Aggregate Signatures using Bilinear Maps

Frank Wang
CS 259C

1 Introduction

Bilinear maps have been used in many revolutionary cryptographic schemes in the past decade. In fact, ever since the emergence of Identity Based Encryption by Boneh and Franklin [1], bilinear maps have been consistently used to construct shorter signatures [3], more powerful encryption schemes, such as Attribute Based Encryption [6], and three-way Diffie-Hellman key exchange [5]. In this paper, we describe how bilinear maps are used to construct aggregate signatures. Aggregate signatures are of particular interest because many times signatures on different messages are generated by different users. For example, as stated by Boneh et al. [2] in a Public Key Infrastructure, a user is given a chain of n certificates, which are signed by n Certificate Authorities. Having so many signatures might be slow and expensive to transfer over a network. As a result, efficiency would be improved if we could compress all these signatures into one single signature. In the paper, we describe an aggregate signature scheme due to Boneh et al. [2], which is based on a short signature scheme by Boneh, Lynn, and Shacham (BLS) [3]. For clarity, aggregate signatures are different from multisignatures where many users sign the same message and those signatures are compressed into a single signature. Similarly, batch RSA [4] can provide signature compression but only if the signatures originate from the same person.

In the sections below, we will provide some definitions and terms used throughout the paper, then describe the relevant BLS signature scheme, present the aggregate signature scheme, and finally prove security.

2 Definitions

BeforeweI continue to the construction and the security of aggregate signatures. Let me present some notations and definitions that we plan to use, which are also used in the original paper of Boneh et al. [2].

Here is some notation that we will use:

1. G_1 and G_2 are two multiplicative cyclic groups of prime order p;

2. g_1 generates G_1 and g_2 generates G_2;
3. ψ is a computable isomorphism from G_2 to G_1, with $\psi(g_2)$=g_1;
4. e is a computable bilinear map e: $G_1 \times G_2 \rightarrow G_T$

For simplicity, assume the isomorphism ψ exists and is easily computable. If G_1, G_2 are subgroups of the group of the points on the elliptic curve $E/\mathbb{F}_q$, the isomorphism can be the trace map on the curve.

Now, let me define the general Computational Diffie-Hellman and Decisional Diffie-Hellman 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. If the answer is yes, (g_2, g_2^a, h, h^a) is a co-Diffie-Hellman tuple.

For completeness, it is important to note that if $G_1 = G_2$ and $g_1 = g_2$, the co-CDH and co-DDH problems reduce down to the standard CDH and DDH problems. We now define co-Gap Diffie-Hellman gap groups to be group pairs G_1 and G_2 on which co-DDH is easy and co-CDH is hard.

Definition Two groups are (G_1, G_2) are a decision group pair for co-Diffie-Hellman if the group 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 The advantage of an algorithm A in solving the Computational co-Diffie-Hellman problem in groups G_1 and G_2 is

$$\text{Adv co-CDH}_A = \Pr[\text{A}(g_2, g_2^a, h) = h^a : a \overset{R}{\leftarrow} \mathbb{Z}_p,\, h \overset{R}{\leftarrow} G_1].$$

The probability is taken over the choice of a, h, and A's coin tosses. An algorithm A (t, ϵ)-breaks Computational co-Diffie-Hellman on G_1 and G_2 if A runs in time at most t, and Adv co-CDH$_A$ 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.

Next, we will define the properties of bilinear maps.

2.1 Bilinear Maps

Let G_1, G_2 be the groups described above and G_T be an additional group such that the size of all the groups are the same. 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: $e(g_1, g_2) \neq 1$.

The following properties follow from those above: 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 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 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.

In the next section, we will present the construction of the short BLS signatures from bilinear maps.

3 Short Signatures from Bilinear Maps

We will present the general co-GDH signature scheme and then apply the scheme to co-GDH signatures from elliptic curves. Both schemes are from the work of Boneh, Lynn, and Shacham [3]. We will not provide proofs of security, but they can be found in the original paper.

3.1 Co-GDH Signature Scheme

A signature scheme involves three algorithms: Key Generation, Signing, and Verification. Assume a full-domain hash function H:$\{0, 1\}^* \to G_1$, which we will treat as a random .

Key Generation. Pick random $x \xleftarrow{R} \mathbb{Z}_p$, 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.

This signature only requires a single element in G_1, so under certain groups, specifically on certain elliptic curves, these signatures are short. In the next section, we will show the elliptic curve version of this signature scheme.

3.2 Co-GDH Signature Scheme on Elliptic Curves

We will adapt the scheme above to use points on an elliptic curve. G_1, G_2 are subgroups of the elliptic curve $E/\mathbb{F}_q$, and assume there is a mapping that can place the message M as a point on the elliptic curve. In other words, just assume that M was mapped to a point on the elliptic curve in G_1. More details regarding how to translate M to a point on the elliptic curve can be found in the original paper [3].

Key generation. Pick random $x \overset{R}{\leftarrow} \mathbb{Z}_p$ and compute $V \leftarrow xQ$ where Q generates G_2. The public key is $V \in G_2$. The secret key is $x \in \mathbb{Z}_p$.

Signing. Given a secret key $x \in \mathbb{Z}_p$, and a message $M \in G_1$, compute $\sigma \leftarrow xM \in E(\mathbb{Z}_q)$. Output the x-coordinate of σ as the signature s on M, so $s \in \mathbb{F}_q$.

Verification Given a public key $V \in G_2$, a message $M \in G_1$, and a signature $s \in \mathbb{F}_q$, find a $y \in \mathbb{F}_q$ such that $\sigma = (s, y)$ is a point of order p in $E(\mathbb{F}_q)$. If no such y exists, output no. Otherwise, test if $e(\sigma, Q) = e(M, V)$ or $e(\sigma, Q)^{-1} = e(M, V)$. If so, output yes. Otherwise, output no.

This signature requires only one point on the elliptic curve, so it is about half the size of DSA signatures. In fact, the signature length is about $\log_2 q$.

In the next section, we use similar ideas to construct aggregate signatures.

4 Aggregate Signatures

In this section, we will define aggregate signatures and then present an aggregate signature scheme based on co-GDH signatures described above. The discussion will follow closely with that of Boneh et al. [2].

4.1 Definition

Consider a set of users U with each user u having a signing public key-private key pair (PK_u, SK_u). To aggregate signatures on a subset of users in U, each user in that subset produces a signature σ_u on any message M_u. These signatures are aggregated by an aggregating party into a single signature σ, which is the same length as a single signature σ_u. The aggregating party has access to all the public keys, the messages, and signatures on those message, but it does not have access to any private keys. For the verifier, given a signature σ and the identities of the users who had signatures in the message, the verifier can be convinced that those users signed the message.

4.2 Bilinear Aggregate Signatures

Since the final aggregate signature is the same as the length of a single signature, we will present an aggregate signature scheme based on BLS signatures [3] described in Section 3.1. It is important to note that this scheme can produce short signatures if specific elliptic curves are used, and a summary of this adaptation on elliptic curves to produce short signatures is also presented above.

The aggregation scheme has five algorithms: Key Generation, Signing, Verification, Aggregation, and Aggregate verification. All the parameters are the same as that described in the co-GDH signature scheme above. In fact, the key generation, signing, and verification are exactly the same as the scheme above. We will state them again below for completeness, and then we will provide two additional algorithms that allow us to aggregate signatures and verify these aggregate signatures.

Key Generation. Pick random $x \overset{R}{\leftarrow} \mathbb{Z}_p$, 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 $e(\sigma, g_2) = e(h, v)$ holds.

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

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

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

Like the co-GDH signature, the bilinear aggregate signature requires only a single element of G_1 and has the same length as any individual signature. Therefore, if we use BLS signatures for the individual signatures, we can get a short aggregate signature.

We will briefly show correctness for the aggregate signature scheme. Given, that $\sigma = \prod_i \sigma_i = \prod_i h_i^{x_i}$ where h_i is the hashed message M_i for user i and the public key for each user i is $v_i = g_2^{x_i}$. Using the bilinear properties, the left-side of the aggregation verification becomes:

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

which is equal to the right hand side. In the next section, we will prove security of this scheme.

4.3 Security Proof

We present the aggregate chosen-key security model from Boneh et al [2]. The adversary A is given a single public key to create an existential forgery of an aggregate signature. The adversary can choose all public keys except the challenge public key and has access to a signing oracle on the challenge key. More formally, we define the adversary's advantage Adv AggSig$_A$ as the probability of success for the following game.

Setup. The aggregate forger A is provided with a public key PK_1, generated at random.

Queries. Proceeding adaptively, A requests signatures with PK_1 on messages of his choice.

Response. Finally, A outputs k - 1 additional public keys $PK_2, ..., PK_k$. Here k is at most N, a game parameter. These keys, along with initial key PK_1, will be included in A's forged aggregate. A also outputs messages $M_1, ..., M_k$; 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 message $M_1, ..., M_k$ under keys $PK_1, ..., PK_k$, and σ is nontrivial, meaning 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 A.

We will now define what it means for an aggregate forger to break an aggregate signature scheme in the security model described above.

Definition An aggregate forger $A(t, q_H, q_S, N, \epsilon)$-breaks an N-user aggregate signature scheme in the aggregate chose-key model if: A runs in time at most t; A makes at most q_H queries to the hash function and at most q_S queries to the signing oracle; AdvAggSig$_A$ 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.

Now, we will provide a security proof for the aggregate signature scheme under the aggregate chosen-key security model that follows closely from the original paper. More precisely, consider the following theorem:

Theorem 4.1 *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*

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

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

Proof We will prove the contrapositive. Suppose A is a forger algorithm that $(t, q_S, q_H, N, \epsilon)$-breaks the signature scheme. We will construct a t'-time algorithm C that breaks co-CDH in (G_1, G_2) with probability at least ϵ'.

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

Setup. Algorithm C starts by giving 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. C maintains a list of tuples $(M^{(i)}, w^{(i)}, b^{(i)}, c^{(i)})$. We call this list the H-list. The list is initially empty. When A queries H, which we model as a random oracle, at a point $M \in \{0, 1\}^*$, algorithm C responds as follows:

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

It is important to note that w is uniform in G_1 and independent of A's current view.

Signature Queries. Algorithm A requests a signature on some message M under the challenge key v_1. Algorithm C responds:

1. Algorithm C runs the above algorithm for responding to H-queries on M, obtaining the corresponding tuple (M, w, b, c) on the H-list. If $c = 0$ holds then, C stops and outputs no.

2. We know that $c = 1$ holds and hence $w = \psi(g_2)^b \in G_1$. Let $\sigma = \psi(u)^b \cdot \psi(g_2)^{rb} \in G_1$. Therefore, $\sigma = w^{a+r}$ and a valid signature on M under the public key $v_1 = u \cdot g_2^r = g_2^{a+r}$. Algorithm C gives σ to A.

Output. Finally, A halts. It either fails or outputs a value k, $k-1$ public keys $v_2, ..., v_k \in G_2$, k messages $M_1, ..., M_k$, and a forged aggregate signature $\sigma \in G_1$. All messages M_i have to be distinct, and A must not have requested a signature on M_1 as described earlier. Algorithm C runs its hash algorithm at each M_i, $1 \leq i \leq k$, obtaining the k corresponding tuples (M_i, w_i, b_i, c_i) on the H-list.

Algorithm C now proceeds only if $c_1 = 0$ and, for $2 \leq i \leq k$, $c_i = 1$; otherwise C outputs no. Since $c_1 = 0$, $w_1 = h \cdot \psi(g_2)^{b_1}$. For $i > 1$, $w_i = \psi(g_2)^{b_i}$ since $c_i = 1$. Now, we need to form an aggregate signature that satisfies $e(\sigma, g_2) = \prod_{i=1}^k e(w_i, v_i)$. For each $i > 1$, C sets $\sigma_i \leftarrow \psi(v_i)^{b_i}$. Then, for $i > 1$, using bilinear properties,

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

For a given message M_i, w_i would be the hash and σ_i would be a valid signature for a public v_i. Since we could not query a signature for M_1, C needs to construct a value for σ_1. Take σ from above and calculate $\sigma_1 \leftarrow \sigma \cdot (\prod_{i=2}^k \sigma_i)^{-1}$. Then

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

C calculates and outputs $h^a \leftarrow \sigma_1 \cdot (\psi(u)^{b_1} \cdot h^r \cdot \psi(g_2)^{rb_1})^{-1}$ as the co-CDH answer.

Now, we need to show that this algorithm can solve the co-CDH problem in (G_1, G_2) with probability at least ϵ'.

In order for the C described above these three events need to happen: E_1: C does not terminate because of A's queries. E_2: A generates a valid and nontrivial aggregate signature forgery. E_3: E_2 occurs and in addition, $c_1 = 0$, and for $2 \leq i \leq k$, $c_i = 1$, where c_i is from the H-list tuple.

Since E_3 encompasses E_2, we want $\Pr[E_1 \wedge E_3]$, which decomposes to $\Pr[E_1] \cdot \Pr[E_2 | E_1] \cdot \Pr[E_3 | E_1 \wedge E_2]$.

For these three terms, we will provide lower bounds below, but we will not prove the claims. The proof for the claims can be found in the original paper.

Claim 4.2 *The probability that algorithm C does not abort as a result of A's aggregate signature queries is at least $(1 - 1/(q_S + N))^{q_S}$. Hence, $Pr[E_1] \geq (1 - 1/(q_S + N))^{q_S}$.*

Claim 4.3 *If algorithm C does not abort as a result of A's queries, then algorithm A's view is identical to its view in the real attack. Hence, $Pr[E_2 | E_1] \geq \epsilon$.*

Claim 4.4 *The probability that algorithm C does not abort after A outputs a valid and nontrivial forgery is at least* $(1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$. *Hence* $Pr[E_3|E_1 \wedge E_2] \geq (1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$.

We combine all the claims and we get the Algorithm C will produce the right answer with probability at least

$$(1\text{-}\ 1/(q_S + N))^{q_S+N-1} \cdot\ 1/(q_S + N)\ \cdot\epsilon \geq \epsilon/(e(q_S + N)) \geq \epsilon'$$

For completeness, the running time of C is dependent on responding to $(q_H + q_S)$ hash queries, q_S signature queries, and translating the final output into a co-CDH solution. Each query takes one exponentiation in G_1, and the output takes N hash computations, two inversions, two exponentiations, and $N + 1$ multiplications, so the running time is at most $t + c_{G_1}(q_H + 2q_S + N + 4) + N + 1 \leq t'$. QED.

5 Conclusion

In this paper, we discuss the importance of bilinear maps in cryptography. They have had important applications for key exchange, encryption, and signatures. We discuss the use of bilinear maps to construct short signatures on elliptic curves, which is used as a foundation for the construction of compressed and aggregated signatures. From aggregation, we can also construct verifiably encrypted signatures as well as ring signatures. Bilinear maps are a very powerful tool, which might have many more undiscovered uses in cryptography.

References

[1] D. Boneh and M. Franklin. "Identity based encryption from the Weil pairing." *SIAM J. of Computing*, Vol. 32, No.3, 2003.

[2] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps." In proceedings of *Eurocrypt 2003*, LNCS 2656, pp. 416-432, 2003.

[3] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from Weil Pairing." *J. of Cryptology*, Vol. 17, No. 4, 2004.

[4] A. Fiat. "Batch RSA." In Proceedings of Crypto '89, pages 175-185, 1989.

[5] A. Joux. "A one round protocol for tripartite Diffie-Hellman." In Proceedings of *ANTSIV*, Vol. 1838 of LNCS, pages 385-94, 2000.

[6] J. Bethencourt, A. Sahai, and B. Waters. "Ciphertext-Policy Attribute-Based Encryption." In Proceedings of *28th IEEE Symposium on Security and Privacy*, 2007.