BLS Multi-Signatures With Public-Key Aggregation

Dan Boneh, Manu Drijvers, Gregory Neven

March 24, 2018

Abstract. This short note describes a simple approach for aggregating many BLS signatures on a common message, so that verifying the short multi-signature is fast. Moreover, the system supports public key aggregation, where the verification algorithm only uses a short aggregated public key. The original public keys are not needed for verifying the multi-signature. An important property of the construction is that the scheme is secure against a rogue public-key attack *without* requiring users to prove knowledge of their secret keys (this is sometimes called the plain public-key model). The construction builds upon the work of Bellare and Neven, and the recent work of Maxwell, Poelstra, Seurin, and Wuille.

Note: The full version of this work titled **Compact Multi-Signatures for Smaller Blockchains** is available here.

1. BLS signature aggregation

The BLS signature scheme [BLS01] operates in a prime order group and supports simple threshold signature generation, threshold key generation, and signature aggregation [BGLS03]. To review, the scheme uses the following ingredients:

- a **bilinear pairing** $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$. The pairing is efficiently computable, non-degenerate, and all three groups have prime order q. We let g_0 and g_1 be generators of $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively.
- a **hash function** $H_0 : \mathcal{M} \to \mathbb{G}_0$. The hash function will be treated as a random oracle in the security analysis.

Now the BLS signature scheme is defined as follows:

- $\mathbf{KeyGen}()$: choose a random $\alpha \xleftarrow{R} \mathbb{Z}_q$ and set $h \leftarrow g_1^\alpha \in \mathbb{G}_1$. output $pk := (h)$ and $sk := (\alpha)$.
- $\mathbf{Sign}(sk, m)$: output $\sigma \leftarrow H_0(m)^\alpha \in \mathbb{G}_0$. The signature is a *single* group element.
- $\mathbf{Verify}(pk, m, \sigma)$: if $e(g_1, \sigma) = e\big(pk,\ H_0(m)\big)$ output "accept", otherwise output "reject".

Signature aggregation. Given triples $(pk_i,\ m_i,\ \sigma_i)$ for $i = 1, \ldots, n$, anyone can aggregate the signatures $\sigma_1, \ldots, \sigma_n \in \mathbb{G}_0$ into a short convincing aggregate signature σ by computing

$$\sigma \leftarrow \sigma_1 \cdots \sigma_n \in \mathbb{G}_0. \tag{1}$$

Verifying an aggregate signature $\sigma \in \mathbb{G}_0$ is done by checking that

$$e(g_1, \sigma) = e\big(pk_1,\ H_0(m_1)\big) \cdots e\big(pk_n,\ H_0(m_n)\big). \tag{2}$$

When all the messages are the same ($m_1 = \ldots = m_n$) the verification relation (2) reduces to a simpler test that requires only two pairings:

$$e(g_1, \sigma) = e\Big(pk_1 \cdots pk_n,\ H_0(m_1)\Big). \tag{3}$$

The rogue public-key attack. This signature aggregation method (1) is insecure by itself due to a rogue public-key attack, where an attacker registers the public key $pk_2 := g_1^\beta \cdot (pk_1)^{-1} \in \mathbb{G}_1$, where $pk_1 \in \mathbb{G}_1$ is a public key of some unsuspecting user Bob, and $\beta \xleftarrow{R} \mathbb{Z}_q$ is chosen by the attacker. The attacker can then claim that both it and Bob signed some message $m \in \mathcal{M}$ by presenting the aggregate signature $\sigma := H_0(m)^\beta$. This signature verifies as an aggregate of two signatures, one from pk_1 and one from pk_2, because

$$e(g_1, \sigma) = e\big(g_1,\ H_0(m)^\beta\big) = e\big(g_1^\beta,\ H_0(m)\big) = e\big(pk_1 \cdot pk_2,\ H_0(m)\big).$$

Hence, this σ satisfies (3). In effect, the attacker committed Bob to the message m, without Bob ever signing m.

Defenses. There are two standard defenses against the rogue public-key attack:

- **Prove knowledge of the secret key (KOSK)** [Bol03, LOS06, RY07]:
  Require every user that registers a public key to prove knowledge of the corresponding secret key. However, this is difficult to enforce in practice, and does not fit well with applications to crypto currencies [MPSW18].
- **Distinct messages** [BGLS03, BNN07]:
  Alternatively, require that all the messages being aggregated are distinct. This can be easily enforced by always prepending the public key to every message prior to signing. However, because now all messages are distinct, we cannot take advantage of the efficiency improvement in (3) that apply when aggregating signatures on a common message m.

This work. In this short note we propose a third defense against the rogue public-key attack that retains the benefits of both defenses above, without the drawbacks. The scheme supports fast verification as in (3) and does not require users to prove knowledge of their secret key. Our construction is based on the approach developed in [BN06] and [MPSW18] for securing Schnorr multi-signatures.

We first describe the scheme and then describe its applications and security proof.

2. The modified BLS multi-signature construction

As in BLS, the scheme needs a bilinear pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ and a hash function $H_0 : \mathcal{M} \to \mathbb{G}_0$. We will also need a second hash function H_1:

$$H_1 : \mathbb{G}_1^n \to R^n \qquad \text{where} \qquad R := \{1, 2, \ldots, 2^{128}\}$$

and where $1 \le n \le \tilde{N}$. The security analysis will treat H_0 and H_1 as random oracles. With these ingredients, the modified BLS multi-signature scheme works as follows:

- $\mathbf{KeyGen}()$: as in BLS, choose a random $\alpha \xleftarrow{R} \mathbb{Z}_q$ and set $h \leftarrow g_1^\alpha$. output $pk := (h)$ and $sk := (\alpha)$.
- $\mathbf{Sign}(sk, m)$: as in BLS, output $\sigma \leftarrow H_0(m)^\alpha \in \mathbb{G}_0$.
- $\mathbf{Aggregate}\Big((pk_1, \sigma_1), \ldots, (pk_n, \sigma_n)\Big)$:
  1. compute $(t_1, \ldots, t_n) \leftarrow H_1(pk_1, \ldots, pk_n) \in R^n$.
  2. output the multi-signature $\sigma \leftarrow \sigma_1^{t_1} \cdots \sigma_n^{t_n} \in \mathbb{G}_0$.
- $\mathbf{Verify}\big(pk_1, \ldots, pk_n,\ m,\ \sigma\big)$: to verify a multi-signature σ on m do
  1. compute $(t_1, \ldots, t_n) \leftarrow H_1(pk_1, \ldots, pk_n) \in R^n$.
  2. compute the aggregate public key $apk \leftarrow pk_1^{t_1} \cdots pk_n^{t_n} \in \mathbb{G}_1$.
  3. if $e(g_1, \sigma) = e(apk, H_0(m))$ output "accept", otherwise output "reject".

  Verification always requires two pairings, independent of n.

Notice that the aggregate public key apk can be computed in step (2) of Verify before the message m is known. In particular, verifying the multi-signature σ is the same as verifying that σ is a standard BLS signature on m with respect to the aggregated public key apk. Once apk is computed, there is no need to provide the verifier with the underlying public keys $pk_1, \ldots, pk_n$. This mechanism is called *public key aggregation*. We prove security of this scheme in Section 3.

Batch verification. A set of b multi-signatures can be verified as a batch faster than verifying them one by one. To see how, suppose we are given triples (m_i, σ_i, apk_i) for $i = 1, \ldots, b$, where apk_i is the aggregated public-key used to verify the multi-signature σ_i on m_i. If all the messages $m_1, \ldots, m_b$ are distinct then we can use signature aggregation as in (1) to verify all these triples as a batch:

1. Compute an aggregate signature $\tilde{\sigma} = \sigma_1 \cdots \sigma_b \in \mathbb{G}_1$,
2. Accept all b multi-signature tuples as valid iff $\quad e(g_1, \tilde{\sigma}) = e\big(apk_1, H_0(m_1)\big) \cdots e\big(apk_b, H_0(m_b)\big)$.

This way, verifying the b multi-signatures requires only $b + 1$ pairings instead of $2b$ pairings to verify them one by one. We stress that this simple batching procedure can only be used when all the messages $m_1, \ldots, m_b$ are distinct. If some messages are repeated then batch verification can be done by first choosing random $\rho_1, \ldots, \rho_b \xleftarrow{R} \{1, \ldots, 2^{64}\}$, computing $\tilde{\sigma} = \sigma_1^{\rho_1} \cdots \sigma_b^{\rho_b} \in \mathbb{G}_1$, and checking that

$$e(g_1, \tilde{\sigma}) = e\big(apk_1^{\rho_1}, H_0(m_1)\big) \cdots e\big(apk_b^{\rho_b}, H_0(m_b)\big).$$

Of course the pairings on the right hand side can be coalesced for repeated messages.

Comparison to Schnorr multi-signatures. Schnorr signatures support multi-signatures [IN83, MOR01, BN06, MPSW18], however, aggregation

can only take place at the time of signing and requires a multi-round protocol between the signers. In BLS, aggregation can take place publicly by a simple multiplication, even long after all the signatures have been generated and the signers are no longer available.

2.1 Application to crypto currencies such as Bitcoin

Maxwell et al. [MPSW18] describe a number of important applications for multi-signatures to crypto currencies such as Bitcoin.

- **Multisig addresses.** In current Bitcoin, to spend from a t-of-n multisig address one must list all n public keys and t signatures in the spending transaction. Usually all t signatures are computed over a common message, namely the spending transaction data. When using BLS multi-signatures, anyone can aggregate all t signatures into a single aggregate signature and shrink the spending transaction size. If a user does not aggregate the signatures, the miner mining the transaction can do it on their behalf.

  Moreover, in the case of n-of-n multisig, one can further aggregate all n public keys into a single aggregate public key that is used to derive the multisig address. When spending from the address, one need only specify the aggregate public key, which further shrinks the transaction size.

  For t-of-n multisig, where $\binom{n}{t}$ is of moderate size, one can generate a Merkle tree from all $\binom{n}{t}$ aggregate public keys, and derive the multisig address from the short Merkle root. When spending from the multisig address, one provides a single aggregate signature, along with the single aggregate public key apk of the t signers, and the Merkle proof for this apk. In the full paper (Section 4.2) we present a construction that can handle t-of-n multisig even when $\binom{n}{t}$ is large.

- **Multi-input transactions.** For transactions that have multiple inputs, one currently includes all the signatures in the transaction. One can set things up so that all signatures are computed over the same message, namely the transaction data (see [MPSW18] for the details). When using BLS multi-signatures, anyone can aggregate all these signatures into a single multi-signature. If the user does not aggregate, the miner who mines the transaction can do it for her. This may be convenient when constructing a CoinJoin transaction.

In all cases above, miners can choose to further aggregate signatures across different transactions using the standard BLS signature aggregation mechanism in (1). This will further shrink the block size.

Compatibility with transaction validation caching. Recall that transaction validation caching is a process whereby a node validates transactions as they are added to the node's mempool and marks them as validated. When a previously validated transaction appears in a new block, the node need not re-validate the transaction. Transaction validation caching is compatible with signature aggregation across multiple transactions in a block. To see why, consider a node that receives a new block containing an aggregate signature $\sigma = \sigma_1 \cdots \sigma_n$, aggregated over n transactions in the block. The node can identify all the transactions in this new block that are already marked as validated in its mempool, and divide σ by the signatures associated with these pre-validated transactions. Effectively, the pre-validated signatures are removed from the aggregate signature σ. Let σ' be the resulting aggregate signature. Now the node need only check that σ' is a valid aggregate signature over the remaining transactions in the block, namely those transactions that are not already in its mempool.

3. Proving security

Security for a multi-signature scheme is defined using the following game between a challenger and an adversary $\mathcal{A}$.

1. **Setup**. The challenger runs KeyGen to generate a key pair pk, sk and sends pk to the adversary.
2. **Signature queries**. The adversary issues adaptive chosen message queries $m_1, m_2, \ldots \in \mathcal{M}$ and receives back the signatures $\sigma_i = \text{Sign}(sk, m_i)$ for $i = 1, 2, \ldots$ from the challenger.
3. **Forgery**. Eventually, the adversary outputs a forgery: it outputs public keys $pk_1, \ldots, pk_n$, a message $m \in \mathcal{M}$, and a multi-signature σ.

The adversary wins the game if it did not issue a signature query for m and $\text{Verify}(pk, pk_1, \ldots, pk_n, m, \sigma)$ outputs "accept". We could allow the challenge pk to appear anywhere in the tuple of public keys given to Verify, but to simplify the notation we require that it be the left most element. The same analysis applies if we allow pk to appear elsewhere. We use

$$\text{SIGadv}[\mathcal{A}, \mathcal{S};\ Q_{\text{sig}}, Q_{H_0}, Q_{H_1}]$$

to denote the adversary's advantage in attacking the scheme $\mathcal{S}$, for an adversary that makes at most Q_{sig} signature queries, at most Q_{H_0} queries to H_0, and at most Q_{H_1} queries to H_1. We say that the scheme $\mathcal{S}$ is secure if for all efficient adversaries the advantage is negligible.

The co-CDH assumption. Security of the scheme $\mathcal{S}$ from Section 2 relies on the standard co-CDH assumption in the bilinear group $(\mathbb{G}_0, \mathbb{G}_1)$. The assumption states that for all efficient algorithms $\mathcal{A}$ the quantity

$$\text{CDHadv}[\mathcal{A}, (\mathbb{G}_0, \mathbb{G}_1)] := \Pr\left[\mathcal{A}(g_1^{\alpha},\ g_0^{\beta},\ g_0^{\alpha}) = g_0^{\alpha\beta}\right]$$

is negligible, where $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$.

3.1 Proving security of multi-sig BLS

We prove security of the scheme $\mathcal{S}$ from Section 2 without requiring signers to prove knowledge of their secret key. We will prove security of a slightly modified scheme $\mathcal{S}'$. The only difference is that there is an additional element in the public key. Specifically, key generation in $\mathcal{S}'$ outputs $pk = (g_1^\alpha, g_0^\alpha)$. However, in the security game, when the adversary outputs the final forgery, which includes a list of public keys, it only needs to output the left term of each public key, as when attacking $\mathcal{S}$. Clearly if $\mathcal{S}'$ is secure then so is $\mathcal{S}$.

The proof is done in three short steps, captured in the following three theorems:

1. We show that a general attacker $\mathcal{A}_1$ on $\mathcal{S}'$ gives an attacker $\mathcal{A}_2$ on $\mathcal{S}'$ that makes no chosen message queries.
2. we show that an attacker $\mathcal{A}_2$ on $\mathcal{S}'$ that makes no chosen message queries but potentially many queries to H_1, gives an attacker $\mathcal{A}_3$ on $\mathcal{S}'$ that makes only one query to H_1.
3. we show than an attacker $\mathcal{A}_3$ on $\mathcal{S}'$ that makes no chosen message queries and only one query to H_1, can be used to break co-CDH.

Putting these three steps together proves security of $\mathcal{S}'$ and therefore of $\mathcal{S}$. The third step is the most interesting so we present that step first, then the second step, and finally the general result in the first step.

Theorem 1: Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}'$ that makes no chosen message queries and at most one query to H_1. Let

$$\epsilon = \text{SIGadv}[\mathcal{A}, \mathcal{S}';\ 0, Q_{H_0}, 1]$$

be its advantage. Then there exists an adversary $\mathcal{B}$ for computing co-CDH, whose running time is about twice that of $\mathcal{A}$, with advantage $\epsilon' = \text{CDHadv}[\mathcal{B}, (\mathbb{G}_0, \mathbb{G}_1)]$ such that

$$\epsilon' \geq \epsilon^2 - \epsilon/N.$$

Here $N = |R|$, the size of one coordinate in the image of H_1. This implies

$$\epsilon \leq (1/N) + \sqrt{\epsilon'}.$$

Proof. Algorithm $\mathcal{B}$ is a given a co-CDH instance $(h = g_1^\alpha,\ u = g_0^\beta,\ \hat{h} = g_0^\alpha) \in \mathbb{G}_1 \times G_0^2$. It needs to compute $g_0^{\alpha\beta}$. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ and gives it the public key $pk = (h, \hat{h})$. Now $\mathcal{A}$ can issue many queries to H_0 and a single query to H_1. For $i = 1, 2, \ldots$ algorithm $\mathcal{B}$ responds to query number i as follows:

- $\mathcal{B}$ responds to a query for $H_0(m_i)$, where $m_i \in \mathcal{M}$, by choosing a random $\rho_i \xleftarrow{R} \mathbb{Z}_q$ and setting $H_0(m_i) \leftarrow u^{\rho_i}$.
- $\mathcal{B}$ responds to a (single) query for $H_1(h_0, h_1, \ldots, h_n)$, where $(h_0, \ldots, h_n) \in G_1^{n+1}$, by choosing a random $(t_0, \ldots, t_n) \xleftarrow{R} R^{n+1}$ and setting $H_1(h_0, h_1, \ldots, h_n) \leftarrow (t_0, \ldots, t_n)$.

Suppose that eventually $\mathcal{A}$ outputs a valid forgery. We can assume that the public keys in the forgery are the public keys in the query to H_1, since otherwise the forgery cannot be valid. In other words, the forgery is a tuple $(h_1, \ldots, h_n, m, \sigma)$ such that

$$e(g_1, \sigma) = e\left(h^{t_0} h_1^{t_1} \cdots h_n^{t_n},\ H_0(m)\right).$$

Moreover, $\mathcal{B}$ knows a $\rho \in \mathbb{Z}_q$ such that $H_0(m) = u^\rho$ and therefore

$$e(g_1, \sigma) = e\left(h^{t_0} h_1^{t_1} \cdots h_n^{t_n},\ u^\rho\right) = e(h, u)^{t_0 \rho} \cdot e\Big(h_1^{t_1} \cdots h_n^{t_n},\ u\Big)^\rho. \tag{4}$$

Next, $\mathcal{B}$ rewinds $\mathcal{A}$ to the point where it issued the query to $H_1(h, h_1, \ldots, h_n)$. This time algorithm $\mathcal{B}$ responds by choosing a fresh $t_0' \xleftarrow{R} R$ and sets

$$H_1(h, h_1, \ldots, h_n) \leftarrow (t_0', t_1, \ldots, t_n).$$

Suppose that again $\mathcal{A}$ outputs a valid forgery $(h_1, \ldots, h_n, m', \sigma')$. As before, $\mathcal{B}$ has $\rho' \in \mathbb{Z}_q$ such that $H_0(m') = u^{\rho'}$ and

$$e(g_1, \sigma') = e\left(h^{t_0'} h_1^{t_1} \cdots h_n^{t_n},\ u^{\rho'}\right) = e(h, u)^{t_0 \rho'} \cdot e\Big(h_1^{t_1} \cdots h_n^{t_n},\ u\Big)^{\rho'}. \tag{5}$$

Then raising equation (5) to the power of ρ and dividing by equation (4) raised to the power of ρ' leads to

$$e\left(g_1,\ (\sigma')^{\rho}/\sigma^{\rho'}\right) = e(h,u)^{\rho\rho'(t_0'-t_0)}.$$

Let us assume that $t_0 \neq t_0'$. Now, because $e(h,u) = e(g_1, g_0^{\alpha\beta})$ it follows that

$$g_0^{\alpha\beta} = \left((\sigma')^{\rho}/\sigma^{\rho'}\right)^{1/\rho\rho'(t_0'-t_0)}.$$

which is the solution to the given co-CDH challenge.

We see that $\mathcal{B}$ solves the given co-CDH instance whenever $t_0 \neq t_0'$ and $\mathcal{A}$ outputs a valid forgery in both runs, so that both (4) and (5) hold. To calculate the probability that this happens we use the following simple lemma, called the *rewinding lemma* (a variant of Lemma 19.2 in the cryptography book).

Rewinding lemma: Let S, R, and T be finite, non-empty sets, and let $f : S \times R \times T \to \{0,1\}$ be a function. Let X and Y, Y' and Z, Z' be mutually independent random variables, where X takes values in the set S, the variables Y and Y' are each uniformly distributed over R, and Z and Z' take values in the set T. Let $\epsilon := \Pr[f(X,Y,Z) = 1]$ and $N := |R|$. Then

$$\Pr\left[f(X,Y,Z) = 1 \wedge f(X,Y',Z') = 1 \wedge Y \neq Y'\right] \geq \epsilon^2 - \epsilon/N.$$

To apply the lemma, let X be all the quantities given to $\mathcal{A}$ up to and including the query to H_1 and its response, excluding t_0. Let Y and Y' be t_0 and t_0', respectively. Let Z be all the quantities given to $\mathcal{A}$ in the first run following the query to H_1. Similarly, let Z' be all the quantities given to $\mathcal{A}$ in the second run following the query to H_1. Then (X,Y,Z) are the random values given to $\mathcal{A}$ in the first run, and (X,Y',Z') are the random values given to $\mathcal{A}$ in the second run. The function $f(X,Y,Z)$ is 1 whenever $\mathcal{A}$ produces a valid forgery given the quantities X, Y, Z. The rewinding lemma now gives the bounds stated in the theorem, and completes the proof. ∎

Theorem 2: Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}'$ that makes no chosen message queries but potentially many queries to H_1. Then there exists an adversary $\mathcal{B}$ attacking $\mathcal{S}'$, that makes only a single query to H_1, and whose running time is about the same as $\mathcal{A}$, such that

$$\text{SIGadv}[\mathcal{A}, \mathcal{S}';\ 0, Q_{H_0}, Q_{H_1}] \leq Q_{H_1} \cdot \text{SIGadv}[\mathcal{B}, \mathcal{S}';\ 0, Q_{H_0}, 1].$$

Proof. The proof is a simple guessing argument. Adversary $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$ and interacts with its own challenger. $\mathcal{B}$ relays all messages between its challenger and $\mathcal{A}$. However, $\mathcal{A}$ can make Q_{H_1} queries to H_1 whereas $\mathcal{B}$ can only make a single H_1 query to its challenger. To address this, $\mathcal{B}$ will answer all of $\mathcal{A}$'s queries to H_1 by generating random responses itself. In addition, $\mathcal{B}$ will choose one of $\mathcal{A}$'s queries to H_1 at random and forward it to its own challenger. If $\mathcal{A}$ uses the arguments of that chosen query in its signature forgery, then $\mathcal{B}$ succeeds in generating a forgery to its own challenger. This happens with probability $1/Q_{H_1}$, and the theorem follows. ∎

Theorem 3: Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}'$. Then there exists an adversary $\mathcal{B}$ attacking $\mathcal{S}'$, that makes no chosen message queries and whose running time is about the same as $\mathcal{A}$, such that

$$\text{SIGadv}[\mathcal{A}, \mathcal{S}';\ Q_{\text{sig}}, Q_{H_0}, Q_{H_1}] \leq (e \cdot Q_{\text{sig}}) \cdot \text{SIGadv}[\mathcal{B}, \mathcal{S}';\ 0, Q_{H_0}, Q_{H_1}] \tag{6}$$

where $e \approx 2.71$.

Proof. Adversary $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$ and interacts with its own challenger. First, $\mathcal{B}$ receives a public key $pk = \left(h = g_1^{\alpha},\ \hat{h} = g_0^{\alpha}\right)$ from its challenger, which it forwards to $\mathcal{A}$. Now, $\mathcal{A}$ issues queries and $\mathcal{B}$ responds. For $i = 1, 2, \ldots$, adversary $\mathcal{B}$ responds to query number i from $\mathcal{A}$ as follows:

- if the query is to H_1 then $\mathcal{B}$ forwards the query to its challenger and forwards the response to $\mathcal{A}$.
- if the query is for $H_0(m_i)$ then $\mathcal{B}$ does the following:
  1. $\mathcal{B}$ generates a biased bit $b_i \in \{0,1\}$ where $\Pr[b_i = 1] = 1/Q_{\text{sig}}$.
  2. if $b_i = 1$ then $\mathcal{B}$ forwards the query to its challenger and sends the response $H_0(m_i)$ to $\mathcal{A}$.
  3. if $b_i = 0$ then $\mathcal{B}$ responds by choosing a random $\rho_i \xleftarrow{R} \mathbb{Z}_q$ and responding to $\mathcal{A}$ by setting $H_0(m_i) \leftarrow g_0^{\rho_i}$.

- if the query is a chosen message query for message $m_i \in \mathcal{M}$, we may assume without loss of generality that $\mathcal{A}$ had already issued a query for $H_0(m_i)$ in some previous query $j < i$. If $b_j = 1$ then $\mathcal{B}$ aborts and outputs "fail". Otherwise, we know that $H_0(m_i) = g_0^{\rho_j}$ and $\mathcal{B}$ responds with the signature $\sigma_i = \hat{h}^{\rho_j}$, which is a valid signature on m_i. This step is the reason we need $\hat{h}$ in the public key.

Suppose that $\mathcal{B}$ does not abort and that $\mathcal{A}$ eventually outputs a valid forgery $(h_1, \ldots, h_n, m, \sigma)$. The adversary must have issued a query for $H_0(m)$, say query number j. If $b_j = 0$ then $\mathcal{B}$ aborts and outputs "fail". Otherwise, $\mathcal{B}$ outputs $(h_1, \ldots, h_n, m, \sigma)$ as a valid forgery.

$\mathcal{B}$ successfully answers all signature queries with probability $(1 - 1/Q_{\text{sig}})^{Q_{\text{sig}}} \geq 1/e$. Yet $\mathcal{B}$ never issued a chosen message query to its own challenger, as required. The final forgery will be accepted by $\mathcal{B}$ with probability $1/Q_{\text{sig}}$. Hence, $\mathcal{B}$ will output a forgery with probability at least $1/(eQ_{\text{sig}})$. Moreover, when $\mathcal{B}$ outputs a forgery, the forgery is valid with respect to the challenger's definition of H_0 with exactly $\mathcal{A}$'s forging advantage. Hence,

$$\text{SIGadv}[\mathcal{B}, \mathcal{S}';\ 0, Q_{H_0}, Q_{H_1}] \geq (1/e \cdot Q_{\text{sig}}) \cdot \text{SIGadv}[\mathcal{A}, \mathcal{S}';\ Q_{\text{sig}}, Q_{H_0}, Q_{H_1}]$$

from which the theorem follows. ∎

Putting it all together. Putting Theorems 1,2,3 together proves security of $\mathcal{S}'$, and therefore $\mathcal{S}$, assuming the co-CDH assumption holds in the bilinear group $(\mathbb{G}_0, \mathbb{G}_1)$. Concretely, we obtain the following corollary.

Corollary: For every adversary $\mathcal{A}$ attacking $\mathcal{S}$ there is a co-CDH algorithm $\mathcal{B}$, whose running time is about twice that of $\mathcal{A}$, such that

$$\text{SIGadv}[\mathcal{A}, \mathcal{S}';\ Q_{\text{sig}}, Q_{H_0}, Q_{H_1}] \leq (eQ_{\text{sig}}Q_{H_1}) \cdot \sqrt{\epsilon} + (eQ_{\text{sig}}Q_{H_1})/N$$

where $N = |R|$ and $\epsilon = \text{CDHadv}[\mathcal{A}, (\mathbb{G}_0, \mathbb{G}_1)]$.

4. Concluding remarks

We presented a modified same-message aggregation mechanism for BLS signatures that supports efficient verification, requiring only two pairings to verify a multi-signature. The point is that security holds without requiring proofs of knowledge of the secret key.

We conclude with the following remark: the proof of security for BLS aggregation for *distinct* messages makes use of an isomorphism $\psi : \mathbb{G}_1 \to \mathbb{G}_0$. While in many pairing instantiations this ψ exists naturally, in some instantiations it does not. When ψ does not exist, the proof of security still applies, but relative to a slightly stronger assumption than co-CDH. Specifically, we need co-CDH to hold even if the adversary has access to an oracle for ψ. We call this the ψ-co-CDH assumption, and it appears to hold whenever co-CDH holds. Hence, BLS aggregation for distinct messages can be safely used even when there is no efficient algorithm to compute ψ. We note that the security proof in this writeup does not make use of ψ, and therefore this issue does not come up.

Bibliography

[BLS01] Dan Boneh, Ben Lynn, Hovav Shacham. Short Signatures from the Weil Pairing. ASIACRYPT 2001. pages 514-532. See also J. Cryptology 17(4): 297-319 (2004).

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In Eli Biham, editor, Advances in Cryptology - EUROCRYPT 2003, volume 2656 of LNCS, pages 416–432. Springer, 2003.

[BN06] Mihir Bellare and Gregory Neven. Multi-Signatures in the Plain Public Key Model and a General Forking Lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, ACM Conference on Computer and Communications Security - CCS 2006, pages 390–399. ACM, 2006.

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted Aggregate Signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, Automata, Languages and Programming - ICALP 2007, volume 4596 of LNCS, pages 411–422. Springer, 2007

[Bol03] Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In Yvo Desmedt, editor, Public Key Cryptography - PKC 2003, volume 2567 of LNCS, pages 31–46. Springer, 2003.

[IN83] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. NEC Research and Development,

71:1–8, 1983.

[LOS06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential Aggregate Signatures and Multisignatures Without Random Oracles. In Serge Vaudenay, editor, Advances in Cryptology - EUROCRYPT 2006, volume 4004 of LNCS, pages 465–485. Springer, 2006.

[MPSW18] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, Pieter Wuille. Simple Schnorr Multi-Signatures with Applications to Bitcoin. Cryptology ePrint Archive, Report 2018/068, 2018.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-Subgroup Multisignatures. In Michael K. Reiter and Pierangela Samarati, editors, ACM Conference on Computer and Communications Security - CCS 2001, pages 245–254. ACM, 2001.

[RY07] Thomas Ristenpart and Scott Yilek. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In Moni Naor, editor, Advances in Cryptology - EUROCRYPT 2007, volume 4515 of LNCS, pages 228–245. Springer, 2007.

Dan's home page, CS Department, Stanford University