

# Sequential Aggregate Signatures Made Shorter

Kwangsung Lee<sup>1,\*</sup>, Dong Hoon Lee<sup>1,\*\*</sup>, and Moti Yung<sup>2,3</sup>

<sup>1</sup> Korea University, Korea

{guspun,donghlee}@korea.ac.kr

<sup>2</sup> Columbia University, USA

moti@cs.columbia.edu

<sup>3</sup> Google Inc., USA

**Abstract.** Sequential aggregate signature (SAS) is a special type of public-key signature that allows a signer to add his signature into a previous aggregate signature in sequential order. In this case, since many public keys are used and many signatures are employed and compressed, it is important to reduce the sizes of signatures and public keys. Recently, Lee et al. proposed an efficient SAS scheme with short public keys and proved its security without random oracles under static assumptions. In this paper, we propose an improved SAS scheme that has a shorter signature size compared with that of Lee et al.'s SAS scheme. Our SAS scheme is also secure without random oracles under static assumptions. To achieve the improvement, we devise a new public-key signature scheme that supports multi-users and public re-randomization. Compared with the SAS scheme of Lee et al., our SAS scheme employs new techniques which allow us to reduce the size of signatures by increasing the size of the public keys (obviously, since signature compression is at the heart of aggregate signature this is a further step in understanding the aggregation capability of such schemes).

## 1 Introduction

Aggregate signature is a relatively new type of public-key signature (PKS) that allows a signer to aggregate different signatures generated by different signers on different messages into a short aggregate signature [6]. Aggregate signature has many applications like signing certificate chains, proxy signing, secure routing protocols, and more. After the introduction of aggregate signature by Boneh, Gentry, Lynn, and Shacham [6], many aggregate signature schemes were proposed by using bilinear groups [1, 2, 4, 6, 10, 11, 13, 15, 17, 21] and trapdoor permutations [7, 18, 20]. However, the security of many aggregate signature schemes was proven in the random oracle model. The random oracle model was very

---

\* Supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2012-H0301-12-3007) supervised by the NIPA (National IT Industry Promotion Agency).

\*\* Supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0029121).

successful to prove the security of practical schemes, but the security proof in the random oracle model is not entirely sound [8] and schemes in the standard model are needed. Standard model solutions for the cases of sequential aggregate signature (introduced in [18]) [13, 15, 17, 21] (where signatures are aggregated in a sequence, as in applications like certification chains), and synchronized aggregate signature (where all signers share a synchronized same value, as introduced by [10]) [1] were given.

A sequential aggregate signature (SAS) scheme without random oracle assumption is what we concentrate on here, such a scheme was first proposed by Lu et al. [17], but the public-key size of this scheme is too large since the scheme is based on the PKS scheme of Waters [22]. In public-key based aggregate signature, reducing the size of public keys is very important since a verifier should retrieve all the public keys of signers to check the validity of the aggregate signature, and needless to say the size of the aggregated signature is important as well. The importance of constructing a SAS scheme with short public keys was addressed by Lu et al. [17], but they left it as an interesting open problem. Schröder proposed the first SAS scheme with short public keys based on the Camenisch-Lysyanskaya (CL) signature scheme [21], but it is only secure under the interactive LRSW assumption. Recently, Lee et al. [15] proposed another SAS scheme with short public keys based on the identity-based encryption (IBE) scheme of Lewko and Waters [16] and proved its security without random oracles under static assumptions.

## 1.1 Our Contributions

In this paper, we revisit the SAS scheme of Lee et al. [15] and propose an improved SAS scheme with shorter signature size. The proposed SAS scheme trades off signature for public-key size since the signature size of our SAS scheme is shorter than that of Lee et al.'s SAS scheme by two group elements but the public-key size of our SAS scheme is longer by two group elements. To construct the SAS scheme with shorter signature size that supports sequential aggregation, we first propose a new PKS scheme and prove its security without random oracles under static assumptions. Additionally, we propose a multi-signature (MS) scheme with shorter signature size and shorter public parameters and prove its security without random oracles under static assumptions.

We suggest new ideas, and technically speaking, we construct a PKS scheme that supports multi-users and public re-randomization for a SAS scheme with shorter signature size. We start the construction from the PKS scheme derived from the IBE scheme of Lewko and Waters [16] (as was done earlier). However, this directly converted PKS scheme does not support multi-users and public re-randomization as pointed out by Lee et al. [15] since the elements  $g, u, h \in \mathbb{G}$  cannot be published in the public key. Lee et al. solved this problem by modifying the verification algorithm of the PKS scheme, but the size of signatures increased by two group elements. In this paper, we solve this obstacle in a different way and publish  $gw_1^{c_g}, uw_1^{c_u}, hw_1^{c_h} \in \mathbb{G}$  in the public key instead of publishing  $g, u, h \in \mathbb{G}$  to maintain the same size of signatures (loosely speaking, we lift the verification

parameters to the exponent). However, note that this method increases the size of public keys by two group elements compared with that of Lee et al.'s scheme since additional group elements should be published in the public key to make public  $gw_1^{c_g}, uw_1^{c_u}, hw_1^{c_h}$ .

## 1.2 Related Work

**Aggregate Signature.** The concept of aggregate signatures was introduced by Boneh et al. [6], and they proposed the first aggregate signature scheme in bilinear groups. Their aggregate signature scheme is the only unique one that supports full aggregation, but the security is proven in the random oracle model and the verification algorithm requires  $l$  number of pairing where  $l$  is the number of signers in the aggregate signature. To remedy this situation, other types of aggregate signatures were introduced.

Lysyanskaya et al. [18] introduced the concept of sequential aggregate signature (SAS) and proposed a SAS scheme in trapdoor permutations. Lu et al. [17] proposed the first SAS scheme without random oracles, but the size of public keys is very large. To reduce the size of public keys, SAS schemes with short public key was proposed [13, 15, 21]. Recently, SAS schemes that do not require a verifier to check the validity of the previous signature were proposed [7, 9]. Boldyreva et al. [4] proposed an identity-based sequential aggregate signature scheme in bilinear groups and proved its security under an interactive assumption. Recently Gerbush et al. [11] proposed a modified identity-based sequential aggregate signature scheme in composite order bilinear groups and proved its security in the random oracle model under static assumptions.

Gentry and Ramzan [10] introduced the concept of synchronized aggregate signature and proposed an identity-based synchronized aggregate signature scheme in the random oracle model. Ahn et al. [1] proposed a synchronized aggregate signature scheme and proved its security without random oracles. Recently, Lee et al. [13] proposed a synchronized aggregate signature scheme with shorter aggregate signatures based on the CL signature and proved its security in the random oracle model.

**Multi-signature.** The concept of multi-signature (MS) was introduced by Itakura and Nakamura [12]. MS is a special type of aggregate signatures where all signers generate signatures for the same message. Micali et al. [19] defined the first formal security model of MS and proposed a MS scheme based on the Schnorr signature. Boldyreva defined a general security model for multi-signatures and proposed a MS scheme in bilinear groups that is secure in the random oracle model [3]. Lu et al. [17] proposed the first MS scheme that is secure without random oracles by modifying their SAS scheme. Recently, Lee et al. [15] proposed a MS scheme with short public parameters and proved its security without random oracles.

## 2 Preliminaries

In this section, we define asymmetric bilinear groups and introduce complexity assumptions in this bilinear groups.

### 2.1 Asymmetric Bilinear Groups

Let  $\mathbb{G}, \hat{\mathbb{G}}$  and  $\mathbb{G}_T$  be multiplicative cyclic groups of prime order  $p$ . Let  $g, \hat{g}$  be generators of  $\mathbb{G}, \hat{\mathbb{G}}$ . The bilinear map  $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$  has the following properties:

1. Bilinearity:  $\forall u \in \mathbb{G}, \forall \hat{v} \in \hat{\mathbb{G}}$  and  $\forall a, b \in \mathbb{Z}_p, e(u^a, \hat{v}^b) = e(u, \hat{v})^{ab}$ .
2. Non-degeneracy:  $\exists g, \hat{g}$  such that  $e(g, \hat{g})$  has order  $p$ , that is,  $e(g, \hat{g})$  is a generator of  $\mathbb{G}_T$ .

We say that  $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$  are bilinear groups with no efficiently computable isomorphisms if the group operations in  $\mathbb{G}, \hat{\mathbb{G}}$ , and  $\mathbb{G}_T$  as well as the bilinear map  $e$  are all efficiently computable, but there are no efficiently computable isomorphisms between  $\mathbb{G}$  and  $\hat{\mathbb{G}}$ .

### 2.2 Complexity Assumptions

We employ three static assumptions in prime order (asymmetric) bilinear groups. Assumptions 1 and 2 were introduced by Lewko and Waters [16], while Assumption 3 has been used extensively.

**Assumption 1 (LW1).** Let  $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$  be a description of the asymmetric bilinear group of prime order  $p$  with the security parameter  $\lambda$ . Let  $g, \hat{g}$  be generators of  $\mathbb{G}, \hat{\mathbb{G}}$  respectively. The assumption is that if the challenge values

$$D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^b, \hat{g}, \hat{g}^a, \hat{g}^b, \hat{g}^{ab^2}, \hat{g}^{b^2}, \hat{g}^{b^3}, \hat{g}^c, \hat{g}^{ac}, \hat{g}^{bc}, \hat{g}^{b^2c}, \hat{g}^{b^3c}) \text{ and } T$$

are given, no PPT algorithm  $\mathcal{B}$  can distinguish  $T = T_0 = \hat{g}^{ab^2c}$  from  $T = T_1 = \hat{g}^d$  with more than a negligible advantage. The advantage of  $\mathcal{B}$  is defined as  $\text{Adv}_{\mathcal{B}}^{A1}(\lambda) = |\Pr[\mathcal{B}(D, T_0) = 0] - \Pr[\mathcal{B}(D, T_1) = 0]|$  where the probability is taken over the random choice of  $a, b, c, d \in \mathbb{Z}_p$ .

**Assumption 2 (LW2).** Let  $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$  be a description of the asymmetric bilinear group of prime order  $p$ . Let  $g, \hat{g}$  be generators of  $\mathbb{G}, \hat{\mathbb{G}}$  respectively. The assumption is that if the challenge values

$$D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^a, g^b, g^c, \hat{g}, \hat{g}^a, \hat{g}^{a^2}, \hat{g}^{bx}, \hat{g}^{ax}, \hat{g}^{a^2x}) \text{ and } T$$

are given, no PPT algorithm  $\mathcal{B}$  can distinguish  $T = T_0 = g^{bc}$  from  $T = T_1 = g^d$  with more than a negligible advantage. The advantage of  $\mathcal{B}$  is defined as  $\text{Adv}_{\mathcal{B}}^{A2}(\lambda) = |\Pr[\mathcal{B}(D, T_0) = 0] - \Pr[\mathcal{B}(D, T_1) = 0]|$  where the probability is taken over the random choice of  $a, b, c, x, d \in \mathbb{Z}_p$ .

**Assumption 3 (Decisional Bilinear Diffie-Hellman).** Let  $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$  be a description of the asymmetric bilinear group of prime order  $p$ . Let  $g, \hat{g}$  be generators of  $\mathbb{G}, \hat{\mathbb{G}}$  respectively. The assumption is that if the challenge values

$$D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^a, g^b, g^c, \hat{g}, \hat{g}^a, \hat{g}^b, \hat{g}^c) \text{ and } T$$

are given, no PPT algorithm  $\mathcal{B}$  can distinguish  $T = T_0 = e(g, \hat{g})^{abc}$  from  $T = T_1 = e(g, \hat{g})^d$  with more than a negligible advantage. The advantage of  $\mathcal{B}$  is defined as  $\mathbf{Adv}_{\mathcal{B}}^{\text{A3}}(\lambda) = |\Pr[\mathcal{B}(D, T_0) = 0] - \Pr[\mathcal{B}(D, T_1) = 0]|$  where the probability is taken over the random choice of  $a, b, c, d \in \mathbb{Z}_p$ .

### 3 Public-Key Signature

In this section, we propose an efficient public-key signature (PKS) scheme with short public keys that supports multi-users and public re-randomization, and prove its security without random oracles under static assumption.

#### 3.1 Construction

To construct a PKS scheme with short public keys that supports multi-users and public re-randomization, we can derive a PKS scheme with short public keys from the IBE scheme in prime order groups of Lewko and Waters [16] by applying the transformation of Naor [5] and representing the signature in  $\mathbb{G}$  to reduce the size of signatures. However, this PKS scheme does not support multi-users and public re-randomization since the elements  $g, u, h \in \mathbb{G}$  cannot be published in the public key. Lee et al. [15] solved this problem by re-randomizing the verification elements of the signature verification algorithm, but the number of signatures increased by two group elements, and our main issue here is further compression of the signature size.

To this end, we present another solution for the above problem that allows the elements  $g, u, h$  to be safely published in the public key. In the PKS scheme of Lewko and Waters [16], if  $g, u, h \in \mathbb{G}$  are published in the public key, then the simulator of the security proof can easily distinguish normal verification components from semi-functional verification components of the signature verification algorithm for a forged signature without the help of an adversary. Thus the simulator of Lewko and Waters sets the CDH value into the elements  $g, u, h$  to prevent the simulator from creating these elements. Our idea for solving this problem is to lift the published values into the exponent and publish  $gw_1^{c_g}, uw_1^{c_u}, hw_1^{c_h}$  that are additionally multiplied with random elements instead of directly publishing  $g, u, h$ . In this case, the simulator can create these elements since the random exponents  $c_g, c_u, c_h$  can be used to cancel out the CDH value embedded in the elements  $g, u, h$ . Additionally, the simulator cannot distinguish the changes of verification components for the forged signature because of the added elements  $w_1^{c_g}, w_1^{c_u}, w_1^{c_h}$ . This solution does not increase the number of group elements in the signatures, rather it increases the number of public keys since additional elements  $w_2^{c_g}, w_2^{c_u}, w_2^{c_h}, w_2^{c_g}, w_2^{c_u}, w_2^{c_h}$  should be published.

Our PKS scheme in prime order bilinear groups is described as follows:

**PKS.KeyGen**( $1^\lambda$ ): This algorithm first generates the asymmetric bilinear groups  $\mathbb{G}, \hat{\mathbb{G}}$  of prime order  $p$  of bit size  $\Theta(\lambda)$ . It chooses random elements  $g, w \in \mathbb{G}$  and  $\hat{g} \in \hat{\mathbb{G}}$ . Next, it selects random exponents  $\nu, \phi_1, \phi_2 \in \mathbb{Z}_p$  and sets  $\tau = \phi_1 + \nu\phi_2$ . It also selects random exponents  $\alpha, x, y \in \mathbb{Z}_p$  and sets  $u = g^x, h = g^y, \hat{u} = \hat{g}^x, \hat{h} = \hat{g}^y, w_1 = w^{\phi_1}, w_2 = w^{\phi_2}$ . It outputs a private key  $SK = (\alpha, x, y)$  and a public key by selecting random values  $c_g, c_u, c_h \in \mathbb{Z}_p$  as

$$PK = ( gw_1^{c_g}, w_2^{c_g}, w^{c_g}, uw_1^{c_u}, w_2^{c_u}, w^{c_u}, hw_1^{c_h}, w_2^{c_h}, w^{c_h}, w_1, w_2, w, \hat{g}, \hat{g}^\nu, \hat{g}^{-\tau}, \hat{u}, \hat{u}^\nu, \hat{u}^{-\tau}, \hat{h}, \hat{h}^\nu, \hat{h}^{-\tau}, \Lambda = e(g, \hat{g}), \Omega = e(g, \hat{g})^\alpha ).$$

**PKS.Sign**( $M, SK$ ): This algorithm takes as input a message  $M \in \mathbb{Z}_p$  and a private key  $SK = (\alpha, x, y)$ . It selects random exponents  $r, c_1, c_2 \in \mathbb{Z}_p$  and outputs a signature as

$$\begin{aligned} \sigma = ( & W_{1,1} = (gw_1^{c_g})^\alpha ((w_1^{c_u})^M (hw_1^{c_h}))^r w_1^{c_1}, \\ & W_{1,2} = (w_2^{c_g})^\alpha ((w_2^{c_u})^M w_2^{c_h})^r w_2^{c_1}, W_{1,3} = (w^{c_g})^\alpha ((w^{c_u})^M w^{c_h})^r w^{c_1}, \\ & W_{2,1} = (gw_1^{c_g})^r w_1^{c_2}, W_{2,2} = (w_2^{c_g})^r w_2^{c_2}, W_{2,3} = (w^{c_g})^r w^{c_2} ). \end{aligned}$$

**PKS.Verify**( $\sigma, M, PK$ ): This algorithm takes as input a signature  $\sigma$  on a message  $M \in \mathbb{Z}_p$  under a public key  $PK$ . It chooses a random exponent  $t \in \mathbb{Z}_p$  and computes verification components as

$$\begin{aligned} V_{1,1} &= \hat{g}^t, V_{1,2} = (\hat{g}^\nu)^t, V_{1,3} = (\hat{g}^{-\tau})^t, \\ V_{2,1} &= (\hat{u}^M \hat{h})^t, V_{2,2} = ((\hat{u}^\nu)^M \hat{h}^\nu)^t, V_{2,3} = ((\hat{u}^{-\tau})^M \hat{h}^{-\tau})^t. \end{aligned}$$

Next, it verifies that  $\prod_{i=1}^3 e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^3 e(W_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$ . If this equation holds, then it outputs 1. Otherwise, it outputs 0.

If we implicitly sets  $\tilde{c}_1 = c_g\alpha + (c_u M + c_h)r + c_1$ ,  $\tilde{c}_2 = c_g r + c_2$ , then the signature is restated as the following form

$$\begin{aligned} W_{1,1} &= g^\alpha (u^M h)^r w_1^{\tilde{c}_1}, W_{1,2} = w_2^{\tilde{c}_1}, W_{1,3} = w^{\tilde{c}_1}, \\ W_{2,1} &= g^r w_1^{\tilde{c}_2}, W_{2,2} = w_2^{\tilde{c}_2}, W_{2,3} = w^{\tilde{c}_2}. \end{aligned}$$

### 3.2 Security Analysis

We prove the security of our PKS scheme without random oracles under static assumptions. To prove the security, we use the dual system encryption technique of Lewko and Waters [16]. The dual system encryption technique was originally developed to prove the full-model security of IBE and its extensions, but it also can be used to prove the security of PKS by using the transformation of Naor [5]. Recently Lee et al. [15] proved the security of their PKS scheme by using the dual system encryption technique, and Gerbush et al. [11] developed the dual form signature technique that is a variation of the dual system encryption technique to prove the security of their PKS schemes.

**Theorem 1.** *The above PKS scheme is existentially unforgeable under a chosen message attack if Assumptions 1, 2, and 3 hold.*

*Proof.* Before proving the security, we first define two additional algorithms for semi-functional types. For the semi-functionality, we set  $f = g^{y_f}$ ,  $\hat{f} = \hat{g}^{y_f}$  where  $y_f$  is a random exponent in  $\mathbb{Z}_p$ .

**PKS.SignSF.** The semi-functional signing algorithm first creates a normal signature using the private key. Let  $(W'_{1,1}, \dots, W'_{2,3})$  be the normal signature of a message  $M$  with random exponents  $r, c_1, c_2 \in \mathbb{Z}_p$ . It selects random exponents  $s_k, z_k \in \mathbb{Z}_p$  and outputs a semi-functional signature as

$$\sigma = ( W_{1,1} = W'_{1,1} \cdot (f^{-\nu})^{s_k z_k}, W_{1,2} = W'_{1,2} \cdot f^{s_k z_k}, W_{1,3} = W'_{1,3}, \\ W_{2,1} = W'_{2,1} \cdot (f^{-\nu})^{s_k}, W_{2,2} = W'_{2,2} \cdot f^{s_k}, W_{2,3} = W'_{2,3} ).$$

**PKS.VerifySF.** The semi-functional verification algorithm first creates a normal verification components using the public key. Let  $(V'_{1,1}, \dots, V'_{2,3})$  be the normal verification components with a random exponent  $t \in \mathbb{Z}_p$ . It chooses random exponents  $s_c, z_c \in \mathbb{Z}_p$  and computes semi-functional verification components as

$$V_{1,1} = V'_{1,1}, V_{1,2} = V'_{1,2} \cdot \hat{f}^{s_c}, V_{1,3} = V'_{1,3} \cdot (\hat{f}^{-\phi_2})^{s_c}, \\ V_{2,1} = V'_{2,1}, V_{2,2} = V'_{2,2} \cdot \hat{f}^{s_c z_c}, V_{2,3} = V'_{2,3} \cdot (\hat{f}^{-\phi_2})^{s_c z_c}.$$

Next, it verifies that  $\prod_{i=1}^3 e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^3 e(W_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$ . If this equation holds, then it outputs 1. Otherwise, it outputs 0.

If the semi-functional verification algorithm is used to verify a semi-functional signature, then an additional random element  $e(f, \hat{f})^{s_k s_c (z_k - z_c)}$  is left in the left part of the above verification equation. If  $z_k = z_c$ , then the semi-functional verification algorithm succeeds. In this case, we say that the signature is *nominally* semi-functional.

The security proof uses a sequence of games  $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$ : The first game  $\mathbf{G}_0$  will be the original security game and the last game  $\mathbf{G}_3$  will be a game such that an adversary  $\mathcal{A}$  has no advantage. Formally, the hybrid games are defined as follows:

**Game  $\mathbf{G}_0$ .** This game is the original security game. In this game, the signatures that are given to  $\mathcal{A}$  are normal and the challenger use the normal verification algorithm **PKS.Verify** to check the validity of the forged signature of  $\mathcal{A}$ . Note that  $\mathcal{A}$  can forge a normal signature or a semi-functional signature to win this game since normal or semi-functional signatures are always verified in the normal verification algorithm.

**Game  $\mathbf{G}_1$ .** This game is almost identical to  $\mathbf{G}_0$  except that the challenger use the semi-functional verification algorithm **PKS.VerifySF** to check the validity of the forged signature of  $\mathcal{A}$ . Note that  $\mathcal{A}$  should forge a normal signature to win this game since semi-functional signatures cannot be verified in the semi-functional verification algorithm.

**Game  $\mathbf{G}_2$ .** This game is the same as the  $\mathbf{G}_1$  except that the signatures that are given to  $\mathcal{A}$  will be semi-functional. At this moment, the signatures are

semi-functional and the challenger use the semi-functional verification algorithm **PKS.VerifySF** to check the validity of the forged signature. Suppose that  $\mathcal{A}$  makes at most  $q$  signature queries. For the security proof, we define a sequence of hybrid games  $\mathbf{G}_{1,0}, \dots, \mathbf{G}_{1,k}, \dots, \mathbf{G}_{1,q}$  where  $\mathbf{G}_{1,0} = \mathbf{G}_1$ . In  $\mathbf{G}_{1,k}$ , a normal signature is given to  $\mathcal{A}$  for all  $j$ -th signature queries such that  $j > k$  and a semi-functional signature is given to  $\mathcal{A}$  for all  $j$ -th signature queries such that  $j \leq k$ . It is obvious that  $\mathbf{G}_{1,q}$  is equal to  $\mathbf{G}_2$ .

**Game  $\mathbf{G}_3$ .** This final game differs from  $\mathbf{G}_2$  in that the challenger always rejects the forged signature of  $\mathcal{A}$  by replacing the element  $\Omega$  in the verification equation to a random element. Therefore, the advantage of this game is zero since  $\mathcal{A}$  cannot win this game.

To prove the security using the dual system encryption technique, we should show that it is hard for  $\mathcal{A}$  to forge a normal signature and a semi-functional signature. At first, from the indistinguishability between  $\mathbf{G}_0$  and  $\mathbf{G}_1$ , we obtain that  $\mathcal{A}$  can forge a normal signature with a non-negligible probability while he cannot forge a semi-functional signature when only normal signatures are given to  $\mathcal{A}$ . To finish the proof, we additionally should show that it is hard for  $\mathcal{A}$  to forge a normal signature. From the indistinguishability between  $\mathbf{G}_1$  and  $\mathbf{G}_2$ , we obtain that the probability of  $\mathcal{A}$  to forge a normal signature does not change when the signatures given to  $\mathcal{A}$  are changed from a normal type to a semi-functional type. Finally, from the indistinguishability between  $\mathbf{G}_2$  and  $\mathbf{G}_3$ , we obtain that it is hard for  $\mathcal{A}$  to forge a normal signature when only semi-functional signatures are given to the adversary. Therefore, we have the unforgeability of the adversary through the indistinguishability of hybrid games.  $\square$

**Lemma 1.** *If Assumption 1 holds, then no polynomial-time adversary can distinguish between  $\mathbf{G}_0$  and  $\mathbf{G}_1$  with non-negligible advantage.*

**Lemma 2.** *If Assumption 2 holds, then no polynomial-time adversary can distinguish between  $\mathbf{G}_1$  and  $\mathbf{G}_2$  with non-negligible advantage.*

**Lemma 3.** *If Assumption 3 holds, then no polynomial-time adversary can distinguish between  $\mathbf{G}_2$  and  $\mathbf{G}_3$  with non-negligible advantage.*

The proof of Lemma 1 is given in Appendix A and the proofs of other lemmas are given in the full version of this paper [14].

## 4 Sequential Aggregate Signature

In this section, we propose an efficient sequential aggregate signature (SAS) scheme with short public keys and prove its security without random oracles.

### 4.1 Definitions

The concept of SAS was introduced by Lysyanskaya et al. [18]. In SAS, all signers first generate public keys and private keys, and then publishes their public keys.



To generate a sequential aggregate signature, a signer may receive an aggregate-so-far from a previous signer, and creates a new aggregate signature by adding his signature to the aggregate-so-far in sequential order. After that, the signer may send the aggregate signature to a next signer. A verifier can check the validity of the aggregate signature by using the public keys of all signers in the aggregate signature. A SAS scheme is formally defined as follows:

**Definition 1 (Sequential Aggregate Signature).** *A sequential aggregate signature (SAS) scheme consists of four PPT algorithms **Setup**, **KeyGen**, **AggSign**, and **AggVerify**, which are defined as follows:*

**Setup**( $1^\lambda$ ). *The setup algorithm takes as input a security parameter  $1^\lambda$  and outputs public parameters  $PP$ .*

**KeyGen**( $PP$ ). *The key generation algorithm takes as input the public parameters  $PP$ , and outputs a public key  $PK$  and a private key  $SK$ .*

**AggSign**( $AS', \mathbf{M}, \mathbf{PK}, M, SK$ ). *The aggregate signing algorithm takes as input an aggregate-so-far  $AS'$  on messages  $\mathbf{M} = (M_1, \dots, M_l)$  under public keys  $\mathbf{PK} = (PK_1, \dots, PK_l)$ , a message  $M$ , and a private key  $SK$ , and outputs a new aggregate signature  $AS$ .*

**AggVerify**( $AS, \mathbf{M}, \mathbf{PK}$ ). *The aggregate verification algorithm takes as input an aggregate signature  $AS$  on messages  $\mathbf{M} = (M_1, \dots, M_l)$  under public keys  $\mathbf{PK} = (PK_1, \dots, PK_l)$ , and outputs either 1 or 0 depending on the validity of the sequential aggregate signature.*

*The correctness requirement is that for each  $PP$  output by **Setup**, for all  $(PK, SK)$  output by **KeyGen**, any  $M$ , we have that  $\mathbf{AggVerify}(\mathbf{AggSign}(AS', \mathbf{M}', \mathbf{PK}', M, SK), \mathbf{M}' || M, \mathbf{PK}' || PK) = 1$  where  $AS'$  is a valid aggregate-so-far signature on messages  $\mathbf{M}'$  under public keys  $\mathbf{PK}'$ .*

The security model of SAS was defined by Lysyanskaya et al. [18], but we use the security model of Lu et al. [17] that requires for an adversary to register key-pair of other signers except the target signer. The security model of SAS is formally defined as follows:

**Definition 2 (Security).** *The security notion of existential unforgeability under a chosen message attack is defined in terms of the following experiment between a challenger  $\mathcal{C}$  and a PPT adversary  $\mathcal{A}$ :*

1. **Setup:**  $\mathcal{C}$  first initializes a certification list  $CL$  as empty. Next, it runs **Setup** to obtain public parameters  $PP$  and **KeyGen** to obtain a key pair  $(PK, SK)$ , and gives  $PK$  to  $\mathcal{A}$ .
2. **Certification Query:**  $\mathcal{A}$  adaptively requests the certification of a public key by providing a key pair  $(PK, SK)$ . Then  $\mathcal{C}$  adds the key pair  $(PK, SK)$  to  $CL$  if the key pair is a valid one.
3. **Signature Query:**  $\mathcal{A}$  adaptively requests a sequential aggregate signature (by providing an aggregate-so-far  $AS'$  on messages  $\mathbf{M}'$  under public keys  $\mathbf{PK}'$ ), on a message  $M$  to sign under the challenge public key  $PK$ , and receives a sequential aggregate signature  $AS$ .

4. **Output:** Finally (after a sequence of the above queries),  $\mathcal{A}$  outputs a forged sequential aggregate signature  $AS^*$  on messages  $\mathbf{M}^*$  under public keys  $\mathbf{PK}^*$ .  $\mathcal{C}$  outputs 1 if the forged signature satisfies the following three conditions, or outputs 0 otherwise: 1)  $\mathbf{AggVerify}(AS^*, \mathbf{M}^*, \mathbf{PK}^*) = 1$ , 2) The challenge public key  $PK$  must exist in  $\mathbf{PK}^*$  and each public key in  $\mathbf{PK}^*$  except the challenge public key must be in  $CL$ , and 3) The corresponding message  $M$  in  $\mathbf{M}^*$  of the challenge public key  $PK$  must not have been queried by  $\mathcal{A}$  to the sequential aggregate signing oracle.

The advantage of  $\mathcal{A}$  is defined as  $\mathbf{Adv}_A^{SAS} = \Pr[C = 1]$  where the probability is taken over all the randomness of the experiment. A SAS scheme is existentially unforgeable under a chosen message attack if all PPT adversaries have at most a negligible advantage in the above experiment.

## 4.2 Construction

To construct a SAS scheme from a PKS scheme, the PKS scheme should support multi-users by sharing some elements among all signers and the randomness of signatures should be sequentially aggregated to a single value. We can employ the randomness reuse method of Lu et al. [17] to aggregate the randomness of signatures. To apply the randomness reuse method, we should re-randomize the aggregate signature to prevent a forgery attack. Thus we build on the PKS scheme of the previous section that supports multi-users and public re-randomization to construct a SAS scheme.

The SAS scheme in prime order bilinear groups is described as follows:

**SAS.Setup**( $1^\lambda$ ): This algorithm first generates the asymmetric bilinear groups  $\mathbb{G}, \hat{\mathbb{G}}$  of prime order  $p$  of bit size  $\Theta(\lambda)$ . It chooses random elements  $g, w \in \mathbb{G}$  and  $\hat{g} \in \hat{\mathbb{G}}$ . Next, it selects random exponents  $\nu, \phi_1, \phi_2 \in \mathbb{Z}_p$  and sets  $\tau = \phi_1 + \nu\phi_2$ ,  $w_1 = w^{\phi_1}$ ,  $w_2 = w^{\phi_2}$ . It publishes public parameters by selecting a random value  $c_g \in \mathbb{Z}_p$  as

$$PP = ( gw_1^{c_g}, w_2^{c_g}, w^{c_g}, w_1, w_2, w, \hat{g}, \hat{g}^\nu, \hat{g}^{-\tau}, \Lambda = e(g, \hat{g}) ).$$

**SAS.KeyGen**( $PP$ ): This algorithm takes as input the public parameters  $PP$ . It selects random exponents  $\alpha, x, y \in \mathbb{Z}_p$  and sets  $\hat{u} = \hat{g}^x$ ,  $\hat{h} = \hat{g}^y$ . It outputs a private key  $SK = (\alpha, x, y)$  and a public key by selecting random values  $c'_u, c'_h \in \mathbb{Z}_p$  as

$$\begin{aligned} PK = ( & uw_1^{c_u} = (gw_1^{c_g})^x w_1^{c'_u}, w_2^{c_u} = (w_2^{c_g})^x w_2^{c'_u}, w^{c_u} = (w^{c_g})^x w^{c'_u}, \\ & hw_1^{c_h} = (gw_1^{c_g})^y w_1^{c'_u}, w_2^{c_h} = (w_2^{c_g})^y w_2^{c'_u}, w^{c_h} = (w^{c_g})^y w^{c'_u}, \\ & \hat{u}, \hat{u}^\nu = (\hat{g}^\nu)^x, \hat{u}^{-\tau} = (\hat{g}^{-\tau})^x, \hat{h}, \hat{h}^\nu = (\hat{g}^\nu)^y, \hat{h}^{-\tau} = (\hat{g}^{-\tau})^y, \Omega = \Lambda^\alpha ). \end{aligned}$$

**SAS.AggSign**( $AS', \mathbf{M}', \mathbf{PK}', M, SK$ ): This algorithm takes as input an aggregate-so-far  $AS' = (S'_{1,1}, \dots, S'_{2,3})$  on messages  $\mathbf{M}' = (M_1, \dots, M_{l-1})$  under public keys  $\mathbf{PK}' = (PK_1, \dots, PK_{l-1})$  where  $PK_i = (u_i w_1^{c_{u,i}}, \dots, \Omega_i)$ , a message

$M \in \mathbb{Z}_p$ , a private key  $SK = (\alpha, x, y)$  with  $PK = (uw_1^{c_u}, \dots, \Omega)$  and  $PP$ . It first checks the validity of  $AS'$  by calling **SAS.AggVerify**( $AS', M', PK'$ ). If  $AS'$  is not valid, then it halts. If the public key  $PK$  of  $SK$  does already exist in  $PK'$ , then it halts. Next, it selects random exponents  $r, c_1, c_2 \in \mathbb{Z}_p$  and outputs an aggregate signature as

$$\begin{aligned} AS &= (S_{1,1} = S'_{1,1} (gw_1^{c_g})^\alpha (S'_{2,1})^{xM+y} \cdot \prod_{i=1}^{l-1} ((u_i w_1^{c_{u,i}})^{M_i} (h_i w_1^{c_{h,i}}))^r ((w_1^{c_u})^M (h w_1^{c_h}))^r w_1^{c_1}, \\ S_{1,2} &= S'_{1,2} (w_2^{c_g})^\alpha (S'_{2,2})^{xM+y} \cdot \prod_{i=1}^{l-1} ((w_2^{c_{u,i}})^{M_i} (w_2^{c_{h,i}}))^r ((w_2^{c_u})^M w_2^{c_h})^r w_2^{c_1}, \\ S_{1,3} &= S'_{1,3} (w^{c_g})^\alpha (S'_{2,3})^{xM+y} \cdot \prod_{i=1}^{l-1} ((w^{c_{u,i}})^{M_i} (w^{c_{h,i}}))^r ((w^{c_u})^M w^{c_h})^r w^{c_1}, \\ S_{2,1} &= S'_{2,1} \cdot (gw_1^{c_g})^r w_1^{c_2}, \quad S_{2,2} = S'_{2,2} \cdot (w_2^{c_g})^r w_2^{c_2}, \quad S_{2,3} = S'_{2,3} \cdot (w^{c_g})^r w^{c_2}. \end{aligned}$$

**SAS.AggVerify**( $AS, M, PK$ ): This algorithm takes as input a sequential aggregate signature  $AS$  on messages  $M = (M_1, \dots, M_l)$  under public keys  $PK = (PK_1, \dots, PK_l)$  where  $PK_i = (u_i w_1^{c_{u,i}}, \dots, \Omega_i)$ . It first checks that any public key does not appear twice in  $PK$  and that any public key in  $PK$  has been certified. If these checks fail, then it outputs 0. If  $l = 0$ , then it outputs 1 if  $S_1 = S_2 = 1$ , 0 otherwise. It chooses a random exponent  $t \in \mathbb{Z}_p$  and computes verification components as

$$\begin{aligned} C_{1,1} &= \hat{g}^t, \quad C_{1,2} = (\hat{g}^\nu)^t, \quad C_{1,3} = (\hat{g}^{-\tau})^t, \\ C_{2,1} &= \prod_{i=1}^l (\hat{u}_i^{M_i} \hat{h}_i)^t, \quad C_{2,2} = \prod_{i=1}^l ((\hat{u}_i^\nu)^{M_i} \hat{h}_i^\nu)^t, \quad C_{2,3} = \prod_{i=1}^l ((\hat{u}_i^{-\tau})^{M_i} \hat{h}_i^{-\tau})^t. \end{aligned}$$

Next, it verifies that  $\prod_{i=1}^3 e(S_{1,i}, C_{1,i}) \cdot \prod_{i=1}^3 e(S_{2,i}, C_{2,i})^{-1} \stackrel{?}{=} \prod_{i=1}^l \Omega_i^t$ . If this equation holds, then it outputs 1. Otherwise, it outputs 0.

### 4.3 Security Analysis

**Theorem 2.** *The above SAS scheme is existentially unforgeable under a chosen message attack if the PKS scheme is existentially unforgeable under a chosen message attack.*

*Proof.* Suppose there exists an adversary  $\mathcal{A}$  that forges the above SAS scheme with non-negligible advantage  $\epsilon$ . A simulator  $\mathcal{B}$  that forges the PKS scheme is first given: a challenge public key  $PK_{PKS} = (gw_1^{c_g}, w_2^{c_g}, w^{c_g}, uw_1^{c_u}, \dots, w^{c_h}, w_1, w_2, w, \hat{g}, \hat{g}^\nu, \hat{g}^{-\tau}, \hat{u}, \dots, \hat{h}^{-\tau}, \Lambda, \Omega)$ . Then  $\mathcal{B}$  that interacts with  $\mathcal{A}$  is described as follows:

**Setup:**  $\mathcal{B}$  first constructs  $PP = (gw_1^{c_g}, w_2^{c_g}, w^{c_g}, w_1, w_2, w, \hat{g}, \hat{g}^\nu, \hat{g}^{-\tau}, \Lambda)$  and  $PK^* = (uw_1^{c_u}, \dots, w^{c_h}, \hat{u}, \dots, \hat{h}^{-\tau}, \Omega)$  from  $PK_{PKS}$ . Next, it initializes a certification list  $CL$  as an empty one and gives  $PP$  and  $PK^*$  to  $\mathcal{A}$ .

**Queries:**  $\mathcal{A}$  may adaptively requests certification queries or sequential aggregate signature queries. If  $\mathcal{A}$  requests the certification of a public key by providing

a public key  $PK_i = (u_i w_1^{c_{u,i}}, \dots, \Omega_i)$  and its private key  $SK_i = (\alpha_i, x_i, y_i)$ , then  $\mathcal{B}$  checks the private key and adds the key pair  $(PK_i, SK_i)$  to  $CL$ . If  $\mathcal{A}$  requests a sequential aggregate signature by providing an aggregate-so-far  $AS'$  on messages  $\mathbf{M}' = (M_1, \dots, M_{l-1})$  under public keys  $\mathbf{PK}' = (PK_1, \dots, PK_{l-1})$ , and a message  $M$  to sign under the challenge private key of  $PK^*$ , then  $\mathcal{B}$  proceeds the aggregate signature query as follows:

1. It first checks that the signature  $AS'$  is valid and that each public key in  $\mathbf{PK}'$  exits in  $CL$ .
2. It queries its signing oracle that simulates **PKS.Sign** on the message  $M$  for the challenge public key  $PK^*$  and obtains a signature  $\sigma$ .
3. For each  $1 \leq i \leq l-1$ , it constructs an aggregate signature on message  $M_i$  using **SAS.AggSign** since it knows the private key that corresponds to  $PK_i$ . The result signature is an aggregate signature for messages  $\mathbf{M}' || M$  under public keys  $\mathbf{PK}' || PK^*$  since this scheme does not check the order of aggregation. It gives the result signature  $AS$  to  $\mathcal{A}$ .

**Output:** Finally,  $\mathcal{A}$  outputs a forged aggregate signature  $AS^* = (S_{1,1}^*, \dots, S_{2,3}^*)$  on messages  $\mathbf{M}^* = (M_1, \dots, M_l)$  under public keys  $\mathbf{PK}^* = (PK_1, \dots, PK_l)$  for some  $l$ . Without loss of generality, we assume that  $PK_1 = PK^*$ .  $\mathcal{B}$  proceeds as follows:

1.  $\mathcal{B}$  first checks the validity of  $AS^*$  by using **SAS.AggVerify**. Additionally, the forged signature should not be trivial: the challenge public key  $PK^*$  must be in  $\mathbf{PK}^*$ , and the message  $M_1$  must not be queried by  $\mathcal{A}$  to the signature query oracle.
2. For each  $2 \leq i \leq l$ , it parses  $PK_i = (u_i w_1^{c_{u,i}}, \dots, \Omega_i)$  from  $\mathbf{PK}^*$ , and it retrieves the private key  $SK_i = (\alpha_i, x_i, y_i)$  of  $PK_i$  from  $CL$ . It then computes

$$W_{1,1} = S_{1,1}^* \prod_{i=2}^l (g^{\alpha_j (S_{2,1}^*)^{x_i M_i + y_i}})^{-1}, \quad W_{1,2} = S_{1,2}^* \prod_{i=2}^l ((S_{2,2}^*)^{x_i M_i + y_i})^{-1},$$

$$W_{1,3} = S_{1,3}^* \prod_{i=2}^l ((S_{2,3}^*)^{x_i M_i + y_i})^{-1}, \quad W_{2,1} = S_{2,1}^*, \quad W_{2,2} = S_{2,2}^*, \quad W_{2,3} = S_{2,3}^*.$$

3. It outputs  $\sigma = (W_{1,1}, \dots, W_{2,3})$  as a non-trivial forgery of the PKS scheme since it did not make a signing query on  $M_1$ .

The public parameters and the public key are correctly distributed, and the sequential aggregate signatures are also correctly distributed since this scheme does not check the order of aggregation. The result signature  $\sigma = (W_{1,1}, \dots, W_{2,3})$  of the simulator is a valid PKS signature on the message  $M_1$  under the public key  $PK^*$  since it satisfies the following equation:

$$\prod_{i=1}^3 e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^3 e(W_{2,i}, V_{2,i})^{-1}$$

$$= e(S_{1,1}^*, \hat{g}^t) \cdot e(S_{1,2}^*, \hat{g}^{\nu t}) \cdot e(S_{1,4}^*, \hat{g}^{-\tau t}) \cdot e\left(\prod_{i=2}^l g^{\alpha_i}, \hat{g}^t\right)^{-1}.$$

$$\begin{aligned}
& e(S_{2,1}^*, \prod_{i=2}^l (\hat{u}_i^{M_i} \hat{h}_i)^t)^{-1} \cdot e(S_{2,2}^*, \prod_{i=2}^l (\hat{u}_i^{M_i} \hat{h}_i)^{\nu t})^{-1} \cdot e(S_{2,3}^*, \prod_{i=2}^l (\hat{u}_i^{M_i} \hat{h}_i)^{-\tau t})^{-1} \\
& e(S_{2,1}^*, (\hat{u}^{M_1} \hat{h})^t)^{-1} \cdot e(S_{2,2}^*, (\hat{u}^{M_1} \hat{h})^{\nu t})^{-1} \cdot e(S_{2,3}^*, (\hat{u}^{M_1} \hat{h})^{-\tau t})^{-1} \\
& = e(S_{1,1}^*, C_{1,1}) \cdot e(S_{1,2}^*, C_{1,2}) \cdot e(S_{1,3}^*, C_{1,3}) \cdot e(\prod_{i=2}^l g^{\alpha_i}, \hat{g}^t)^{-1} \\
& e(S_{2,1}^*, \prod_{i=1}^l (\hat{u}_i^{M_i} \hat{h}_i)^t)^{-1} \cdot e(S_{2,2}^*, \prod_{i=1}^l (\hat{u}_i^{M_i} \hat{h}_i)^{\nu t})^{-1} \cdot e(S_{2,3}^*, \prod_{i=1}^l (\hat{u}_i^{M_i} \hat{h}_i)^{-\tau t})^{-1} \\
& = \prod_{i=1}^3 e(S_{1,i}^*, C_{1,i}) \cdot \prod_{i=1}^3 e(S_{2,i}^*, C_{2,i})^{-1} \cdot e(\prod_{i=2}^l g^{\alpha_i}, \hat{g}^t)^{-1} = \prod_{i=1}^l \Omega_i^t \cdot \prod_{i=2}^l \Omega_i^{-t} = \Omega_1^t
\end{aligned}$$

where  $\delta_i = x_i M_i + y_i$  and  $\tilde{s}_2 = \sum_{i=2}^l (x_i M_i + y_i) s_1 + s_2$ . This completes our proof.

#### 4.4 Discussions

**Multi-signature.** A MS scheme can be easily constructed from our SAS scheme by moving some group elements in the public key to the public parameters. This scheme is also secure without random oracles under static assumptions and the signature size of this scheme is shorter than that of Lee et al.'s MS scheme [15].

## 5 Conclusion

In this paper, we improved the SAS scheme of Lee et al. [15] by reducing the size of aggregate signatures and similarly proved its security without random oracles under static assumptions. To reduce the size of signatures, we first devised a PKS scheme that supports multi-users and public re-randomization and proved its security using the dual system encryption technique. The proposed SAS scheme of this paper trades off signature size against public-key size compared with the scheme of Lee et al. since the signature size of our scheme decreases by two group elements but the public-key size increases by two group elements (but signatures are many and a public key is published once). Our techniques include lifting and randomization of verification parameters used in the previous scheme.

## References

1. Ahn, J.H., Green, M., Hohenberger, S.: Synchronized aggregate signatures: new definitions, constructions and applications. In: ACM Conference on Computer and Communications Security, pp. 473–484 (2010)
2. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 411–422. Springer, Heidelberg (2007)

3. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2002)
4. Boldyreva, A., Gentry, C., O’Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. Cryptology ePrint Archive, Report 2007/438 (2010), <http://eprint.iacr.org/2007/438>
5. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
6. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
7. Brogle, K., Goldberg, S., Reyzin, L.: Sequential aggregate signatures with lazy verification from trapdoor permutations. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 644–662. Springer, Heidelberg (2012)
8. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM 51(4), 557–594 (2004)
9. Fischlin, M., Lehmann, A., Schröder, D.: History-free sequential aggregate signatures. In: Visconti, I., De Prisco, R. (eds.) SCN 2012. LNCS, vol. 7485, pp. 113–130. Springer, Heidelberg (2012)
10. Gentry, C., Ramzan, Z.: Identity-based aggregate signatures. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 257–273. Springer, Heidelberg (2006)
11. Gerbush, M., Lewko, A., O’Neill, A., Waters, B.: Dual form signatures: An approach for proving security from static assumptions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 25–42. Springer, Heidelberg (2012)
12. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC Research & Development (71), 1–8 (1983)
13. Lee, K., Lee, D.H., Yung, M.: Aggregating cl-signatures revisited: Extended functionality and better efficiency. Cryptology ePrint Archive, Report 2012/562 (2012), <http://eprint.iacr.org/2012/562>
14. Lee, K., Lee, D.H., Yung, M.: Sequential aggregate signatures made shorter. Cryptology ePrint Archive (2013), <http://eprint.iacr.org/>
15. Lee, K., Lee, D.H., Yung, M.: Sequential aggregate signatures with short public keys: Design, analysis and implementation studies. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 423–442. Springer, Heidelberg (2013)
16. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010)
17. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)
18. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004)
19. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: extended abstract. In: Reiter, M.K., Samarati, P. (eds.) ACM Conference on Computer and Communications Security, pp. 245–254. ACM (2001)

20. Neven, G.: Efficient sequential aggregate signed data. In: Smart, N.P. (ed.) EU-ROCRYPT 2008. LNCS, vol. 4965, pp. 52–69. Springer, Heidelberg (2008)
21. Schröder, D.: How to aggregate the cl signature scheme. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 298–314. Springer, Heidelberg (2011)
22. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)

## A Security Proofs of Lemmas

### A.1 The Proof of Lemma 1

The proof of this lemma is almost similar to the proof of Lemma 1 in [16] except that the public key is generated differently and the proof is employed in the PKS setting. Suppose there exists an adversary  $\mathcal{A}$  that distinguishes between  $\mathbf{G}_0$  and  $\mathbf{G}_1$  with non-negligible advantage. A simulator  $\mathcal{B}_1$  that solves Assumption 1 using  $\mathcal{A}$  is given: a challenge tuple  $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), k, k^b, \hat{k}, \hat{k}^a, \hat{k}^b, \hat{k}^{ab^2}, \hat{k}^{b^2}, \hat{k}^{b^3}, \hat{k}^c, \hat{k}^{ac}, \hat{k}^{bc}, \hat{k}^{b^2c}, \hat{k}^{b^3c})$  and  $T$  where  $T = T_0 = \hat{k}^{ab^2c}$  or  $T = T_1 = \hat{k}^{ab^2c+d}$ . Then  $\mathcal{B}_1$  that interacts with  $\mathcal{A}$  is described as follows:  $\mathcal{B}_1$  first chooses random exponents  $\phi_2, A, B, \alpha \in \mathbb{Z}_p$ , random values  $y_g, y_u, y_h, y_w \in \mathbb{Z}_p$ . It computes  $w_1 = w^{\phi_1} = (k^b)^{y_w}, w_2 = w^{\phi_2} = k^{y_w \phi_2}, w = k^{y_w}$  by implicitly setting  $\phi_1 = b$ . It implicitly sets  $c_g = -b/y_w + c'_g, c_u = -bA/y_w + c'_u, c_h = -bB/y_w + c'_h, \nu = a, \tau = b + a\phi_2$  and publishes a public key by selecting random values  $c'_g, c'_u, c'_h \in \mathbb{Z}_p$  as

$$\begin{aligned}
gw_1^{c_g} &= k^{y_g} w_1^{c'_g}, \quad w_2^{c_g} = (k^b)^{-b_2} w_2^{c'_g}, \quad w^{c_g} = (k^b)^{-1} w^{c'_g}, \\
uw_1^{c_u} &= k^{y_u} w_1^{c'_u}, \quad w_2^{c_u} = (k^b)^{-b_2 A} w_2^{c'_u}, \quad w^{c_u} = (k^b)^{-A} w^{c'_u}, \\
hw_1^{c_h} &= k^{y_h} w_1^{c'_h}, \quad w_2^{c_h} = (k^b)^{-b_2 B} w_2^{c'_h}, \quad w^{c_h} = (k^b)^{-B} w^{c'_h}, \quad w_1, w_2, w, \\
\hat{g} &= \hat{k}^{b^2} \hat{k}^{y_g}, \quad \hat{g}^\nu = \hat{k}^{ab^2} (\hat{k}^a)^{y_g}, \quad \hat{g}^{-\tau} = (\hat{k}^{b^3} (\hat{k}^b)^{y_g} (\hat{k}^{ab^2})^{b_2} (\hat{k}^a)^{y_g b_2})^{-1}, \\
\hat{u} &= (\hat{k}^{b^2})^A \hat{k}^{y_u}, \quad \hat{u}^\nu = (\hat{k}^{ab^2})^A (\hat{k}^a)^{y_u}, \quad \hat{u}^{-\tau} = ((\hat{k}^{b^3})^A (\hat{k}^b)^{y_u} (\hat{k}^{ab^2})^{Ab_2} (\hat{k}^a)^{y_u b_2})^{-1}, \\
\hat{h} &= (\hat{k}^{b^2})^B \hat{k}^{y_h}, \quad \hat{h}^\nu = (\hat{k}^{ab^2})^B (\hat{k}^a)^{y_h}, \quad \hat{h}^{-\tau} = ((\hat{k}^{b^3})^B (\hat{k}^b)^{y_h} (\hat{k}^{ab^2})^{Bb_2} (\hat{k}^a)^{y_h b_2})^{-1}, \\
\Lambda &= e(k^{b^3}, \hat{k}^b) \cdot e(k^{b^2}, \hat{k})^{2y_g} \cdot e(k, \hat{k})^{y_g^2}, \quad \Omega = \Lambda^\alpha.
\end{aligned}$$

It implicitly sets  $g = k^{b^2} k^{y_g}, u = (k^{b^2})^A k^{y_u}, h = (k^{b^2})^B k^{y_h}$ , but it cannot create these elements since  $k^{b^2}$  is not given. Additionally, it sets  $f = k, \hat{f} = \hat{k}$  for the semi-functional signature and verification.  $\mathcal{A}$  adaptively requests a signature for a message  $M$ . To response this sign query,  $\mathcal{B}_1$  first selects random exponents  $r, c'_1, c'_2 \in \mathbb{Z}_p$ . It implicitly sets  $c_1 = -b(\alpha + (AM + B)r)/y_w + c'_1, c_2 = -br_1/y_w + c'_2$  and creates a normal signature as

$$\begin{aligned}
W_{1,1} &= k^{y_g \alpha + (y_u M + y_h) r} (w_1)^{c'_1}, \quad W_{1,2} = (W_{1,3})^{\phi_2}, \quad W_{1,3} = (k^b)^{-(\alpha + (AM + B)r)} w^{c'_1}, \\
W_{2,1} &= k^{y_g r} (w_1)^{c'_2}, \quad W_{2,2} = (W_{2,3})^{\phi_2}, \quad W_{2,3} = (k^b)^{-r} w^{c'_2}.
\end{aligned}$$

Finally,  $\mathcal{A}$  outputs a forged signature  $\sigma^* = (W_{1,1}^*, \dots, W_{2,3}^*)$  on a message  $M^*$  from  $\mathcal{A}$ . To verify the forged signature,  $\mathcal{B}_1$  first chooses a random exponent  $t \in \mathbb{Z}_p$  and computes verification components by implicitly setting  $t = c$  as

$$\begin{aligned} V_{1,1} &= \hat{k}^{b^2c} (\hat{k}^c)^{y_g}, \quad V_{1,2} = T (\hat{k}^{ac})^{y_g}, \quad V_{1,3} = ((\hat{k}^{b^3c}) (\hat{k}^{bc})^{y_g} (T)^{\phi_2} (\hat{k}^{ac})^{y_g \phi_2})^{-1}, \\ V_{2,1} &= (\hat{k}^{b^2c})^{AM^*+B} (\hat{k}^c)^{y_u M^* + y_h}, \quad V_{2,2} = (T)^{AM^*+B} (\hat{k}^{ac})^{y_u M^* + y_h}, \\ V_{2,3} &= ((\hat{k}^{b^3c})^{AM^*+B} (\hat{k}^{bc})^{y_u M^* + y_h} (T)^{\phi_2(AM^*+B)} (\hat{k}^{ac})^{\phi_2(y_u M^* + y_h)})^{-1}. \end{aligned}$$

Next, it verifies that  $\prod_{i=1}^3 e(W_{1,i}^*, V_{1,i}) \cdot \prod_{i=1}^3 e(W_{2,i}^*, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$ . If this equation holds, then it outputs 0. Otherwise, it outputs 1.

To finish this proof, we show that the distribution of the simulation is correct. We first show that the distribution using  $D, T_0 = \hat{k}^{ab^2c}$  is the same as  $\mathbf{G}_0$ . The public key is correctly distributed as

$$\begin{aligned} gw_1^{c_g} &= (k^{b^2} k^{y_g}) (k^{by_w})^{-b/y_w + c'_g} = k^{y_g} w_1^{c'_g}, \\ uw_1^{c_u} &= (k^{b^2A} k^{y_u}) (k^{by_w})^{-bA/y_w + c'_u} = k^{y_u} w_1^{c'_u}, \\ hw_1^{c_h} &= (k^{b^2B} k^{y_h}) (k^{by_w})^{-bB/y_w + c'_h} = k^{y_h} w_1^{c'_h}. \end{aligned}$$

The simulator cannot create  $g, u, h$  since  $k^{b^2}$  is not given in the assumption, but it can create  $gw_1^{c_g}, uw_1^{c_u}, hw_1^{c_h}$  since  $c_g, c_u, c_h$  can be used to cancel out  $k^{b^2}$ . The signature and the verification components are also correctly distributed since these are similar to the simulation in [16]. We next show that the distribution of the simulation using  $D, T_1 = \hat{k}^{ab^2c+d}$  is the same as  $\mathbf{G}_1$ . We only consider the distribution of the verification components since  $T$  is only used in the verification components. The difference between  $T_0$  and  $T_1$  is that  $T_1$  additionally has  $\hat{k}^d$ . Thus  $V_{1,2}, V_{1,3}, V_{2,2}, V_{2,3}$  that have  $T$  in the simulation additionally have  $\hat{k}^d, (\hat{k}^d)^{\phi_2}, (\hat{k}^d)^{AM^*+B}, (\hat{k}^d)^{\phi_2(AM^*+B)}$  respectively. If we implicitly set  $s_c = d, z_c = AM^* + B$ , then the verification components of the forged signature are semi-functional since  $A$  and  $B$  are information-theoretically hidden to the adversary.