes $a_1 = r - (a_2 x_2 + \ldots + a_n x_n)$, and returns the signature $\sigma = \langle g_1^{a_1}, \ldots, g_1^{a_n} \rangle$.

Eventually $\mathcal{F}$ outputs a forgery $\langle \sigma_1, \ldots, \sigma_n \rangle$ for a message M. Again by assumption, $\mathcal{F}$ has previously issued a h-query for M. If the coin flipped by $\mathcal{A}$ for this query did not show 0 then $\mathcal{A}$ fails. Otherwise $H(M) = g_1^{abr}$ for some r chosen by $\mathcal{A}$, and $\mathcal{A}$ outputs the rth root of $\sigma_1 {\sigma_2}^{x_2} \ldots {\sigma_n}^{x_n}$.

Algorithm $\mathcal{F}$ cannot distinguish between $\mathcal{A}$'s simulation and real life. Also, $\mathcal{A}$ will not fail with probability $p^{q_S}(1 - p)$ which is maximized when $p = q_S/(q_S + 1)$, giving a bound of $(1/e)(1 + q_S)$. If it does not fail and $\mathcal{F}$ successfully forges a ring signature then $\mathcal{A}$ is successful and outputs g_1^b. Algorithm $\mathcal{A}$ requires n exponentiations on G_2 in setup, one exponentiation for each of $\mathcal{F}$'s hash queries, n exponentiations for each of $\mathcal{F}$'s signature queries, and n exponentiations in the output phase, so its running time is $\mathcal{F}$'s running time plus $c_{G_2}(2n + q_H + nq_S)$. □

5.4 Observations on Ring Signatures

Any ring signature scheme restricts to an ordinary signature scheme when $n = 1$. Our scheme restricts to a short signature scheme similar to the co-GDH scheme [6]. In this modified co-GDH scheme, σ equals $h^{1/x}$ rather than h^x, and one verifies that $e(h, g_2) = e(\sigma, v)$ rather than that $e(\sigma, g_2) = e(h, v)$.

However, bilinear ring signatures have interesting properties that do not appear to be shared by ring signatures in general. For any set of users U with $u \in U$, *anyone* can convert a modified co-GDH signature by u into a ring signature by U. Specifically, to convert a modified co-GDH signature σ_1 on M for public key v_1 into a ring signature $\sigma = \langle \sigma'_1, \ldots, \sigma'_n \rangle$ on M for public keys $v_1, \ldots, v_n$, we choose $r_i \xleftarrow{R} \mathbb{Z}_p$ for $2 \le i \le n$, and set $\sigma'_1 \leftarrow \sigma_1 \prod_{i=2}^{n} \psi(v_i^{r_i})$ and $\sigma'_i \leftarrow \psi(v_1^{-r_i})$ for $2 \le i \le n$. More generally, anyone can further anonymize a ring signature by adding users to U.

6 Conclusions

We introduced the concept of aggregate signatures and constructed an efficient aggregate signature scheme based on bilinear maps. Key generation, aggregation, and verification require no interaction. We proved security of the system in a model that gives the adversary his choice of public keys and messages to forge. For security, we introduced the additional constraint that an aggregate signature is valid only if it is an aggregation of signatures on *distinct* messages. This constraint is satisfied naturally for the applications we have in mind. More generally, the constraint can be satisfied by prepending the public key to the message prior to signing.

We gave several applications for aggregate signatures. For example, they can be used to reduce the size of certificate chains and reduce communication bandwidth in protocols such as SBGP. We also showed that our specific aggregate signature scheme gives verifiably encrypted signatures.

Previous signature constructions using bilinear maps [6, 18, 7, 4] only required a gap Diffie-Hellman group (i.e., DDH easy, but CDH hard). The signature constructions in this paper require the extra structure provided by the bilinear map. These constructions are an example where a bilinear map provides more power than a generic gap Diffie-Hellman group.

Acknowledgments

The authors thank Leonid Reyzin, Liqun Chen, and Cynthia Dwork for helpful discussions about this work. The first author is supported by DARPA, the Packard foundation, and an NSF CAREER award. The third and fourth authors are supported by DARPA and NSF.

References

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

[2] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with offline TTP. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 77–85, 1998.

[3] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Proceedings of Eurocrypt '96*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, 1996.

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

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

[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] Y. Dodis. Efficient construction of (distributed) verifiable random functions. In *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 1–17. Springer-Verlag, 2003.

[8] A. Fiat. Batch RSA. In *Proceedings of Crypto '89*, pages 175–185, 1989.

[9] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Proceedings of Crypto '99*, volume 1666 of *LNCS*, pages 449–466. Springer-Verlag, 1999.

[10] P. Gemmel. An introduction to threshold cryptography. *RSA CryptoBytes*, 2(3):7–12, 1997.

[11] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. Robust and efficient sharing of RSA functions. *J. Cryptology*, 13(2):273–300, 2000.

[12] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Proceedings of Asiacrypt 2002*, volume 2501 of *LNCS*, pages 548–66. Springer-Verlag, 2002.

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

[14] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 466–81. Springer-Verlag, 2002.

[15] A. Joux. A one round protocol for tripartite Diffie-Hellman. In *Proceedings of ANTS IV*, volume 1838 of *LNCS*, pages 385–94. Springer-Verlag, 2000.

[16] 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/`.

[17] 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.

[18] A. Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 597–612. Springer-Verlag, 2002.

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

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

[21] 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.

[22] M. Naor. Deniable ring authentication. In *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 481–98. Springer-Verlag, 2002.

[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):432–441, 1998.

[25] T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic primitives. In *Proceedings of PKC 2001*, volume 1992 of *LNCS*, pages 104–118. Springer-Verlag, 2001.

[26] G. Poupard and J. Stern. Fair encryption of RSA keys. In *Proceedings of Eurocrypt 2000*, volume 1807 of *LNCS*, pages 172–89. Springer-Verlag, 2000.

[27] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 552–65. Springer-Verlag, 2001.

[28] F. Zhang and K. Kim. ID-based blind signature and ring signature from pairings. In *Proceedings of Asiacrypt 2002*, volume 2501 of *LNCS*, pages 533–47. Springer-Verlag, 2002.