Sequential Aggregate Signatures Made Shorter

Kwangsu Lee* Dong Hoon Lee† Moti Yung‡

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, Lee, and Yung (PKC 2013) 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).

Keywords: Public-key signature, Aggregate signature, Sequential Aggregation, Multi-signature, Bilinear map.

*Korea University, Korea. Email: guspin@korea.ac.kr. This work was partially done while the author was at Columbia University. 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).

†Korea University, Korea. Email: donghlee@korea.ac.kr. Supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0029121).

‡Google Inc. and Columbia University, USA. Email: moti@cs.columbia.edu.

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 [1,2,4,6,11,12,15,16,18,24] and trapdoor permutations [7, 19, 21]. However, the security of many aggregate signature schemes was proven in the random oracle model. The random oracle model was very 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 [19]) [15, 16, 18, 24] (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 [11]) [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. [18], but the public-key size of this scheme is too large since the scheme is based on the PKS scheme of Waters [25]. 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. [18], 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 [24], but it is only secure under the interactive LRSW assumption. Recently, Lee et al. [16] proposed another SAS scheme with short public keys based on the identity-based encryption (IBE) scheme of Lewko and Waters [17] 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. [16] 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 [17] (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. [16] 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. [19] introduced the concept of sequential aggregate signature (SAS) and proposed a SAS scheme in trapdoor permutations. Lu et al. [18] 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 [15, 16, 24]. Recently, SAS schemes that do not require a verifier to check the validity of the previous signature were proposed [7, 10]. 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. [12] 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 [11] 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 an synchronized aggregate signature scheme and proved its security without random oracles. Recently, Lee et al. [15] 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 [14]. MS is a special type of aggregate signatures where all signers generate signatures for the same message. Micali et al. [20] 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. [18] proposed the first MS scheme that is secure without random oracles by modifying their SAS scheme. Recently, Lee et al. [16] 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 $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ 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 [17], 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 λ. 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 $\mathbf{Adv}^{A1}_{\mathcal{B}}(\lambda) = \left| \Pr[\mathcal{B}(D, T_0) = 0] - \Pr[\mathcal{B}(D, T_1) = 0] \right|$ 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

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

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 $\mathbf{Adv}^{A2}_{\mathcal{B}}(\lambda) = \left| \Pr[\mathcal{B}(D, T_0) = 0] - \Pr[\mathcal{B}(D, T_1) = 0] \right|$ 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 $\mathbf{Adv}^{A3}_{\mathcal{B}}(\lambda) = \left| \Pr[\mathcal{B}(D, T_0) = 0] - \Pr[\mathcal{B}(D, T_1) = 0] \right|$ 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 Definitions

The concept of PKS was introduced by Diffie and Hellman [9], and the first PKS scheme was proposed by Rivest et al. [23] based on trapdoor permutations. In PKS, a signer first generates a public key and a private key, and he keeps the private key himself and publishes the public key. The signer generates a signature on a message by using the private key. A verifier can check the validity of the signature of the signer on the message by using the signer's public key. A PKS scheme is formally defined as follows:

Definition 3.1 (Public-Key Signature)**.** *A public key signature (PKS) scheme consists of three PPT algorithms* ***KeyGen****,* ***Sign****, and* ***Verify****, which are defined as follows:*

KeyGen(1^λ)*. The key generation algorithm takes as input the security parameters* 1^λ*, and outputs a public key PK and a private key SK.*

Sign(M, SK)*. The signing algorithm takes as input a message M and a private key SK, and outputs a signature* σ*.*

Verify(σ, M, PK)*. The verification algorithm takes as input a signature* σ*, a message M, and a public key PK, and outputs either* 1 *or* 0 *depending on the validity of the signature.*

The correctness requirement is that for any (PK, SK) *output by* ***KeyGen*** *and any* $M \in \mathcal{M}$*, we have that* ***Verify***$($***Sign***$(M, SK), M, PK) = 1$*. We can relax this notion to require that the verification is correct with overwhelming probability over all the randomness of the experiment.*

The security model of PKS is defined as existential unforgeability under a chosen message attack (EUF-CMA), and this was formally defined by Goldwasser et al. [13]. In this security model, an adversary adaptively can request a polynomial number of signatures on messages through the signing oracle, and he finally outputs a forged signature on a message M^*. If the message M^* was not queried to the signing oracle and the forged signature is valid, then the adversary wins this game. The security of PKS is formally defined as follows:

Definition 3.2 (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 generates a key pair* (PK, SK) *by running* ***KeyGen****, and gives PK to* $\mathcal{A}$*.*
2. ***Signature Query****: Then* $\mathcal{A}$*, adaptively and polynomially many times, requests a signature query on a message M under the challenge public key PK, and receives a signature* σ *generated by running* ***Sign****.*
3. ***Output****: Finally,* $\mathcal{A}$ *outputs a forged signature* σ^* *on a message* M^**.* $\mathcal{C}$ *then outputs* 1 *if the forged signature satisfies the following two conditions, or outputs* 0 *otherwise: 1)* ***Verify***$(\sigma^*, M^*, PK) = 1$ *and 2)* M^* *was not queried by* $\mathcal{A}$ *to the signing oracle.*

The advantage of $\mathcal{A}$ *is defined as* $\mathbf{Adv}_{\mathcal{A}}^{PKS} = \Pr[\mathcal{C} = 1]$ *where the probability is taken over all the randomness of the experiment. A PKS scheme is existentially unforgeable under a chosen message attack if all PPT adversaries have at most a negligible advantage in the above experiment (for large enough security parameter).*

3.2 Design Principle

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 [17] 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. [16] 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 [17], 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 $w_2^{c_g}, w^{c_g}, w_2^{c_u}, w^{c_u}, w_2^{c_h}, w^{c_h}$ should be published.

3.3 Construction

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

PKS.KeyGen(1^λ): 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 $SK = (\alpha, g, u, h)$ and a public key by selecting random values $c_g, c_u, c_h \in \mathbb{Z}_p$ as

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

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

$$\sigma = \big(W_{1,1} = g^\alpha (u^M h)^r w_1^{c_1},\ W_{1,2} = w_2^{c_1},\ W_{1,3} = w^{c_1},$$
$$W_{2,1} = g^r w_1^{c_2},\ W_{2,2} = w_2^{c_2},\ W_{2,3} = w^{c_2} \big).$$

PKS.Rand(σ', M, PK): This algorithm takes as input a signature $\sigma' = (W'_{1,1}, \ldots, W'_{2,3})$, a message $M \in \mathbb{Z}_p$ and a public key PK. It selects random exponents $r, c_1, c_2 \in \mathbb{Z}_p$ and outputs a randomized signature as

$$\sigma = \big(W_{1,1} = W'_{1,1} \cdot ((uw_1^{c_u})^M (hw_1^{c_h}))^r w_1^{c_1},$$
$$W_{1,2} = W'_{1,2} \cdot ((w_2^{c_u})^M w_2^{c_h})^r w_2^{c_1},\ W_{1,3} = W'_{1,3} \cdot ((w^{c_u})^M w^{c_h})^r w^{c_1},$$
$$W_{2,1} = W'_{2,1} \cdot (gw_1^{c_g})^r w_1^{c_2},\ W_{2,2} = W'_{2,2} \cdot (w_2^{c_g})^r w_2^{c_2},\ W_{2,3} = W'_{2,3} \cdot (w^{c_g})^r w^{c_2} \big).$$

PKS.Verify(σ, M, PK): This algorithm takes as input a signature σ 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

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

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 randomized signature is also correctly distributed as

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

3.4 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 [17]. 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. [16] proved the security of their PKS scheme by using the dual system encryption technique, and Gerbush et al. [12] developed the dual form signature technique that is a variation of the dual system encryption technique to prove the security of theirs PKS schemes.

Theorem 3.3. *The above PKS scheme is existentially unforgeable under a chosen message attack if Assumptions 1, 2, and 3 hold. That is, for any PPT adversary $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ such that $\mathbf{Adv}_{\mathcal{A}}^{PKS}(\lambda) \leq \mathbf{Adv}_{\mathcal{B}_1}^{A1}(\lambda) + q\mathbf{Adv}_{\mathcal{B}_2}^{A2}(\lambda) + \mathbf{Adv}_{\mathcal{B}_3}^{A3}(\lambda)$ where q is the maximum number of signature queries of $\mathcal{A}$.*

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 $(W'_{1,1}, \ldots, W'_{2,3})$ 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

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

PKS.VerifySF. The semi-functional verification algorithm first creates a normal verification components using the public key. Let $(V'_{1,1}, \ldots, V'_{2,3})$ 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 $\mathbf{G}_{1,0},\ldots,\mathbf{G}_{1,k},\ldots,\mathbf{G}_{1,q}$ 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 Ω 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.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_j}$ be the advantage of $\mathcal{A}$ in $\mathbf{G}_j$ for $j=0,\ldots,3$. Let $\mathbf{Adv}_{\mathcal{A}}^{G_{1,k}}$ be the advantage of $\mathcal{A}$ in $\mathbf{G}_{1,k}$ for $k=0,\ldots,q$. It is clear that $\mathbf{Adv}_{\mathcal{A}}^{G_0}=\mathbf{Adv}_{\mathcal{A}}^{PKS}(\lambda)$, $\mathbf{Adv}_{\mathcal{A}}^{G_{1,0}}=\mathbf{Adv}_{\mathcal{A}}^{G_1}$, $\mathbf{Adv}_{\mathcal{A}}^{G_{1,q}}=\mathbf{Adv}_{\mathcal{A}}^{G_2}$, and $\mathbf{Adv}_{\mathcal{A}}^{G_3}=0$. From the following three lemmas, we prove that it is hard for $\mathcal{A}$ to distinguish $\mathbf{G}_{i-1}$ from $\mathbf{G}_i$ under the given assumptions. Therefore, we have that

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{PKS}(\lambda)&=\mathbf{Adv}_{\mathcal{A}}^{G_0}+\sum_{i=1}^{2}\left(\mathbf{Adv}_{\mathcal{A}}^{G_i}-\mathbf{Adv}_{\mathcal{A}}^{G_i}\right)-\mathbf{Adv}_{\mathcal{A}}^{G_3}\leq\sum_{i=1}^{3}\left|\mathbf{Adv}_{\mathcal{A}}^{G_{i-1}}-\mathbf{Adv}_{\mathcal{A}}^{G_i}\right|\\&=\mathbf{Adv}_{\mathcal{B}_1}^{A1}(\lambda)+\sum_{k=1}^{q}\mathbf{Adv}_{\mathcal{B}_2}^{A2}(\lambda)+\mathbf{Adv}_{\mathcal{B}_3}^{A3}(\lambda).\end{aligned}$$

This completes our proof. □

Lemma 3.4. *If Assumption 1 holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with non-negligible advantage. That is, for any adversary $\mathcal{A}$, there exists a PPT algorithm $\mathcal{B}_1$ such that $\left|\mathbf{Adv}_{\mathcal{A}}^{G_0}-\mathbf{Adv}_{\mathcal{A}}^{G_1}\right|=\mathbf{Adv}_{\mathcal{B}_1}^{A1}(\lambda)$.*

Proof. The proof of this lemma is almost similar to the proof of Lemma 1 in [17] 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

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

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

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

Finally, $\mathcal{A}$ outputs a forged signature $\sigma^* = (W^*_{1,1},\ldots,W^*_{2,3})$ 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

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

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

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

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 is correctly distributed as

$$\begin{aligned}
W_{1,1} &= g^{\alpha}(u^M h)^r w_1^{c_1} = (k^{b^2+y_g})^{\alpha}(k^{(b^2A+y_u)M}k^{b^2B+y_h})^r(k^{by_w})^{-b(\alpha+(AM+B)r)/y_w+c_1'} \\
&= k^{y_g\alpha+(y_uM+y_h)r}w_1^{c_1'}, \\
W_{2,1} &= g^r(w^{b_1})^{c_2} = (k^{b^2+y_g})^r(k^{by_w})^{-br/y_w+c_2'} = k^{y_g r}(w^{b_1})^{c_2'}.
\end{aligned}$$

It can create a normal signature since c_1, c_2 enable the cancellation of k^{b^2}, but it cannot create a semi-functional signature since k^a is not given. The verification components are correctly distributed as

$$\begin{aligned}
V_{1,1} &= \hat{g}^t = (\hat{k}^{b^2+y_g})^c = \hat{k}^{b^2c}(\hat{k}^c)^{y_g},\ V_{1,2} = (\hat{g}^{\nu})^t = \hat{k}^{(b^2+y_g)ac} = T_0(\hat{k}^{ac})^{y_g}, \\
V_{1,3} &= (\hat{g}^{-\tau})^t = (\hat{k}^{(b^2+y_g)(b+a\phi_2)c})^{-1} = ((\hat{k}^{b^3c})(\hat{k}^{bc})^{y_g}(T_0)^{\phi_2}(\hat{k}^{ac})^{y_g\phi_2})^{-1}, \\
V_{2,1} &= (u^{M^*}h)^t = (k^{(b^2A+y_u)M^*}k^{b^2B+y_h})^c = (k^{b^2c})^{AM^*+B}(k^c)^{y_uM^*+y_h}, \\
V_{2,2} &= ((u^{\nu})^{M^*}h^{\nu})^t = (k^{(b^2A+y_u)aM^*}k^{(b^2B+y_h)a})^c = (T_0)^{AM^*+B}(k^{ac})^{y_uM^*+y_h}, \\
V_{2,3} &= ((u^{-\tau})^{M^*}h^{-\tau})^t = ((k^{(b^2A+y_u)(b+a\phi_2)M^*}k^{(b^2B+y_h)(b+a\phi_2)})^c)^{-1} \\
&= ((k^{b^3c})^{AM^*+B}(k^{bc})^{y_uM^*+y_h}(T_0)^{\phi_2(AM^*+B)}(k^{ac})^{\phi_2(y_uM^*+y_h)})^{-1}.
\end{aligned}$$

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. We obtain $\Pr[\mathcal{B}_1(D, T_0) = 0] = \mathbf{Adv}_{\mathcal{A}}^{G_0}$ and $\Pr[\mathcal{B}_1(D, T_1) = 0] = \mathbf{Adv}_{\mathcal{A}}^{G_1}$ from the above analysis. Thus, we can easily derive the advantage of $\mathcal{B}_1$ as

$$\mathbf{Adv}_{\mathcal{B}_1}^{A1}(\lambda) = \left|\Pr[\mathcal{B}_1(D, T_0) = 0] - \Pr[\mathcal{B}_1(D, T_1) = 0]\right| = \left|\mathbf{Adv}_{\mathcal{A}}^{G_0} - \mathbf{Adv}_{\mathcal{A}}^{G_1}\right|.$$

Note that if $\mathcal{A}$ forge a semi-functional signature, then $\mathcal{B}_1$ can distinguish T by using the forged semi-functional signature. However, if $\mathcal{A}$ forge a normal signature, then $\mathcal{B}_1$ cannot distinguish T since a normal signature is always verified in the normal or semi-functional verification algorithms. This completes our proof. □

Lemma 3.5. *If Assumption 2 holds, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ with non-negligible advantage. That is, for any adversary $\mathcal{A}$, there exists a PPT algorithm $\mathcal{B}_2$ such that $\left|\boldsymbol{Adv}_{\mathcal{A}}^{G_{1,k-1}} - \boldsymbol{Adv}_{\mathcal{A}}^{G_{1,k}}\right| = \boldsymbol{Adv}_{\mathcal{B}_2}^{A2}(\lambda)$.*

Proof. The proof of this lemma is almost same as the proof of Lemma 2 in [17] except that the proof is employed in the PKS setting. Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_{1,k-1}$ and $\mathbf{G}_{1,k}$ with non-negligible advantage. A simulator $\mathcal{B}_2$ that solves Assumption 2 using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), k, k^a, k^b, k^c, \hat{k}^a, \hat{k}^{a^2}, \hat{k}^{bx}, \hat{k}^{abx}, \hat{k}^{a^2x})$ and T where $T = T_0 = k^{bc}$ or $T = T_1 = k^{bc+d}$. Then $\mathcal{B}_2$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}_2$ first selects random exponents $\nu, y_\tau, A, B, \alpha, y_u, y_h, y_w \in \mathbb{Z}_p$. It computes $w_1 = w^{\phi_1} = ((k^b)^{-\nu}k^a k^{y_\tau})^{y_w}, w_2 = w^{\phi_2} = (k^b)^{y_w}, w = k^{y_w}$ by implicitly setting $\phi_1 = -\nu b + (a + y_\tau), \phi_2 = b$. It implicitly sets $\tau = a + y_\tau$ and publishes a public key by selecting

random values $c_g, c_u, c_h \in \mathbb{Z}_p$ as

$$
\begin{aligned}
&gw_1^{c_g} = k^a w_1^{c_g}, w_2^{c_g}, w^{c_g},\ uw_1^{c_u} = (k^a)^A k^{y_u} w_1^{c_u}, w_2^{c_u}, w^{c_u},\\
&hw_1^{c_h} = (k^a)^B k^{y_h} w_1^{c_h}, w_2^{c_h}, w^{c_h},\ w_1, w_2, w,\\
&\hat{g} = \hat{k}^a, \hat{g}^{\nu}, \hat{g}^{-\tau} = (\hat{k}^{a^2}(\hat{k}^a)^{y_\tau})^{-1}),\ \hat{u} = (\hat{k}^a)^A \hat{k}^{y_u}, \hat{u}^{\nu}, \hat{u}^{-\tau} = ((\hat{k}^{a^2})^A (\hat{k}^a)^{y_u + A y_\tau} \hat{k}^{y_u y_\tau})^{-1},\\
&\hat{h} = (\hat{k}^a)^B \hat{k}^{y_h}, \hat{h}^{\nu}, \hat{h}^{-\tau} = ((\hat{k}^{a^2})^B (\hat{k}^a)^{y_h + B y_\tau} \hat{k}^{y_h y_\tau})^{-1},\ \Omega = e(k^a, \hat{k}^a)^\alpha.
\end{aligned}
$$

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. If this is a j-th signature query, then $\mathcal{B}_2$ handles this query as follows:

- Case $j < k$: It creates a semi-functional signature by calling **PKS.SignSF** since it knows the tuple $(f^{-\nu}, f, 1)$ for the semi-functional signature.
- Case $j = k$: It selects random exponents $r', c_1', c_2' \in \mathbb{Z}_p$ and creates a signature by implicitly setting $r = -c + r',\ c_1 = c(AM+B)/y_w + c_1',\ c_2 = c/y_w + c_2'$ as

$$
\begin{aligned}
&W_{1,1} = g^\alpha (k^c)^{-(y_u M + y_h)} (u^M h)^{r'} (T)^{-\nu(AM+B)} (k^c)^{y_\tau (AM+B)} w_1^{c_1'},\\
&W_{1,2} = (T)^{(AM+B)} w_2^{c_1'},\ W_{1,3} = (k^c)^{(AM+B)} w^{c_1'},\\
&W_{2,1} = g^{r'} (T)^{-\nu} (k^c)^{y_\tau} w_1^{c_2'},\ W_{2,2} = T w_2^{c_2'},\ W_{2,3} = k^c w^{c_2'}.
\end{aligned}
$$

- Case $j > k$: It creates a normal signature by calling **PKS.Sign** since it knows the private key.

Finally, $\mathcal{A}$ outputs a forged signature $\sigma^* = (W_{1,1}^*, \ldots, W_{2,3}^*)$ on a message M^*. To verify the forged signature, $\mathcal{B}_2$ first chooses a random exponent $t' \in \mathbb{Z}_p$ and computes semi-functional verification components by implicitly setting $t = bx + t',\ s_c = -a^2 x,\ z_c = AM^* + B$ as

$$
\begin{aligned}
&V_{1,1} = \hat{k}^{abx} (\hat{k}^a)^{t'},\ V_{1,2} = (\hat{k}^{abx})^{\nu} (\hat{k}^a)^{\nu t'} (\hat{k}^{a^2 x})^{-1},\ V_{1,3} = (\hat{k}^{abx})^{-y_\tau} (\hat{g}^{-y_\tau})^{t'},\\
&V_{2,1} = (\hat{k}^{abx})^{AM^*+B} (\hat{k}^{bx})^{y_u M^* + y_h} (\hat{u}^{M^*} \hat{h})^{t'},\\
&V_{2,2} = (\hat{k}^{abx})^{(AM^*+B)\nu} (\hat{k}^{bx})^{(y_u M^* + y_h)\nu} (\hat{u}^{M^*} \hat{h})^{\nu t'},\\
&V_{2,3} = (\hat{k}^{abx})^{-(AM^*+B) y_\tau} (\hat{k}^{abx})^{-(y_u M^* + y_h)} (\hat{k}^{bx})^{-(y_u M^* + y_h) y_\tau} ((\hat{u}^{-\tau})^{M^*} \hat{h}^{-\tau})^{t'}.
\end{aligned}
$$

Next, it verifies that $\prod_{i=1}^3 e(W_{1,i}^*, V_{1,i}) \cdot \prod_{i=1}^3 e(W_{2,i}^*, V_{2,i})^{-1} \stackrel{?}{=} e(k^a, \hat{k}^{abx})^\alpha \cdot e(k^a, \hat{k}^a)^{\alpha t'}$. If this equation holds, then it outputs 0. Otherwise, it outputs 1.

To finish the proof, we should show that the distribution of the simulation is correct. We first show that the distribution of the simulation using $D, T_0 = k^{bc}$ is the same as $\mathbf{G}_{1,k-1}$. The public key is correctly distributed since the random blinding values y_u, y_h, y_w, y_v are used. The k-th signature is correctly distributed as

$$
\begin{aligned}
W_{1,1} &= g^\alpha (u^M h)^r w_1^{c_1} = g^\alpha (k^{(aA+y_u)M} k^{aB+y_h})^{-c+r'} (k^{y_w(-\nu b + a + y_\tau)})^{c(AM+B)/y_w + c_1'}\\
&= g^\alpha (k^c)^{-(y_u M + y_h)} (u^M h)^{r'} (T)^{-nu(AM+B)} (k^c)^{y_\tau (AM+B)} w_1^{c_1'},\\
W_{1,2} &= w_2^{c_1} = (k^{y_w b})^{c(AM+B)/y_w + c_1'} = (T)^{(AM+B)} w_2^{c_1'},\\
W_{1,3} &= w^{c_1} = (k^{y_w})^{c(AM+B)/y_w + c_1'} = (k^c)^{(AM+B)} w^{c_1'}.
\end{aligned}
$$

The semi-functional verification components are correctly distributed as

$$
\begin{aligned}
V_{2,1} &= (\hat{u}^{M^*}\hat{h})^t = (\hat{k}^{(aA+y_u)M^*}\hat{k}^{aB+y_h})^{bx+t'} = (\hat{k}^{abx})^{AM^*+B}(\hat{k}^{bx})^{y_uM^*+y_h}(\hat{u}^{M^*}\hat{h})^{t'},\\
V_{2,2} &= ((\hat{u}^{v})^{M^*}\hat{h}^{v})^t\hat{f}^{s_cz_c} = (\hat{k}^{(aA+y_u)vM^*}\hat{k}^{(aB+y_h)v})^{bx+t'}\hat{k}^{-a^2x(AM^*+B)}\\
&= (\hat{k}^{abx})^{(AM^*+B)v}(\hat{k}^{bx})^{(y_uM^*+y_h)v}((\hat{u}^{v})^{M^*}\hat{h}^{v})^{t'}(\hat{k}^{a^2x})^{-(AM^*+B)},\\
V_{2,3} &= ((\hat{u}^{-\tau})^{M^*}\hat{h}^{-\tau})^t(\hat{f}^{-\phi_2})^{s_cz_c} = (\hat{k}^{-(aA+y_u)(a+y_\tau)M^*}\hat{k}^{-(aB+y_h)(a+y_\tau)})^{bx+t'}\hat{k}^{-b(-a^2x)(AM^*+B)}\\
&= (\hat{k}^{abx})^{-(AM^*+B)y_\tau-(y_uM^*+y_h)}(\hat{k}^{bx})^{-(y_uM^*+y_h)y_\tau}((\hat{u}^{-\tau})^{M^*}\hat{h}^{-\tau})^{t'}.
\end{aligned}
$$

The simulator can create the semi-functional verification components with only fixed $z_c = AM^* + B$ since s_c, s_c enable the cancellation of $\hat{k}^{a^2bx}$. Even though it uses the fixed z_c, the distribution of z_c is correct since A, B are information theoretically hidden to $\mathcal{A}$. We next show that the distribution of the simulation using $D, T_1 = k^{bc+d}$ is the same as $\mathbf{G}_{1,k}$. We only consider the distribution of the k-th signature since T is only used in the k-th signature. The only difference between T_0 and T_1 is that T_1 additionally has k^d. The signature components $W_{1,1}, W_{1,2}, W_{2,1}, W_{2,2}$ that have T in the simulation additionally have $(k^d)^{-v(AM+B)}$, $(k^d)^{(AM+B)}$, $(k^d)^{-v}$, k^d respectively. If we implicitly set $s_k = d, z_k = AM + B$, then the distribution of the k-th signature is the same as $\mathbf{G}_{1,k}$ except that the k-th signature is nominally semi-functional.

Finally, we show that $\mathcal{A}$ cannot distinguish the nominally semi-functional signature from the semi-functional signature. The main idea of this is that $\mathcal{A}$ cannot request a signature for the forgery message M^* in the security model. Suppose there exists an unbounded adversary, then he can gather the values $z_k = AM + B$ from the k-th signature and $z_c = AM^* + B$ from the forged signature. It is easy to show that z_k, z_c look random to the unbounded adversary since $f(M) = AM + B$ is pair-wise independent function and A, B are information theoretically hidden to the adversary. We obtain $\Pr[\mathcal{B}_2(D, T_0) = 0] = \mathbf{Adv}_{\mathcal{A}}^{G_{1,k-1}}$ and $\Pr[\mathcal{B}_2(D, T_1) = 0] = \mathbf{Adv}_{\mathcal{A}}^{G_{1,k}}$ from the above analysis. Thus, we can easily derive the advantage of $\mathcal{B}_2$ as

$$
\mathbf{Adv}_{\mathcal{B}_2}^{A2}(\lambda) = \left|\Pr[\mathcal{B}_2(D, T_0) = 0] - \Pr[\mathcal{B}_2(D, T_1) = 0]\right| = \left|\mathbf{Adv}_{\mathcal{A}}^{G_{1,k-1}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,k}}\right|.
$$

This completes our proof. □

Lemma 3.6. *If Assumption 3 holds, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_2$ and $\boldsymbol{G}_3$ with non-negligible advantage. That is, for any adversary $\mathcal{A}$, there exists a PPT algorithm $\mathcal{B}_3$ such that $\left|\boldsymbol{Adv}_{\mathcal{A}}^{G_2} - \boldsymbol{Adv}_{\mathcal{A}}^{G_3}\right| = \boldsymbol{Adv}_{\mathcal{B}_3}^{A3}(\lambda)$.*

Proof. The proof of this lemma is almost same as the proof of Lemma 3 in [17] except that the proof is employed in the PKS setting. Suppose there exists an adversary $\mathcal{A}$ that distinguish $\mathbf{G}_2$ from $\mathbf{G}_3$ with non-negligible advantage. A simulator $\mathcal{B}_3$ that solves Assumption 3 using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), k, k^a, k^b, k^c, \hat{k}, \hat{k}^a, \hat{k}^b, \hat{k}^c)$ and T where $T = T_0 = e(k, \hat{k})^{abc}$ or $T = T_1 = e(k, \hat{k})^d$. Then $\mathcal{B}_3$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}_3$ first chooses random exponents $\phi_1, \phi_2, y_g, x, y \in \mathbb{Z}_p$ and random elements $w \in \mathbb{G}$. It computes $g = k^{y_g}, u = g^x, h = g^y, \hat{g} = \hat{k}^{y_g}, \hat{u} = \hat{g}^x, \hat{h} = \hat{g}^y, w_1 = w^{\phi_1}, w_2 = w^{\phi_2}$. It implicitly sets $v = a, \tau = \phi_1 + a\phi_2, \alpha = ab$ and publishes a public key by selecting random values $c_g, c_u, c_h \in \mathbb{Z}_p$ as

$$
\begin{aligned}
&gw_1^{c_g}, w_2^{c_g}, w^{c_g},\ uw_1^{c_u}, w_2^{c_u}, w^{c_u},\ hw_1^{c_h}, w_2^{c_h}, w^{c_h},\ w_1, w_2, w,\\
&\hat{g}, \hat{g}^{v} = (\hat{k}^a)^{y_g}, \hat{g}^{-\tau} = \hat{k}^{-y_g\phi_1}(\hat{k}^a)^{-y_g\phi_2},\ \hat{u}, \hat{u}^{v} = (\hat{g}^{v})^x, \hat{u}^{-\tau} = (\hat{g}^{-\tau})^x,\\
&\hat{h}, \hat{h}^{v} = (\hat{g}^{v})^y, \hat{h}^{-\tau} = (\hat{g}^{-\tau})^y,\ \Omega = e(k^a, \hat{k}^b)^{y_g^2}.
\end{aligned}
$$

Additionally, it sets $f = k, \hat{f} = \hat{k}$ for the semi-functional signature and semi-functional verification. $\mathcal{A}$ adaptively requests a signature for a message M. To respond to this query, $\mathcal{B}_3$ selects random exponents $r, c_1, c_2, s_k, z'_k \in \mathbb{Z}_p$ and creates a semi-functional signature by implicitly setting $z_k = by_g/s_k + z'_k$ as

$$W_{1,1} = (u^M h)^r w_1^{c_1} (k^a)^{-s_k z'_k},\ W_{1,2} = w_2^{c_1} (k^b)^{y_g} k^{s_k z'_k},\ W_{1,3} = w^{c_1},$$
$$W_{2,1} = g^r w_1^{c_2} (k^a)^{-s_k},\ W_{2,2} = w_2^{c_2} k^{s_k},\ W_{2,3} = w^{c_2}.$$

It can only create a semi-functional signature since s_k, z_k enables the cancellation of k^{ab}. Finally, $\mathcal{A}$ outputs a forged signature $\sigma^* = (W^*_{1,1}, \ldots, W^*_{2,3})$ on a message M^*. To verify the forged signature, $\mathcal{B}_3$ first chooses random exponents $s_1, s_2, s'_c, z'_c \in \mathbb{Z}_p$ and computes semi-functional verification components by implicitly setting $t = c,\ s_c = -acy_g + s'_c,\ z_c = -acy_g(xM^* + y)/s_c + z'_c/s_c$ as

$$V_{1,1} = (\hat{k}^c)^{y_g},\ V_{1,2} = \hat{k}^{s'_c},\ V_{1,3} = (\hat{k}^c)^{-y_g \phi_1} \hat{k}^{-\phi_2 s'_c},$$
$$V_{2,1} = (\hat{k}^c)^{y_g (xM^* + y)},\ V_{2,2} = \hat{k}^{z'_c},\ V_{2,3} = (\hat{k}^c)^{-y_g \phi_1 (xM^* + y)} \hat{k}^{-\phi_2 z'_c}.$$

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

To finish the proof, we first show that the distribution of the simulation using $D, T = e(k, \hat{k})^{abc}$ is the same as $\mathbf{G}_2$. The public key is correctly distributed since the random values y_g, x, y, c_g, c_u, c_h are used. The semi-functional signature is correctly distributed as

$$\begin{aligned} W_{1,1} &= g^\alpha (u^M h)^r w_1^{c_1} (f^{-\nu})^{s_k z_k} = k^{y_g ab} (u^M h)^r w_1^{c_1} (k^{-a})^{s_k (by_g/s_k + z'_k)} \\ &= (u^M h)^r w_1^{c_1} (k^a)^{-s_k z'_k}. \end{aligned}$$

The simulator can only create a semi-functional signature since $z_k = by_g/s_k + z'_k$ enables the cancellation of k^{ab}. The semi-functional verification components are correctly distributed as

$$\begin{aligned} V_{1,1} &= \hat{g}^t = (\hat{k}^{y_g})^c = (\hat{k}^c)^{y_g}, \\ V_{1,2} &= (\hat{g}^\nu)^t \hat{f}^{s_c} = (\hat{k}^{y_g a})^c \hat{k}^{-acy_g + s'_c} = \hat{k}^{s'_c}, \\ V_{1,4} &= (\hat{g}^{-\tau})^t (\hat{f}^{-\phi_2})^{s_c} = (\hat{k}^{-y_g(\phi_1 + a\phi_2)})^c \hat{k}^{-\phi_2(-acy_g + s'_c)} = (\hat{k}^c)^{-y_g \phi_1} \hat{k}^{-\phi_2 s'_c}, \\ V_{2,1} &= (\hat{u}^{M^*} \hat{h})^t = (\hat{k}^{y_g (xM^* + y)})^c = (\hat{k}^c)^{y_g (xM^* + y)}, \\ V_{2,2} &= (\hat{u}^{\nu M^*} \hat{h}^\nu)^t \hat{f}^{s_c z_c} = (\hat{k}^{y_g a (xM^* + y)})^c \hat{k}^{-acy_g (xM^* + y) + z'_c} = \hat{k}^{z'_c}, \\ V_{2,3} &= (\hat{u}^{-\tau M^*} \hat{h}^{-\tau})^t (\hat{f}^{-\phi_2})^{s_c z_c} = (\hat{k}^{-y_g(\phi_1 + a\phi_2)(xM^* + y)})^c (\hat{k}^{-\phi_2})^{-acy_g (xM^* + y) + z'_c} \\ &= (\hat{k}^c)^{-y_g \phi_1 (xM^* + y)} \hat{k}^{-\phi_2 z'_c}, \\ \Omega^t &= e(g, \hat{g})^{\alpha t} = e(k, \hat{k})^{y_g^2 abc} = (T_0)^{y_g^2}. \end{aligned}$$

We next show that the distribution of the simulation using $D, T_1 = e(k, \hat{k})^d$ is almost the same as $\mathbf{G}_3$. It is obvious that the signature verification for the forged signature always fails if $T_1 = e(k, \hat{k})^d$ is used except with $1/p$ probability since d is a random value in $\mathbb{Z}_p$. We obtain $\Pr[\mathcal{B}_3(D, T_0) = 0] = \mathbf{Adv}^{G_2}_{\mathcal{A}}$ and $\Pr[\mathcal{B}_3(D, T_1) = 0] = \mathbf{Adv}^{G_3}_{\mathcal{A}}$ from the above analysis. Thus, we can easily derive the advantage of $\mathcal{B}_3$ as

$$\mathbf{Adv}^{A3}_{\mathcal{B}_3}(\lambda) = \left| \Pr[\mathcal{B}_3(D, T_0) = 0] - \Pr[\mathcal{B}_3(D, T_1) = 0] \right| = \left| \mathbf{Adv}^{G_2}_{\mathcal{A}} - \mathbf{Adv}^{G_3}_{\mathcal{A}} \right|.$$

Note that if $\mathcal{A}$ forge a normal signature, then $\mathcal{B}_3$ can distinguish T by using the forged normal signature. However, if $\mathcal{A}$ forge a semi-functional signature, then $\mathcal{B}_3$ cannot distinguish T since the verification of $\mathcal{B}_3$ by using the semi-functional verification components always fails except negligible probability. This completes our proof. □

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. [19]. 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 pubic keys of all signers in the aggregate signature. A SAS scheme is formally defined as follows:

Definition 4.1 (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^λ). *The setup algorithm takes as input a security parameter 1^λ 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 $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$, 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 $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$, 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 **AggVerify**$($**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 $\mathbf{PK}'$.*

A trivial SAS scheme can be constructed from a PKS scheme by concatenating each signer's signature in sequential order, but the size of aggregate signature is proportional to the size of signers. Therefore, a non-trivial SAS scheme should satisfy the signature compactness property that requires the size of aggregate signature to be independent of the size of signers.

The security model of SAS was defined by Lysyanskaya et al. [19], but we use the security model of Lu et al. [18] that requires for an adversary to register the key-pairs of other signers except the target signer, namely the knowledge of secret key (KOSK) setting or the proof of knowledge (POK) setting. In this security model, an adversary first given the public key of a target signer. After that, the adversary adaptively requests a certification for a public key by registering the key-pair of other signer, and he adaptively requests a sequential aggregate signature by providing a previous aggregate signature to the signing oracle. Finally, the adversary outputs a forged sequential aggregate signature on messages under public keys. If the forged sequential signature satisfies the conditions of the security model, then the adversary wins the security game. The security model of SAS is formally defined as follows:

Definition 4.2 (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) **AggVerify**$(AS^*, \mathbf{M}^*, \mathbf{PK}^*) = 1$, 2) The challenge public key PK must exists 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 $\mathbf{Adv}_{\mathcal{A}}^{SAS} = \Pr[\mathcal{C} = 1]$ 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. [18] 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^λ): 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

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

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

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

SAS.AggSign$(AS', \mathbf{M}', \mathbf{PK}', M, SK)$**:** This algorithm takes as input an aggregate-so-far $AS' = (S'_{1,1}, \ldots, S'_{2,3})$ on messages $\mathbf{M}' = (M_1, \ldots, M_{l-1})$ under public keys $\mathbf{PK}' = (PK_1, \ldots, PK_{l-1})$ where $PK_i = (u_i w_1^{c_{u,i}}, \ldots, \Omega_i)$, a message $M \in \mathbb{Z}_p$, a private key $SK = (\alpha, x, y)$ with $PK = (u w_1^{c_u}, \ldots, \Omega)$ and PP. It first checks the validity of AS' by calling **SAS.AggVerify**$(AS', \mathbf{M}', \mathbf{PK}')$. If AS' is not valid, then it halts. If the public key PK of SK does already exist in $\mathbf{PK}'$, then it halts. Next, it computes a temporal aggregate signature as

$$TS = \big(S_{1,1} = S'_{1,1}(g w_1^{c_g})^{\alpha} (S'_{2,1})^{xM+y},\ S_{1,2} = S'_{1,2}(w_2^{c_g})^{\alpha}(S'_{2,2})^{xM+y},\ S_{1,3} = S'_{1,3}(w^{c_g})^{\alpha}(S'_{2,3})^{xM+y},$$
$$S_{2,1} = S'_{2,1},\ S_{2,2} = S'_{2,2},\ S_{2,3} = S'_{2,3} \big).$$

Finally it output a re-randomized aggregate signature AS by running **SAS.AggRand**$(TS, \mathbf{M}, \mathbf{PK}, PP)$ where $\mathbf{M} = \mathbf{M}' || M$ and $\mathbf{PK} = \mathbf{PK}' || PK$.

SAS.AggRand$(AS', \mathbf{M}', \mathbf{PK}', PP)$**:** This algorithm takes as input an aggregate signature $AS' = (S'_{1,1}, \ldots, S'_{2,3})$ on messages $\mathbf{M}' = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK}' = (PK_1, \ldots, PK_l)$ where $PK_i = (u_i w_1^{c_{u,i}}, \ldots, \Omega_i)$, and PP. Next, it selects random exponents $r, c_1, c_2 \in \mathbb{Z}_p$ and outputs a randomized aggregate signature as

$$AS = \big(S_{1,1} = S'_{1,1} \cdot \prod_{i=1}^{l} ((u_i w_1^{c_{u,i}})^{M_i} (h_i w_1^{c_{h,i}}))^r w_1^{c_1},\ S_{1,2} = S'_{1,2} \cdot \prod_{i=1}^{l} ((w_2^{c_{u,i}})^{M_i} (w_2^{c_{h,i}}))^r w_2^{c_1},$$
$$S_{1,3} = S'_{1,3} \cdot \prod_{i=1}^{l} ((w^{c_{u,i}})^{M_i} (w^{c_{h,i}}))^r w^{c_1},$$
$$S_{2,1} = S'_{2,1} \cdot (g w_1^{c_g})^r w_1^{c_2},\ S_{2,2} = S'_{2,2} \cdot (w_2^{c_g})^r w_2^{c_2},\ S_{2,3} = S'_{2,3} \cdot (w^{c_g})^r w^{c_2} \big).$$

SAS.AggVerify$(AS, \mathbf{M}, \mathbf{PK})$**:** This algorithm takes as input a sequential aggregate signature AS on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $PK_i = (u_i w_1^{c_{u,i}}, \ldots, \Omega_i)$. It first checks that any public key does not appear twice in $\mathbf{PK}$ and that any public key in $\mathbf{PK}$ has been certified. If these checks fail, then it outputs 0. If $l = 0$, then it outputs 1 if $S_{1,1} = \cdots = S_{2,3} = 1$, 0 otherwise. It chooses a random exponent $t \in \mathbb{Z}_p$ and computes verification components as

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

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.

Let r', c'_1, c'_2 be the randomness of an aggregate-so-far. If we implicitly sets $\tilde{r} = r' + r$, $\tilde{c}_1 = c'_1 + c_g \alpha_l + \sum_{i=1}^{l} (c_{u,i} M_i + c_{h,i}) r + c_1$, $\tilde{c}_2 = c'_2 + c_g r + c_2$, then the aggregate signature is correctly distributed as

$$S_{1,1} = \prod_{i=1}^{l} g^{\alpha_i} \prod_{i=1}^{l} (u_i^{M_i} h_i)^{\tilde{r}} w_1^{\tilde{c}_1},\ S_{1,2} = w_2^{\tilde{c}_1},\ S_{1,3} = w^{\tilde{c}_1},$$
$$S_{2,1} = g^{\tilde{r}} w_1^{\tilde{c}_2},\ S_{2,2} = w_2^{\tilde{c}_2},\ S_{2,3} = w^{\tilde{c}_2}.$$

4.3 Security Analysis

Theorem 4.3. *The above SAS scheme is existentially unforgeable under a chosen message attack if the PKS scheme is existentially unforgeable under a chosen message attack. That is, for any PPT adversary $\mathcal{A}$ for the above SAS scheme, there exists a PPT algorithm $\mathcal{B}$ for the PKS scheme such that $\boldsymbol{Adv}_{\mathcal{A}}^{SAS}(\lambda) \leq \boldsymbol{Adv}_{\mathcal{B}}^{PKS}(\lambda)$.*

Proof. The security proof of this theorem is similar to that of Lu et al. [18]. That is, a simulator can simulate the generation of aggregate signatures since the verification algorithm of aggregate signatures does not check the order of aggregation and the simulator knows the private keys of other signers except the target signer. To extract a forged PKS signature from the forged aggregate signature, the simulator uses the knowledge of private keys of other signers.

Suppose there exists an adversary $\mathcal{A}$ that forges the above SAS scheme with non-negligible advantage ε. A simulator $\mathcal{B}$ that forges the PKS scheme is first given: a challenge public key $PK_{PKS} = (gw_1^{c_g}, w_2^{c_g}, w^{c_g}, uw_1^{c_u}, \ldots, w^{c_h}, w_1, w_2, w, \hat{g}, \hat{g}^{\nu}, \hat{g}^{-\tau}, \hat{u}, \ldots, \hat{h}^{-\tau}, \Omega)$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}$ first constructs $PP = (gw_1^{c_g}, w_2^{c_g}, w^{c_g}, w_1, w_2, w, \hat{g}, \hat{g}^{\nu}, \hat{g}^{-\tau}, \Lambda)$ by computing $\Lambda = e(gw_1^{c_g}, \hat{g}) \cdot e(w_2^{c_g}, \hat{g}^{\nu}) \cdot e(w^{c_g}, \hat{g}^{-\tau}) = e(g, \hat{g})$ and $PK^* = (uw_1^{c_u}, \ldots, w^{c_h}, \hat{u}, \ldots, \hat{h}^{-\tau}, \Omega)$ from PK_{PKS}. Next, it initializes a certification list CL as an empty one and gives PP and PK^* to $\mathcal{A}$. $\mathcal{A}$ 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 $PK_i = (u_i w_1^{c_{u,i}}, \ldots, \Omega_i)$ 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 $\mathbf{M}' = (M_1, \ldots, M_{l-1})$ under public keys $\mathbf{PK}' = (PK_1, \ldots, PK_{l-1})$, 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 σ.
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 $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a forged aggregate signature $AS^* = (S_{1,1}^*, \ldots, S_{2,3}^*)$ on messages $\mathbf{M}^* = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK}^* = (PK_1, \ldots, PK_l)$ 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 $PK_i = (u_i w_1^{c_{u,i}}, \ldots, \Omega_i)$ 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

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

3. It outputs $\sigma = (W_{1,1}, \ldots, W_{2,3})$ 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 $\sigma = (W_{1,1}, \ldots, W_{2,3})$ of the simulator is a valid PKS signature on the message M_1 under the public key PK^* since it satisfies the following equation:

$$
\begin{aligned}
&\prod_{i=1}^{3} e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{3} e(W_{2,i}, V_{2,i})^{-1} \\
&= e(S^*_{1,1}, \hat{g}^t) \cdot e(S^*_{1,2}, \hat{g}^{vt}) \cdot e(S^*_{1,4}, \hat{g}^{-\tau t}) \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g}^t)^{-1} \cdot \\
&\quad e(S^*_{2,1}, \prod_{i=2}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^t)^{-1} \cdot e(S^*_{2,2}, \prod_{i=2}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{vt})^{-1} \cdot e(S^*_{2,3}, \prod_{i=2}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{-\tau t})^{-1} \cdot \\
&\quad e(S^*_{2,1}, (\hat{u}^{M_1} \hat{h})^t)^{-1} \cdot e(S^*_{2,2}, (\hat{u}^{M_1} \hat{h})^{vt})^{-1} \cdot e(S^*_{2,3}, (\hat{u}^{M_1} \hat{h})^{-\tau t})^{-1} \\
&= e(S^*_{1,1}, C_{1,1}) \cdot e(S^*_{1,2}, C_{1,2}) \cdot e(S^*_{1,3}, C_{1,3}) \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g}^t)^{-1} \cdot \\
&\quad e(S^*_{2,1}, \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^t)^{-1} \cdot e(S^*_{2,2}, \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{vt})^{-1} \cdot e(S^*_{2,3}, \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{-\tau t})^{-1} \cdot \\
&= \prod_{i=1}^{3} e(S^*_{1,i}, C_{1,i}) \cdot \prod_{i=1}^{3} e(S^*_{2,i}, C_{2,i})^{-1} \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g}^t)^{-1} = \prod_{i=1}^{l} \Omega_i^t \cdot \prod_{i=2}^{l} \Omega_i^{-t} = \Omega_1^t
\end{aligned}
$$

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. □

5 Multi-Signature

In this section, we propose an efficient multi-signature (MS) scheme and prove its security without random oracles.

5.1 Definitions

Multi-Signature (MS) can be regarded as a special kind of PKAS such that different signatures generated by different signer on the same message are combined as a short multi-signature. Thus MS scheme consists of four algorithms of PKS scheme and additionally two algorithms **Combine** and **MultiVerify** for combining a multi-signature and verifying a multi-signature. In MS, each signer generates a public key and a private key, and he can generate an individual signature on a message by using his private key. To generate a multi-signature, anyone can combine individual signature of different signers on the same message. A verifier can check the validity of the multi-signature by using the public keys of signers. A MS scheme is formally defined as follows:

Definition 5.1 (Multi-Signature)**.** *A multi-signature (MS) scheme consists of six PPT algorithms **Setup**, **KeyGen**, **Sign**, **Verify**, **Combine**, and **MultVerify**, which are defined as follows:*

Setup(1^λ)*: The setup algorithm takes as input a security parameter λ, 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.*

Sign(M,SK)*: The signing algorithm takes as input a message M, and a private key SK. It outputs a signature* σ*.*

Verify(σ,M,PK)*: The verification algorithm takes as input a signature* σ *on a message M under a public key PK, and outputs either* 1 *or* 0 *depending on the validity of the signature.*

Combine$(\sigma,M,\mathbf{PK})$*: The combining algorithm takes as input signatures* σ *on a message M under public keys* $\mathbf{PK} = (PK_1,\ldots,PK_l)$*, and outputs a multi-signature MS.*

MultVerify$(MS,M,\mathbf{PK})$*: The multi-verification algorithm takes as input a multi-signature MS on a message M under public keys* $\mathbf{PK} = (PK_1,\ldots,PK_l)$*, and outputs either* 1 *or* 0 *depending on the validity of the multi-signature.*

The correctness requirement is that for each PP output by ***Setup***(1^λ)*, for all* (PK,SK) *output by* ***KeyGen***(PP)*, and any M, we have that* ***Verify***$($***Sign***$(M,SK),M,PK) = 1$ *and for each* σ *on message M under public keys* **PK***,* ***MultVerify***$($***Combine***$(\sigma,M,\mathbf{PK}),M,\mathbf{PK}) = 1$*.*

The security model of MS was defined by Micali et al. [20], but we use the security model of Boldyreva [3] that requires for an adversary to register the key-pairs of other signers except the target signer, namely the knowledge of secret key (KOSK) setting or the proof of knowledge (POK) setting. In this security model, an adversary first given the public key of a target signer. After that, the adversary adaptively requests a certification for a public key by registering the key-pair of other signer, and he adaptively requests a signature for the target signer on a message. Finally, the adversary outputs a forged multi-signature on a message M^* under public keys. If the forged multi-signature satisfies the conditions of the security model, then the adversary wins the security game. The security model of MS is formally defined as follows:

Definition 5.2 (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 initialize the 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 PP,PK to* $\mathcal{A}$*.*

2. ***Certification Query****:* $\mathcal{A}$ *adaptively requests the certification of a public key by providing a key pair* (PK,SK)*.* $\mathcal{C}$ *adds the key pair* (PK,SK) *to CL if the private key is a valid one.*

3. ***Signature Query****:* $\mathcal{A}$ *adaptively requests a signature by providing a message M to sign under the challenge public key PK, and receives a signature* σ*.*

4. ***Output****: Finally,* $\mathcal{A}$ *outputs a forged multi-signature* MS^* *on a message* M^* *under public keys* $\mathbf{PK}^*$*.* $\mathcal{C}$ *outputs* 1 *if the forged signature satisfies the following three conditions, or outputs* 0 *otherwise: 1)* ***MultVerify***$(MS^*,M^*,\mathbf{PK}^*) = 1$*, 2) The challenge public key PK must exists in* $\mathbf{PK}^*$ *and each public key in* $\mathbf{PK}^*$ *except the challenge public key must be in CL, and 3) The message* M^* *must not have been queried by* $\mathcal{A}$ *to the signing oracle.*

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

5.2 Construction

To construct a MS scheme from a PKS scheme, the PKS scheme should support multi-user settings and the randomness aggregation method. If the elements $gw_1^{c_g}, uw_1^{c_u}, hw_1^{c_h}$ of the PKS scheme are shared among all signers, then we can construct a MS scheme from the PKS scheme since the random exponent for the public-key and the randomness for the signature are placed in different positions. Note that it is not required for a signer to publicly re-randomize the multi-signature since each signer selects an independent random value.

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

MS.Setup(1^λ)**:** 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 selects random exponents $x, y \in \mathbb{Z}_n$ and computes $u = g^x, h = g^y, \hat{u} = \hat{g}^x, \hat{h} = \hat{g}^y$. It publishes public parameters by selecting random values $c_g, c_u, c_h \in \mathbb{Z}_p$ as

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

MS.KeyGen(PP)**:** This algorithm takes as input the public parameters PP. It selects a random exponent $\alpha \in \mathbb{Z}_p$ and computes $\Omega = \Lambda^\alpha$. Then it outputs a private key $SK = \alpha$ and a public key as $PK = \Omega$.

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

$$\sigma = \big(W_{1,1} = (gw_1^{c_g})^\alpha ((uw_1^{c_u})^M (hw_1^{c_h}))^r w_1^{c_1},$$
$$W_{1,2} = (w_2^{c_g})^\alpha ((w_2^{c_u})^M w_2^{c_h})^r w_2^{c_1},\ W_{1,3} = (w^{c_g})^\alpha ((w^{c_u})^M w^{c_h})^r w^{c_1},$$
$$W_{2,1} = (gw_1^{c_g})^r w_1^{c_2},\ W_{2,2} = (w_2^{c_g})^r w_2^{c_2},\ W_{2,3} = (w^{c_g})^r w^{c_2} \big).$$

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

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

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.

MS.Combine$(\sigma, M, \mathbf{PK})$**:** This algorithm takes as input signatures $\sigma = (\sigma_1, \ldots, \sigma_l)$ on a message M under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $PK_i = \Omega_i$. It first checks the validity of each signature $\sigma_i = (W_{1,1}^i, \ldots, W_{2,3}^i)$ by calling **MS.Verify**(σ_i, M, PK_i). If any signature is invalid, then it halts. It then outputs a multi-signature for a message M as

$$MS = \big(S_{1,1} = \prod_{i=1}^l W_{1,1}^i,\ S_{1,2} = \prod_{i=1}^l W_{1,2}^i,\ S_{1,3} = \prod_{i=1}^l W_{1,3}^i,$$
$$S_{2,1} = \prod_{i=1}^l W_{2,1}^i,\ S_{2,2} = \prod_{i=1}^l W_{2,2}^i,\ S_{2,3} = \prod_{i=1}^l W_{2,3}^i \big).$$

MS.MultVerify$(MS, M, \mathbf{PK})$**:** This algorithm takes as input a multi-signature MS on a message M under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $PK_i = \Omega_i$. It chooses a random exponent $t \in \mathbb{Z}_p$ and computes verification components as

$$V_{1,1} = \hat{g}^t, V_{1,2} = (\hat{g}^v)^t, V_{1,3} = (\hat{g}^{-\tau})^t,$$
$$V_{2,1} = (\hat{u}^M \hat{h})^t, V_{2,2} = ((\hat{u}^v)^M \hat{h}^v)^t, V_{2,3} = ((\hat{u}^{-\tau})^M \hat{h}^{-\tau})^t.$$

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

5.3 Security Analysis

Theorem 5.3. *The above MS scheme is existentially unforgeable under a chosen message attack if the PKS scheme is existentially unforgeable under a chosen message attack. That is, for any PPT adversary $\mathcal{A}$ for the above MS scheme, there exists a PPT algorithm $\mathcal{B}$ for the PKS scheme such that $\boldsymbol{Adv}_{\mathcal{A}}^{MS}(\lambda) \leq \boldsymbol{Adv}_{\mathcal{B}}^{PKS}(\lambda)$.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that forges the above MS scheme with a non-negligible advantage ε. A simulator $\mathcal{B}$ that forges the PKS scheme is given: a challenge public key $PK_{PKS} = (gw_1^{c_g}, \ldots, \Lambda, \Omega)$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}$ first constructs $PP = (gw_1^{c_g}, \ldots, \Lambda)$ by computing $\Lambda = e(gw_1^{c_g}, \hat{g}) \cdot e(w_2^{c_g}, \hat{g}^v) \cdot e(w^{c_g}, \hat{g}^{-\tau}) = e(g, \hat{g})$ and $PK^* = \Omega$ from PK_{PKS}. Next, it initialize a certification list CL as an empty one and gives PP and PK^* to $\mathcal{A}$. $\mathcal{A}$ may adaptively request certification queries or signature queries. If $\mathcal{A}$ requests the certification of a public key by providing a public key $PK_i = \Omega_i$ and its private key $SK_i = \alpha_i$, then $\mathcal{B}$ checks the key pair and adds (PK_i, SK_i) to CL. If $\mathcal{A}$ requests a signature by providing a message M to sign under the challenge private key of PK^*, then $\mathcal{B}$ queries its signing oracle that simulates **PKS.Sign** on the message M for the challenge public key PK^*, and gives the signature to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a forged multi-signature $MS^* = (S_{1,1}^*, \ldots, S_{2,3}^*)$ on a message M^* under public keys $\mathbf{PK}^* = (PK_1, \ldots, PK_l)$ for some l. Without loss of generality, we assume that $PK_1 = PK^*$. $\mathcal{B}$ proceeds as follows:

1. $\mathcal{B}$ first check the validity of MS^* by calling **MS.MultVerify**. Additionally, the forged signature should not be trivial: the challenge public key PK^* must be in $\mathbf{PK}^*$, and the message M must not be queried by $\mathcal{A}$ to the signing oracle.

2. For each $2 \leq i \leq l$, it parses $PK_i = \Omega_i$ from $\mathbf{PK}^*$, and it retrieves the private key $SK_i = g^{\alpha_i}$ of PK_i from CL. It then computes

$$W_{1,1} = S_{1,1}^* \cdot \prod_{i=2}^{l} \left(g^{\alpha_i}\right)^{-1},\ W_{1,2} = S_{1,2}^*,\ W_{1,3} = S_{1,3}^*,$$
$$W_{2,1} = S_{2,1}^*,\ W_{2,2} = S_{2,2}^*,\ W_{2,3} = S_{2,3}^*.$$

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

To finish the proof, we first show that the distribution of the simulation is correct. It is obvious that the public parameters, the public key, and the signatures are correctly distributed. Next we show that the output

signature $\sigma = (W_{1,1}, \ldots, W_{2,3})$ of the simulator is a valid signature for the PKS scheme on the message M_1 under the public key PK^* since it satisfies the following equation

$$\prod_{i=1}^{3} e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{3} e(W_{2,i}, V_{2,i})^{-1}$$
$$= \prod_{i=1}^{3} e(S^*_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{3} e(S^*_{2,i}, V_{2,i})^{-1} \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g})^{-1} = \prod_{i=1}^{l} \Omega_i^t \cdot \prod_{i=2}^{l} \Omega_i^{-t} = \Omega_1^t.$$

This completes our proof. □

5.4 Discussions

Removing the Proof of Knowledge. In our MS scheme, an adversary should prove that he knows the private key of other signer by using a zero-knowledge proof system. Ristenpart and Yilek showed that some MS schemes can be proven in the proof of possession (POP) setting instead of the POK setting [22]. Our MS scheme also can be proven in the POP setting by using their technique. That is, if our MS scheme is incorporated with a POP scheme that uses a different hash function, and the adversary submits a signature on the private key of other signer as the proof of possession, then the security of our scheme is also achieved. In the security proof, a simulator cannot extract the private key element g^{α} from the signature of the POP scheme, but he can extract other values $g^{\alpha} w_1^{c'}, w_2^{c'}, w^{c'}$ and these values are enough for the security proof.

6 Conclusion

In this paper, we improved the SAS scheme of Lee et al. [16] 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] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *ACM Conference on Computer and Communications Security*, pages 473–484, 2010.

[2] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 411–422. Springer, 2007.

[3] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[4] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. 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] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[6] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[7] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 644–662. Springer, 2012.

[8] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[9] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[10] Marc Fischlin, Anja Lehmann, and Dominique Schröder. History-free sequential aggregate signatures. In Ivan Visconti and Roberto De Prisco, editors, *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2012.

[11] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2006.

[12] Michael Gerbush, Allison B. Lewko, Adam O'Neill, and Brent Waters. Dual form signatures: An approach for proving security from static assumptions. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 25–42. Springer, 2012.

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

[14] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983.

[15] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Aggregating cl-signatures revisited: Extended functionality and better efficiency. Cryptology ePrint Archive, Report 2012/562, 2012. http://eprint.iacr.org/2012/562.

[16] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Sequential aggregate signatures with short public keys: Design, analysis and implementation studies. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 423–442. Springer, 2013.

[17] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure hibe with short ciphertexts. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 455–479. Springer, 2010.

[18] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485. Springer, 2006.

[19] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2004.

[20] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM Conference on Computer and Communications Security*, pages 245–254. ACM, 2001.

[21] Gregory Neven. Efficient sequential aggregate signed data. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 52–69. Springer, 2008.

[22] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2007.

[23] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[24] Dominique Schröder. How to aggregate the cl signature scheme. In Vijay Atluri and Claudia Díaz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 298–314. Springer, 2011.

[25] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.