Sequential Aggregate Signatures with Short Public Keys: Design, Analysis and Implementation Studies

Kwangsu Lee* Dong Hoon Lee† Moti Yung‡

Abstract

The notion of aggregate signature has been motivated by applications and it enables any user to compress different signatures signed by different signers on different messages into a short signature. Sequential aggregate signature, in turn, is a special kind of aggregate signature that only allows a signer to add his signature into an aggregate signature in sequential order. This latter scheme has applications in diversified settings, such as in reducing bandwidth of a certificate chains, and in secure routing protocols. Lu, Ostrovsky, Sahai, Shacham, and Waters presented the first sequential aggregate signature scheme in the standard (non idealized ROM) model. The size of their public key, however, is quite large (i.e., the number of group elements is proportional to the security parameter), and therefore they suggested as an open problem the construction of such a scheme with short keys. Schröder recently proposed a sequential aggregate signature (SAS) with short public keys using the Camenisch-Lysyanskaya signature scheme, but the security is only proven under an interactive assumption (which is considered a relaxed notion of security). In this paper, we propose the first sequential aggregate signature scheme with short public keys (i.e., a constant number of group elements) in prime order (asymmetric) bilinear groups which is secure under static assumptions in the standard model. Further, our scheme employs constant number of pairing operation per message signing and message verification operation. Technically, we start with a public key signature scheme based on the recent dual system encryption technique of Lewko and Waters. This technique cannot give directly an aggregate signature scheme since, as we observed, additional elements should be published in the public key to support aggregation. Thus, our construction is a careful augmentation technique for the dual system technique to allow it to support a sequential aggregate signature scheme via randomized verification. We further implemented our scheme and conducted a performance study and implementation optimization.

Keywords: Public key signature, Aggregate signature, Sequential aggregate signature, Dual system encryption, Bilinear pairing.

1 Introduction

Aggregate signature is a relatively new type of public key signature which enables any user to combine n signatures signed by different n signers on different n messages into a short signature. The concept of public key aggregate signature (PKAS) was introduced by Boneh, Gentry, Lynn, and Shacham [9], and

*Columbia University, NY, USA. Email: kwangsu@cs.columbia.edu. 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, Seoul, 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, NY, USA. Email: moti@cs.columbia.edu.

they proposed an efficient PKAS scheme in the random oracle model using the bilinear groups. After that, numerous PKAS schemes were proposed using bilinear groups [14, 22, 6, 7, 1, 15] or using trapdoor permutations [24, 3, 25].

One application of aggregate signature is the certificate chains of the public key infrastructure (PKI) [9]. The PKI system has a tree structure, and a certificate for a user consists of a certificate chain from a root node to a leaf node, each node in the chain signing its predecessor. If the signatures in the certificate chain are replaced with a single aggregate signature, then the bandwidth for signatures transfer can be significantly saved. Another application is to the secure routing protocol of the internet protocol [9]. If each router which participates in the routing protocol uses PKAS instead of a public key signature (PKS), then the communication overload of signature transfer can be dramatically reduced. Further, aggregate signatures have other applications such as reducing bandwidth in sensor networks or ad-hoc networks, and in software authentication in the presence of software update [1].

1.1 Previous Methods

Aggregate signature schemes are categorized as *full* aggregate signature, *synchronized* aggregate signature, and *sequential* aggregate signature depending on the type of signature aggregation. They have also been applied to regular signatures in the PKI model, and to ID-based signatures (with trusted key server).

The first type of aggregate signature is *full aggregate signature* which enables any user to freely aggregate different signatures of different signers. This full aggregate signature is the most flexible aggregate signature since it does not require any restriction on the aggregation step (though, restriction may be needed at times for certain applications). However, there is only one full aggregate signature scheme that was proposed by Boneh et al. [9]. Since this scheme is based on the short signature scheme of Boneh et al. [10], the signature length it provides is also very short. However, the security of the scheme is just proven in the idealized random oracle model and the number of pairing operations in the aggregate signature verification algorithm is proportional to the number of signers in the aggregate signature.

The second type of aggregate signature is *synchronized aggregate signature* which enables any user to combine different signatures with the same synchronizing information into a single signature. The synchronized aggregate signature has a demerit which dictates that all signers should share the same synchronizing information (like a time clock or other shared value). Gentry and Ramzan introduced the concept of synchronized aggregate signature, they proposed an identity-based synchronized aggregate signature scheme using bilinear groups, and they proved its security in the random oracle model [14]. We note that identity-based aggregate signature (IBAS) is an ID-based scheme and thus relies on a trusted server knowing all private keys (i.e., its trust structure is different than in regular PKI). However, it also has a notable advantage such that it is not required to retrieve the public keys of signers in the verification algorithm since an identity string plays the role of a public key (the lack of public key is indicated in our comparison table as public key of no size!). Recently, Ahn et al. presented a public key synchronized aggregate signature scheme without relying on random oracles [1].

The third type of aggregate signature is *sequential aggregate signature* (SAS) that enables each signer to aggregate his signature to a previously aggregated signature in a sequential order. The sequential aggregate signature has the obvious limitation of signers being ordered to aggregate their signatures in contrast to the full aggregate signature and the synchronized aggregate signature. However, it has an advantage such that it is not required to share synchronized information among signers in contrast to the synchronized aggregate signature, and many natural applications lead themselves to this setting. The concept of sequential aggregate signature was introduced by Lysyanskaya et al., and they proposed a public key sequential aggregate signature scheme using the certified trapdoor permutations in the random oracle model [24]. Boldyreva et

Table 1: Comparison of aggregate signature schemes

Scheme	Type	ROM	KOSK	PK Size	AS Size	Sign Time	Verify Time	Assumption
BGLS [9]	Full	Yes	No	$1k_p$	$1k_p$	1E	lP	CDH
GR [14]	IB, Sync	Yes	No	–	$2k_p+\lambda$	3E	3P + lE	CDH
AGH [1]	Sync	Yes	Yes	$1k_p$	$2k_p+32$	6E	4P + lE	CDH
AGH [1]	Sync	No	Yes	$1k_p$	$2k_p+32$	10E	8P + lE	CDH
LMRS [24]	Seq	Yes	No	$1k_f$	$1k_f$	lE	lE	cert TDP
Neven [25]	Seq	Yes	No	$1k_f$	$1k_f+2\lambda$	1E + 2lM	2lM	uncert CFP
BGOY [7]	IB, Seq	Yes	No	–	$3k_p$	4P + lE	4P + lE	Interactive
GLOW [15]	IB, Seq	Yes	No	–	$5k_f$	10P + 2lE	10P + 2lE	Static
LOSSW [22]	Seq	No	Yes	$2\lambda k_p$	$2k_p$	2P + 4λlM	2P + 2λlM	CDH
Schröder [27]	Seq	No	Yes	$2k_p$	$4k_p$	lP + 2lE	lP + lE	Interactive
Ours	Seq	No	Yes	$11k_p$	$8k_p$	8P + 5lE	8P + 4lE	Static

ROM = random oracle model, KOSK = certified-key model, IB = identity based

λ = security parameter, k_p, k_f = the bit size of element for pairing and factoring, l = the number of signers

P = pairing computation, E = exponentiation, M = multiplication

al. presented an identity-based sequential aggregate signature scheme in the random oracle model using an interactive assumption [6], but it was shown that their construction is not secure by Hwang et al. [17]. After that, Boldyreva et al. proposed a new identity-based sequential aggregate signature by modifying their previous construction and proved its security in the generic group model [7]. Recently, Gerbush et al. showed that the modified IBAS scheme of Boldyreva et al. is secure under static assumptions using the dual form signatures framework [15]. The first sequential aggregate signature scheme without the random oracle idealization was proposed by Lu et al. [22]. They converted the PKS scheme of Waters [28] to the PKAS scheme, and proved its security under the well known CDH assumption. However, the scheme of Lu et al. has a demerit since the number of group elements in the public key is proportional to the security parameter (for a security of 2^{80} they need 160 elements or about 80 elements in a larger group); they left as an open question to design a scheme with shorter public key. Schröder proposed a PKAS scheme with short public keys relying on the Camenisch-Lysyanskaya signature scheme [27], however the scheme's security is proven under an interactive assumption (which typically, is a relaxation used when designs based on static assumptions are hard to find).[1] Therefore, the construction of sequential aggregate signature scheme with short public keys without relaxations like random oracles or an interactive assumptions was left as an open question.

1.2 Our Contributions

Challenged by the above question, the motivation of our research is to construct an efficient sequential aggregate signature scheme secure in the standard model (i.e., without employing assumptions like random oracle or interactive assumptions as part of the proof) with short public keys (e.g., constant number of group

[1] Gerbush et al. showed that a modified Camenisch-Lysyanskaya signature scheme in composite order groups is secure under static assumptions [15]. However, it is unclear whether the construction of Schröder can be directly applied to this modified Camenisch-Lysyanskaya signature scheme. The reason is that aggregating $\mathbb{G}_{p_1}$ and $\mathbb{G}_{p_2}$ subgroups is hard and a private key element $g_{2,3} \in \mathbb{G}_{p_2 p_3}$ can not be generated by the key generation algorithm of an aggregate signature scheme. Additionally, our work and findings are independent from the work of Gerbush et al.

elements). To achieve this goal, we use the public key signature scheme derived from the identity-based encryption (IBE) scheme that adopts the innovative dual system encryption techniques of Waters [29, 21]. That is, an IBE scheme is first converted to a PKS scheme by the clever observation of Naor [8]. The PKS schemes that adopt the dual system encryption techniques are the scheme of Waters [29] which includes a random tag in a signature and the scheme of Lewko and Waters [21] which does not include a random tag in a signature. The scheme of Waters is not appropriate to aggregate signature since the random tags in signatures cannot be compressed into a single value. The scheme of Lewko and Waters in composite order groups is easily converted to an aggregate signature scheme if the element of $\mathbb{G}_{p_3}$ is moved from a private key to a public key, but it is inefficient because of composite order groups.[2] Therefore, we start the construction from the IBE scheme in prime order (asymmetric) bilinear groups of Lewko and Waters. However, this PKS scheme which is directly derived from the IBE scheme of Lewko and Waters is not easily converted to a sequential aggregate signature scheme (as far as we see). The reason is that we need a PKS scheme that supports multi-user setting and public re-randomization to construct a SAS scheme by using the randomness reuse technique of Lu et al. [22], but this PKS scheme does not support these two properties.

Here we first construct a PKS scheme in prime order (asymmetric) bilinear groups which supports multi-user seting and public re-randomization by modifying the PKS scheme of Lewko and Waters, and we prove its security using the dual system encryption technique. Next, we convert the modified PKS scheme to a SAS scheme with short public keys by using the randomness reuse technique of Lu et al. [22], and we prove its security without random oracles and based on the traditional static assumptions. Our security proof crucially relies on the fact that we add additional randomization elements to the SAS verification algorithm, so that we can expand these elements to a semi-functional space; this allows us to introduce in the SAS scheme public-key elements used in aggregation. Note that Table 1 gives a comparison of past schemes to ours. Finally, to support our claim of efficiency, we implemented our SAS scheme using the PBC library (code available from authors [19]) and we measured the performance of the scheme. Additionally, as part of the implementation we provide a computational preprocessing method which improves the amortized performance of our scheme.

1.3 Additional Related Work

There are some works on aggregate signature schemes which allow signers to communicate with each other or schemes which compress only partial elements of a signature in the aggregate algorithm [4, 2, 16, 11]. Generally, communication resources of computer systems are very expensive compared to the computation resources. Thus, it is preferred to perform several expensive computational operations instead of a single communication exchange. Additionally, a signature scheme with added communications does not correspond to a pure public key signature schemes, but corresponds more to a multi-party protocol. In addition, signature schemes which compress just partial elements of signatures cannot be an aggregate signature since the total size of signatures is still proportional to the number of signers.

Another research area related to aggregate signature is multi-signature [5, 22]. Multi-signature is a special type of aggregate signature in which all signers generate signatures on the same message, and then any user can combine these signature to a single signature. Aggregate message authentication code (AMAC) is the symmetric key analogue of aggregate signature: Katz and Lindell introduced the concept of AMAC

[2] We can safely move the element of $\mathbb{G}_{p_3}$ from a private to a public key since it is always given in assumptions. Lewko obtained a prime order IBE scheme by translating the Lewko-Waters composite order IBE scheme using the dual pairing vector spaces [20]. One may consider to construct an aggregate signature scheme using this IBE scheme. However, it is not easy to aggregate individual signatures since the dual orthonormal basis vectors of each users are randomly generated.

and showed that it is possible to construct AMAC schemes based on any message authentication code schemes [18].

2 Preliminaries

We first define public key signature and sequential aggregate signature, and then give the definition of their correctness and security.

2.1 Public Key Signature

A public key signature (PKS) scheme consists of three PPT algorithms **KeyGen**, **Sign**, and **Verify**, which are defined as follows: The key generation algorithm $\mathbf{KeyGen}(1^\lambda)$ takes as input the security parameters 1^λ, and outputs a public key PK and a private key SK. The signing algorithm $\mathbf{Sign}(M, SK)$ takes as input a message M and a private key SK, and outputs a signature σ. The verification algorithm $\mathbf{Verify}(\sigma, M, PK)$ 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 $\mathbf{Verify}(\mathbf{Sign}(M, SK), M, PK) = 1$. We can relax this notion to require that the verification is correct with overwhelming probability over all the randomness of the experiment.

The security 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 $\mathcal{A}$: $\mathcal{C}$ first generates a key pair (PK, SK) by running **KeyGen**, and gives PK to $\mathcal{A}$. 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. 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) $\mathbf{Verify}(\sigma^*, M^*, PK) = 1$ 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).

2.2 Sequential Aggregate Signature

A sequential aggregate signature (SAS) scheme consists of four PPT algorithms **Setup**, **KeyGen**, **AggSign**, and **AggVerify**, which are defined as follows: The setup algorithm $\mathbf{Setup}(1^\lambda)$ takes as input a security parameter 1^λ and outputs public parameters PP. The key generation algorithm $\mathbf{KeyGen}(PP)$ takes as input the public parameters PP, and outputs a public key PK and a private key SK. The aggregate signing algorithm $\mathbf{AggSign}(AS', \mathbf{M}, \mathbf{PK}, M, SK)$ 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. The aggregate verification algorithm $\mathbf{AggVerify}(AS, \mathbf{M}, \mathbf{PK})$ 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 $\mathbf{AggVerify}(\mathbf{AggSign}(AS', \mathbf{M}', \mathbf{PK}', M, SK), \mathbf{M}'||M, \mathbf{PK}'||PK) = 1$ where AS' is a valid aggregate-so-far signature on messages $\mathbf{M}'$ under public keys $\mathbf{PK}'$.

The security 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 $\mathcal{A}$:

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 $\mathcal{A}$.

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.

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.

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 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 (for large enough security parameter) in the above experiment.

2.3 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 $\hat{\mathbb{G}}$.

2.4 Complexity Assumptions

We employ three static assumptions in prime order bilinear groups. Assumptions 1 and 3 have been used extensively, while Assumption 2 was introduced by Lewko and Waters [21].

Assumption 1 (Symmetric eXternal 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

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

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

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 Aggregate Signature

We construct a SAS scheme in prime order (asymmetric) bilinear groups and prove its existential unforgeability under a chosen message attack. The main idea is to modify a PKS scheme to support multi-user setting and signature aggregation by using the "randomness reuse" technique of Lu et al. [22]. To support multi-user setting, it is required for all users to share common elements in the public parameters. To use the randomness reuse technique, it is crucial for a signer to publicly re-randomize a sequential aggregate signature to prevent a forgery attack. Thus we need a PKS scheme with short public key that supports "multi-user setting" and "public re-randomization".

Before we present a SAS scheme, we first construct a PKS scheme with short public key that will be augmented to support multi-user setting and public re-randomization. One method to build a PKS scheme is to use the observation of Naor [8] that private keys of fully secure IBE are easily converted to signatures of PKS. Thus we can convert the prime order IBE scheme of Lewko and Waters [21] to a prime order PKS scheme. However, this directly converted PKS scheme does not support multi-user setting and public re-randomization since it needs to publish additional public key components: Specifically, we need to publish an element g for multi-user setting and elements u, h for public re-randomization. Note that $\hat{g}, \hat{u}, \hat{h}$ are already in the public key, but g, u, h are not. One may try to publish g, u, h in the public key. The technical difficulty arising in this case is that the simulator of the security proof can easily distinguish the changes of the verification algorithm that checks the validity of the forged signature from the normal verification algorithm to the semi-functional one, without using an adversary.

To solve this problem, we devise a method that allows a PKS scheme to safely publish elements g, u, h in the public key for multi-user setting and public re-randomization. The main idea is to additionally randomize the verification components using $\hat{v}, \hat{v}^{v_3}, \hat{v}^{-\pi}$ in the verification algorithm. If a valid signature is given in the verification algorithm, then the additionally added randomization elements $\hat{v}, \hat{v}^{v_3}, \hat{v}^{-\pi}$ are canceled. Otherwise, the added randomization components prevent the verification of an invalid signature. Therefore, the simulator of the security proof cannot distinguish the changes of the verification algorithm even if g, u, h are published, since the additional elements $\hat{v}, \hat{v}^{v_3}, \hat{v}^{-\pi}$ prevent the signature verification.

3.1 Our PKS Scheme

The 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 $\hat{g}, \hat{v} \in \hat{\mathbb{G}}$. Next, it chooses random exponents $\nu_1, \nu_2, \nu_3, \phi_1, \phi_2, \phi_3 \in \mathbb{Z}_p$ and sets $\tau = \phi_1 + \nu_1\phi_2 + \nu_2\phi_3, \pi = \phi_2 + \nu_3\phi_3$. It selects random exponents $\alpha, x, y \in \mathbb{Z}_p$ and sets $u = g^x, h = g^y, \hat{u} = \hat{g}^x, \hat{h} = \hat{g}^y$. It outputs a private key $SK = (\alpha, x, y)$ and a public key as

$$PK = \big(g, u, h,\ w_1 = w^{\phi_1}, w_2 = w^{\phi_2}, w_3 = w^{\phi_3}, w,\ \hat{g}, \hat{g}^{\nu_1}, \hat{g}^{\nu_2}, \hat{g}^{-\tau},$$
$$\hat{u}, \hat{u}^{\nu_1}, \hat{u}^{\nu_2}, \hat{u}^{-\tau},\ \hat{h}, \hat{h}^{\nu_1}, \hat{h}^{\nu_2}, \hat{h}^{-\tau},\ \hat{v}, \hat{v}^{\nu_3}, \hat{v}^{-\pi},\ \Omega = e(g,\hat{g})^\alpha \big).$$

PKS.Sign(M, SK): This algorithm takes as input a message $M \in \{0,1\}^k$ where $k < \lambda$ 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

$$\sigma = \big(W_{1,1} = g^\alpha (u^M h)^r w_1^{c_1}, W_{1,2} = w_2^{c_1}, W_{1,3} = w_3^{c_1}, W_{1,4} = w^{c_1},$$
$$W_{2,1} = g^r w_1^{c_2}, W_{2,2} = w_2^{c_2}, W_{2,3} = w_3^{c_2}, W_{2,4} = w^{c_2} \big).$$

PKS.Verify(σ, M, PK): This algorithm takes as input a signature σ on a message $M \in \{0,1\}^k$ under a public key PK. It first chooses random exponents $t, s_1, s_2 \in \mathbb{Z}_p$ and computes verification components as

$$V_{1,1} = \hat{g}^t, V_{1,2} = (\hat{g}^{\nu_1})^t \hat{v}^{s_1}, V_{1,3} = (\hat{g}^{\nu_2})^t (\hat{v}^{\nu_3})^{s_1}, V_{1,4} = (\hat{g}^{-\tau})^t (\hat{v}^{-\pi})^{s_1},$$
$$V_{2,1} = (\hat{u}^M \hat{h})^t, V_{2,2} = ((\hat{u}^{\nu_1})^M \hat{h}^{\nu_1})^t \hat{v}^{s_2}, V_{2,3} = ((\hat{u}^{\nu_2})^M \hat{h}^{\nu_2})^t (\hat{v}^{\nu_3})^{s_2}, V_{2,4} = ((\hat{u}^{-\tau})^M \hat{h}^{-\tau})^t (\hat{v}^{-\pi})^{s_2}.$$

Next, it verifies that $\prod_{i=1}^4 e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^4 e(W_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$. If this equation holds, then it outputs 1. Otherwise, it outputs 0.

We first note that the inner product of $(\phi_1, \phi_2, \phi_3, 1)$ and $(1, \nu_1, \nu_2, -\tau)$ is zero since $\tau = \phi_1 + \nu_1\phi_2 + \nu_2\phi_3$, and the inner product of $(\phi_1, \phi_2, \phi_3, 1)$ and $(0, 1, \nu_3, -\pi)$ is zero since $\pi = \phi_2 + \nu_3\phi_3$. Using these facts, the correctness requirement of the above PKS scheme is easily verified as

$$\prod_{i=1}^4 e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^4 e(W_{2,i}, V_{2,i})^{-1} = e(g^\alpha (u^M h)^r, \hat{g}^t) \cdot e(g^r, (\hat{u}^M \hat{h})^t)^{-1} = \Omega^t.$$

Theorem 3.1. *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 $\mathcal{A}$.*

The proof of this theorem is given in Section 4.1.

3.2 Our 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 $\hat{g}, \hat{v} \in \hat{\mathbb{G}}$. Next, it chooses random exponents $\nu_1, \nu_2, \nu_3, \phi_1, \phi_2, \phi_3 \in \mathbb{Z}_p$ and sets $\tau = \phi_1 + \nu_1\phi_2 + \nu_2\phi_3, \pi = \phi_2 + \nu_3\phi_3$. It publishes public parameters as

$$PP = \big(g,\ w_1 = w^{\phi_1}, w_2 = w^{\phi_2}, w_3 = w^{\phi_3}, w,\ \hat{g}, \hat{g}^{\nu_1}, \hat{g}^{\nu_2}, \hat{g}^{-\tau},\ \hat{v}, \hat{v}^{\nu_3}, \hat{v}^{-\pi} \big).$$

SAS.KeyGen(PP): This algorithm takes as input the public parameters PP. It selects random exponents $\alpha, x, y \in \mathbb{Z}_p$ and computes $u = g^x, h = g^y, \hat{u} = \hat{g}^x, \hat{u}^{\nu_1} = (\hat{g}^{\nu_1})^x, \hat{u}^{\nu_2} = (\hat{g}^{\nu_2})^x, \hat{u}^{-\tau} = (\hat{g}^{-\tau})^x, \hat{h} = \hat{g}^y, \hat{h}^{\nu_1} = (\hat{g}^{\nu_1})^y, \hat{h}^{\nu_2} = (\hat{g}^{\nu_2})^y, \hat{h}^{-\tau} = (\hat{g}^{-\tau})^y$. It outputs a private key $SK = (\alpha, x, y)$ and a public key as

$$PK = \big(u, h,\ \hat{u}, \hat{u}^{\nu_1}, \hat{u}^{\nu_2}, \hat{u}^{-\tau},\ \hat{h}, \hat{h}^{\nu_1}, \hat{h}^{\nu_2}, \hat{h}^{-\tau},\ \Omega = e(g,\hat{g})^\alpha \big).$$

SAS.AggSign($AS', \mathbf{M}', \mathbf{PK}', M, SK$): This algorithm takes as input an aggregate-so-far $AS' = (S'_{1,1}, \ldots, S'_{2,4})$ on messages $\mathbf{M}' = (M_1, \ldots, M_{l-1})$ under public keys $\mathbf{PK}' = (PK_1, \ldots, PK_{l-1})$ where $PK_i = (u_i, h_i, \ldots, \Omega_i)$, a message $M \in \{0,1\}^k$ where $k < \lambda$, a private key $SK = (\alpha, x, y)$ with $PK = (u, h, \ldots, \Omega)$ and PP. It first checks the validity of AS' by calling **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 selects random exponents $r, c_1, c_2 \in \mathbb{Z}_p$ and outputs an aggregate signature as

$$\begin{aligned} AS = \big(& S_{1,1} = S'_{1,1} g^\alpha (S'_{2,1})^{xM+y} \cdot \prod_{i=1}^{l-1} (u_i^{M_i} h_i)^r (u^M h)^r w_1^{c_1},\ S_{1,2} = S'_{1,2} (S'_{2,2})^{xM+y} \cdot w_2^{c_1}, \\ & S_{1,3} = S'_{1,3} (S'_{2,3})^{xM+y} \cdot w_3^{c_1},\ S_{1,4} = S'_{1,4} (S'_{2,4})^{xM+y} \cdot w^{c_1}, \\ & S_{2,1} = S'_{2,1} \cdot g^r w_1^{c_2},\ S_{2,2} = S'_{2,2} \cdot w_2^{c_2},\ S_{2,3} = S'_{2,3} \cdot w_3^{c_2},\ S_{2,4} = S'_{2,4} \cdot w^{c_2} \big). \end{aligned}$$

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 $PK_i = (u_i, h_i, \ldots, \Omega_i)$. 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 $S_1 = S_2 = 1$, 0 otherwise. It chooses random exponents $t, s_1, s_2 \in \mathbb{Z}_p$ and computes verification components as

$$\begin{aligned} & C_{1,1} = \hat{g}^t,\ C_{1,2} = (\hat{g}^{\nu_1})^t \hat{v}^{s_1},\ C_{1,3} = (\hat{g}^{\nu_2})^t (\hat{v}^{\nu_3})^{s_1},\ C_{1,4} = (\hat{g}^{-\tau})^t (\hat{v}^{-\pi})^{s_1}, \\ & C_{2,1} = \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^t,\ C_{2,2} = \prod_{i=1}^{l} ((\hat{u}_i^{\nu_1})^{M_i} \hat{h}_i^{\nu_1})^t \hat{v}^{s_2}, \\ & C_{2,3} = \prod_{i=1}^{l} ((\hat{u}_i^{\nu_2})^{M_i} \hat{h}_i^{\nu_2})^t (\hat{v}^{\nu_3})^{s_2},\ C_{2,4} = \prod_{i=1}^{l} ((\hat{u}_i^{-\tau})^{M_i} \hat{h}_i^{-\tau})^t (\hat{v}^{-\pi})^{s_2}. \end{aligned}$$

Next, it verifies that $\prod_{i=1}^4 e(S_{1,i}, C_{1,i}) \cdot \prod_{i=1}^4 e(S_{2,i}, C_{2,i})^{-1} \stackrel{?}{=} \prod_{i=1}^l \Omega_i^t$. If this equation holds, then it outputs 1. Otherwise, it outputs 0.

The aggregate signature AS is a valid sequential aggregate signature on messages $\mathbf{M}'||M$ under public keys $\mathbf{PK}'||PK$ with randomness $\tilde{r} = r' + r,\ \tilde{c}_1 = c'_1 + c'_2(xM+y) + c_1,\ \tilde{c}_2 = c'_2 + c_2$ where r', c'_1, c'_2 are random values in AS'. The sequential aggregate signature has the following form

$$\begin{aligned} & S_{1,1} = \prod_{i=1}^{l} g^{\alpha_i} \prod_{i=1}^{l} (u_i^{M_i} h_i)^{\tilde{r}} w_1^{\tilde{c}_1},\ S_{1,2} = w_2^{\tilde{c}_1},\ S_{1,3} = w_3^{\tilde{c}_1},\ S_{1,4} = w^{\tilde{c}_1}, \\ & S_{2,1} = g^{\tilde{r}} w_1^{\tilde{c}_2},\ S_{2,2} = w_2^{\tilde{c}_2},\ S_{2,3} = w_3^{\tilde{c}_2},\ S_{2,4} = w^{\tilde{c}_2}. \end{aligned}$$

Theorem 3.2. *The above SAS scheme is existentially unforgeable under a chosen message attack if the PKS scheme is existentially unforgeable under a chosen message attack. 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* $\boldsymbol{Adv}_{\mathcal{A}}^{SAS}(\lambda) \leq \boldsymbol{Adv}_{\mathcal{B}}^{PKS}(\lambda)$.

The proof of this theorem is given in Section 4.2.

3.3 Extensions

In this section, we discuss various extensions of our SAS scheme.

Multiple Messages. The SAS scheme of this paper only allows a signer to sign once in the aggregate algorithm. To support multiple signing per one signer, we can use the method of Lu et al. [22]. The basic idea of Lu et al. is to apply a collision resistant hash function H to a message M before performing the signing algorithm. If a signer wants to add a signature on a message M_2 into the aggregate signature, he first removes his previous signature on $H(M_1)$ from the aggregate signature using his private key, and then he adds the new signature on the $H(M_1||M_2)$ to the aggregate signature.

Multi-signatures. The SAS scheme of this paper can be easily converted to a multi-signature scheme. In case of multi-signature, some elements of public keys in SAS can be moved to the public parameters since multi-signature only allows signers to sign on the same message. Compared to the multi-signature scheme of Lu et al. [22], our multi-signature scheme has short size public parameters.

4 Security Analysis

In this section, we analyze the security of the basic PKS scheme and our SAS scheme.

4.1 Proof of Theorem 3.1

To prove the security of our PKS scheme, we use the dual system encryption technique of Lewko and Waters [21]. We describe a semi-functional signing algorithm and a semi-functional verification algorithm. They are not used in a real system, rather they are used in the security proof. When comparing our proof to that of Lewko and Waters, we employ a different assumption since we have published additional elements g, u, h used in aggregation (in fact, direct adaptation of the earlier technique will break the assumption and thus the proof). A crucial idea in our proof is that we have added elements $\hat{v}, \hat{v}^{v_3}, \hat{v}^{-\pi}$ in the public key which are used in randomization of the verification algorithm. In the security proof when moving from normal to semi-functional verification, it is the randomization elements $\hat{v}, \hat{v}^{v_3}, \hat{v}^{-\pi}$ which are expanded to the semi-functional space; this enables deriving semi-functional verification as part of the security proof under our assumption, without being affected by the publication of the additional public key elements used for aggregation.

For the semi-functional signing and verification 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 $(W'_{1,1}, \ldots, W'_{2,4})$ 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

$$\sigma = \big(W_{1,1} = W'_{1,1}(f^{v_1 v_3 - v_2})^{s_k z_k},\ W_{1,2} = W'_{1,2}(f^{-v_3})^{s_k z_k},\ W_{1,3} = W'_{1,3} f^{s_k z_k},\ W_{1,4} = W'_{1,4},$$
$$W_{2,1} = W'_{2,1}(f^{v_1 v_3 - v_2})^{s_k},\ W_{2,2} = W'_{2,2}(f^{-v_3})^{s_k},\ W_{2,3} = W'_{2,3} f^{s_k},\ W_{2,4} = W'_{2,4} \big).$$

PKS.VerifySF. The semi-functional verification algorithm first creates a normal verification components using the public key. Let $(V'_{1,1}, \ldots, V'_{2,4})$ be the normal verification components with random exponents $t, s_1, s_2 \in \mathbb{Z}_p$. It chooses random exponents $s_c, z_c \in \mathbb{Z}_p$ and computes semi-functional verification components as

$$V_{1,1} = V'_{1,1},\ V_{1,2} = V'_{1,2},\ V_{1,3} = V'_{1,3}\hat{f}^{s_c},\ V_{1,4} = V'_{1,4}(\hat{f}^{-\phi_3})^{s_c},$$
$$V_{2,1} = V'_{2,1},\ V_{2,2} = V'_{2,2},\ V_{2,3} = V'_{2,3}\hat{f}^{s_c z_c},\ V_{2,4} = V'_{2,4}(\hat{f}^{-\phi_3})^{s_c z_c}.$$

Next, it verifies that $\prod_{i=1}^{4} e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{4} e(W_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$. If this equation holds, then it outputs 1. Otherwise, it outputs 0.

Note that if the semi-functional verification algorithm verifies a semi-functional signature, then the left part of the above verification equation contains an additional random element $e(f, \hat{f})^{s_k s_c (z_k - z_c)}$. 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 $\mathcal{A}$.

Game $\mathbf{G}_1$. We first modify the original game to a new 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 $\mathcal{A}$.

Game $\mathbf{G}_2$. Next, we change $\mathbf{G}_1$ to a new 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 $\mathbf{G}_3$. Finally, we define a new game $\mathbf{G}_3$. This game differs from $\mathbf{G}_2$ in that the challenger always rejects the forged signature of $\mathcal{A}$. Therefore, the advantage of this game is zero since $\mathcal{A}$ cannot win this game.

For the security proof, we show the indistinguishability of each hybrid games. We informally describe the meaning of each indistinguishability as follows:

- Indistinguishability of $\mathbf{G}_0$ and $\mathbf{G}_1$: This property shows that $\mathcal{A}$ cannot forge a semi-functional signature if it is only given normal signatures. That is, if $\mathcal{A}$ forges a semi-functional signature, then it can distinguish $\mathbf{G}_0$ from $\mathbf{G}_1$.

- Indistinguishability of $\mathbf{G}_1$ and $\mathbf{G}_2$: This property shows that the probability of $\mathcal{A}$ to forge a normal signature is almost the same when the signatures given to the adversary are changed from normal type to semi-functional type. That is, if the probability of $\mathcal{A}$ to forge a normal signature is different in $\mathbf{G}_1$ and $\mathbf{G}_2$, then $\mathcal{A}$ can distinguish two games.

- Indistinguishability of $\mathbf{G}_2$ and $\mathbf{G}_3$: This property shows that $\mathcal{A}$ cannot forge a normal signature if it is only given semi-functional signatures. That is, if $\mathcal{A}$ forges a normal signature, then it can distinguish $\mathbf{G}_2$ from $\mathbf{G}_3$.

The security (unforgeability) of our PKS scheme follows from a hybrid argument. We first consider an adversary $\mathcal{A}$ to attack our PKS scheme in the original security game $\mathbf{G}_0$. By the indistinguishability of $\mathbf{G}_0$ and $\mathbf{G}_1$, we have that $\mathcal{A}$ can forge a normal signature with a non-negligible ε probability, but it can forge a semi-functional signature with only a negligible probability. Now we should show that the ε probability of $\mathcal{A}$ to forge a normal signature is also negligible. By the indistinguishability of $\mathbf{G}_1$ and $\mathbf{G}_2$, we have that the ε probability of $\mathcal{A}$ to forge a normal signature is almost the same when the signatures given to $\mathcal{A}$ are changed from normal type to semi-functional type. Finally, by the indistinguishability of $\mathbf{G}_2$ and $\mathbf{G}_3$, we have that $\mathcal{A}$ can forge a normal signature with only a negligible probability. Summing up, we obtain that the probability of $\mathcal{A}$ to forge a semi-functional signature is negligible (from the indistinguishability of $\mathbf{G}_0$ and $\mathbf{G}_1$) and the probability of $\mathcal{A}$ to forge a normal signature is also negligible (from the indistinguishability of $\mathbf{G}_2$ and $\mathbf{G}_3$).

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 4.1. *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. Before proving this lemma, we introduce Assumption 1-A as follows: Let $(p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e)$ be a description of the asymmetric bilinear group of prime order p. Let $k,\hat{k}$ be generators of $\mathbb{G},\hat{\mathbb{G}}$ respectively. Assumption 1-A is stated as following: given a challenge tuple $D = ((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),k,\hat{k}^a,\hat{k}^{d_1},\hat{k}^{d_2})$ and $T = (A_1,A_2)$, it is hard to decide whether $T = (\hat{k}^{ad_1},\hat{k}^{ad_2})$ or $T = (\hat{k}^{d_3},\hat{k}^{d_4})$ with random choices of $a,d_1,d_2,d_3,d_4 \in \mathbb{Z}_p$. It is easy to prove by simple hybrid arguments that if there exists an adversary that breaks Assumption 1-A, then it can break Assumption 1. Alternatively, we can tightly prove the reduction using the random self-reducibility of the Decisional Diffie-Hellman assumption.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_0$ and $\mathbf{G}_1$ with non-negligible advantage. Simulator $\mathcal{B}_1$ that solves Assumption 1-A using $\mathcal{A}$ is given: a challenge tuple $D = ((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e)$, $k,\hat{k},\hat{k}^a,\hat{k}^{d_1},\hat{k}^{d_2})$ and $T = (A_1,A_2)$ where $T = T_0 = (A_1^0,A_2^0) = (\hat{k}^{ad_1},\hat{k}^{ad_2})$ or $T = T_1 = (A_1^1,A_2^1) = (\hat{k}^{ad_1+d_3},\hat{k}^{ad_2+d_4})$. Then $\mathcal{B}_1$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}_1$ first chooses random exponents $\nu_1,\nu_2,\phi_1,\phi_2,\phi_3 \in \mathbb{Z}_p$, then it sets $\tau = \phi_1 + \nu_1\phi_2 + \nu_2\phi_3$. It selects random exponents $\alpha,x,y,y_g,y_v,y_w \in \mathbb{Z}_p$ and sets $g = k^{y_g}, u =$

$g^x, h = g^y, w = k^{y_w}, \hat{g} = \hat{k}^{y_g}, \hat{u} = \hat{g}^x, \hat{h} = \hat{g}^y$. It implicitly sets $v_3 = a, \pi = \phi_2 + a\phi_3$ and publishes a public key as

$$g, u, h,\ w_1 = w^{\phi_1}, w_2 = w^{\phi_2}, w_3 = w^{\phi_3}, w,\ \hat{g}, \hat{g}^{v_1}, \hat{g}^{v_2}, \hat{g}^{-\tau},\ \hat{u}, \hat{u}^{v_1}, \hat{u}^{v_2}, \hat{u}^{-\tau},$$
$$\hat{h}, \hat{h}^{v_1}, \hat{h}^{v_2}, \hat{h}^{-\tau},\ \hat{v} = \hat{k}^{y_v}, \hat{v}^{v_3} = (\hat{k}^a)^{y_v}, \hat{v}^{-\pi} = \hat{k}^{-y_v\phi_2}(\hat{k}^a)^{-y_v\phi_3},\ \Omega = e(g,\hat{g})^\alpha.$$

It sets a private key as (α, x, y). 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, $\mathcal{B}_1$ creates a normal signature by calling **PKS.Sign** since it knows the private key. Note that it cannot create a semi-functional signature since it does not know k^a. Finally, $\mathcal{A}$ outputs a forged signature $\sigma^* = (W^*_{1,1}, \ldots, W^*_{2,4})$ 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 $s_1 = d_1,\ s_2 = d_2$ as

$$V_{1,1} = \hat{g}^t,\ V_{1,2} = (\hat{g}^{v_1})^t(\hat{k}^{d_1})^{y_v},\ V_{1,3} = (\hat{g}^{v_2})^t(A_1)^{y_v},\ V_{1,4} = (\hat{g}^{-\tau})^t(\hat{k}^{d_1})^{-y_v\phi_2}(A_1)^{-y_v\phi_3},$$
$$V_{2,1} = (\hat{u}^{M^*}\hat{h})^t,\ V_{2,2} = ((\hat{u}^{v_1})^{M^*}\hat{h}^{v_1})^t(\hat{k}^{d_2})^{y_v},\ V_{2,3} = ((\hat{u}^{v_2})^{M^*}\hat{h}^{v_2})^t(A_2)^{y_v},$$
$$V_{2,4} = ((\hat{u}^{-\tau})^{M^*}\hat{h}^{-\tau})^t(\hat{k}^{d_2})^{-y_v\phi_2}(A_2)^{-y_v\phi_3}.$$

Next, it verifies that $\prod_{i=1}^4 e(W^*_{1,i}, V_{1,i}) \cdot \prod_{i=1}^4 e(W^*_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$. If this equation holds, then it outputs 0. Otherwise, it outputs 1.

To finish this proof, we now show that the distribution of the simulation is correct. We first show that the distribution using $D, T_0 = (A^0_1, A^0_2) = (\hat{k}^{ad_1}, \hat{k}^{ad_2})$ is the same as $\mathbf{G}_0$. The public key is correctly distributed since the random blinding values y_g, y_w, y_v are used. The signatures is correctly distributed since it uses the signing algorithm. The verification components are correctly distributed as

$$V_{1,3} = (\hat{g}^{v_2})^t(\hat{v}^{v_3})^{s_1} = (\hat{g}^{v_2})^t(\hat{k}^{y_v a})^{d_1} = (\hat{g}^{v_2})^t(A^0_1)^{y_v},$$
$$V_{1,4} = (\hat{g}^{-\tau})^t(\hat{v}^{-\pi})^{s_1} = (\hat{g}^{-\tau})^t(\hat{k}^{-y_v(\phi_2+a\phi_3)})^{d_1} = (\hat{g}^{-\tau})^t(\hat{k}^{d_1})^{-y_v\phi_2}(A^0_1)^{-y_v\phi_3},$$
$$V_{2,3} = ((\hat{u}^{v_2})^{M^*}\hat{h}^{v_2})^t(\hat{v}^{v_3})^{s_2} = ((\hat{u}^{v_2})^{M^*}\hat{h}^{v_2})^t(\hat{k}^{y_v a})^{d_2} = ((\hat{u}^{v_2})^{M^*}\hat{h}^{v_2})^t(A^0_2)^{y_v}$$
$$V_{2,4} = ((\hat{u}^{-\tau})^{M^*}\hat{h}^{-\tau})^t(\hat{v}^{-\pi})^{s_2} = ((\hat{u}^{-\tau})^{M^*}\hat{h}^{-\tau})^t(\hat{k}^{-y_v(\phi_2+a\phi_3)})^{d_2}$$
$$= ((\hat{u}^{-\tau})^{M^*}\hat{h}^{-\tau})^t(\hat{k}^{d_2})^{-y_v\phi_2}(A^0_2)^{-y_v\phi_3}.$$

We next show that the distribution of the simulation using $D, T_1 = (A^1_1, A^1_2) = (\hat{k}^{ad_1+d_3}, \hat{k}^{ad_2+d_4})$ 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 $T_0 = (A^0_1, A^0_2)$ and $T_1 = (A^1_1, A^1_2)$ is that $T_1 = (A^1_1, A^1_2)$ additionally has $(\hat{k}^{d_3}, \hat{k}^{d_4})$. Thus $V_{1,3}, V_{1,4}, V_{2,3}, V_{2,4}$ that have $T = (A_1, A_2)$ in the simulation additionally have $(\hat{k}^{d_3})^{y_v}, (\hat{k}^{d_3})^{-y_v\phi_3}, (\hat{k}^{d_4})^{y_v}, (\hat{k}^{d_4})^{-y_v\phi_3}$ respectively. If we implicitly set $s_c = y_v d_3,\ z_c = d_4/d_3$, then the verification components for the forged signature are semi-functional since d_3, d_4 are randomly chosen. We obtain $\Pr[\mathcal{B}_1(D, T_0) = 0] = \mathbf{Adv}^{G_0}_{\mathcal{A}}$ and $\Pr[\mathcal{B}_1(D, T_1) = 0] = \mathbf{Adv}^{G_1}_{\mathcal{A}}$ from the above analysis. Thus, we can easily derive the advantage of $\mathcal{B}_1$ as

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

This completes our proof. □

Lemma 4.2. *If Assumption 2 holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ with non-negligible advantage. That is, for any adversary $\mathcal{A}$, there exists a PPT algorithm $\mathcal{B}_2$ such that $\left|\mathbf{Adv}^{G_{1,k-1}}_{\mathcal{A}} - \mathbf{Adv}^{G_{1,k}}_{\mathcal{A}}\right| = \mathbf{Adv}^{A2}_{\mathcal{B}_2}(\lambda)$.*

Proof. 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 $\nu_1, \nu_2, \nu_3, y_\tau, \pi, A, B, \alpha, y_u, y_h, y_w, y_v \in \mathbb{Z}_p$ and sets $g = k^a, u = (k^a)^A k^{y_u}, h = (k^a)^B k^{y_h}, w = k^{y_w}, \hat{g} = \hat{k}^a, \hat{u} = (\hat{k}^a)^A \hat{k}^{y_u}, \hat{h} = (\hat{k}^a)^B \hat{k}^{y_h}, \hat{v} = \hat{k}^{y_v}$. It implicitly sets $\phi_1 = (\nu_1\nu_3 - \nu_2)b - \nu_1\pi + (a + y_\tau), \phi_2 = -\nu_3 b + \pi, \phi_3 = b, \tau = a + y_\tau$ and publishes a public key as

$$
\begin{aligned}
&g, u, h,\ w_1 = ((k^b)^{\nu_1\nu_3-\nu_2} k^{-\nu_1\pi} (k^a)^{y_\tau})^{y_w}, w_2 = ((k^b)^{-\nu_3} k^{\pi})^{y_w}, w_3 = (k^b)^{y_w}, w,\\
&\hat{g}, \hat{g}^{\nu_1}, \hat{g}^{\nu_2}, \hat{g}^{-\tau} = (\hat{k}^{a^2} (\hat{k}^a)^{y_\tau})^{-1}),\ \hat{u}, \hat{u}^{\nu_1}, \hat{u}^{\nu_2}, \hat{u}^{-\tau} = ((\hat{k}^{a^2})^A (\hat{k}^a)^{y_u + A y_\tau} \hat{k}^{y_u y_\tau})^{-1},\\
&\hat{h}, \hat{h}^{\nu_1}, \hat{h}^{\nu_2}, \hat{h}^{-\tau} = ((\hat{k}^{a^2})^B (\hat{k}^a)^{y_h + B y_\tau} \hat{k}^{y_h y_\tau})^{-1},\ \hat{v}, \hat{v}^{\nu_3}, \hat{v}^{-\pi},\ \Omega = e(k^a, \hat{k}^a)^\alpha.
\end{aligned}
$$

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 $(f^{\nu_1\nu_3-\nu_2}, f^{-\nu_3}, f, 1)$ 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

$$
\begin{aligned}
&W_{1,1} = g^\alpha (k^c)^{-(y_u M + y_h)} (u^M h)^{r'} (T)^{(\nu_1\nu_3-\nu_2)(AM+B)} (k^c)^{(-\nu_1\pi+y_\tau)(AM+B)} w_1^{c_1'},\\
&W_{1,2} = (T)^{-\nu_3(AM+B)} (k^c)^{\pi(AM+B)} w_2^{c_1'},\ W_{1,3} = (T)^{(AM+B)} w_3^{c_1'},\ W_{1,4} = (k^c)^{(AM+B)} w^{c_1'},\\
&W_{2,1} = g^{r'} (T)^{(\nu_1\nu_3-\nu_2)} (k^c)^{(-\nu_1\pi+y_\tau)} w_1^{c_2'},\ W_{2,2} = (T)^{-\nu_3} (k^c)^{y_w\pi} w_2^{c_2'},\ W_{2,3} = T w_3^{c_2'},\ W_{2,4} = (k^c)^{y_w} w^{c_2'}.
\end{aligned}
$$

- Case $j > k$: It creates a normal signature by calling **PKS.Sign** since it knows α of the private key. Note that x, y are not required.

Finally, $\mathcal{A}$ outputs a forged signature $\sigma^* = (W_{1,1}^*, \ldots, W_{2,4}^*)$ on a message M^*. To verify the forged signature, $\mathcal{B}_2$ first chooses random exponents $t', s_1, s_2 \in \mathbb{Z}_p$ and computes semi-functional verification components by implicitly setting $t = bx + t',\ s_c = -a^2 x,\ z_c = AM^* + B$ as

$$
\begin{aligned}
&V_{1,1} = \hat{k}^{abx} (\hat{k}^a)^{t'},\ V_{1,2} = (\hat{k}^{abx})^{\nu_1} (\hat{k}^a)^{\nu_1 t'} \hat{v}^{s_1},\\
&V_{1,3} = (\hat{k}^{abx})^{\nu_2} (\hat{k}^a)^{\nu_2 t'} \hat{v}^{\nu_3 s_1} (\hat{k}^{a^2x})^{-1},\ V_{1,4} = (\hat{k}^{abx})^{-y_\tau} (\hat{k}^{a^2})^{-t'} (\hat{k}^a)^{-y_\tau t'} \hat{v}^{-\pi s_1},\\
&V_{2,1} = (\hat{k}^{abx})^{AM^*+B} (\hat{k}^{bx})^{y_u M^* + y_h} (\hat{u}^{M^*} \hat{h})^{t'},\\
&V_{2,2} = (\hat{k}^{abx})^{(AM^*+B)\nu_1} (\hat{k}^{bx})^{(y_u M^* + y_h)\nu_1} (\hat{u}^{M^*} \hat{h})^{\nu_1 t'} \hat{v}^{s_2},\\
&V_{2,3} = (\hat{k}^{abx})^{(AM^*+B)\nu_2} (\hat{k}^{bx})^{(y_u M^* + y_h)\nu_2} (\hat{u}^{M^*} \hat{h})^{\nu_2 t'} \hat{v}^{\nu_3 s_2} (\hat{k}^{a^2x})^{-(AM^*+B)},\\
&V_{2,4} = (\hat{k}^{abx})^{-(AM^*+B)y_\tau - (y_u M^* + y_h)} (\hat{k}^{bx})^{-(y_u M^* + y_h) y_\tau} (\hat{k}^{a^2})^{-(AM^*+B)t'} (\hat{k}^a)^{-(y_u M^* + y_h)t'} (\hat{u}^{M^*} \hat{h})^{-y_\tau t'} \hat{v}^{-\pi s_2}.
\end{aligned}
$$

Next, it verifies that $\prod_{i=1}^4 e(W_{1,i}^*, V_{1,i}) \cdot \prod_{i=1}^4 e(W_{2,i}^*, V_{2,i})^{-1} \stackrel{?}{=} e(k^a, \hat{k}^{abx})^\alpha \cdot e(k^a, \hat{k}^a)^{\alpha t'}$. 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

$$\begin{aligned}
W_{1,1} &= g^{\alpha}(u^M h)^r w_1^{c_1} = g^{\alpha}(k^{(aA+y_u)M}k^{aB+y_h})^{-c+r'}(k^{y_w((\nu_1\nu_3-\nu_2)b-\nu_1\pi+(a+y_\tau))})^{c(AM+B)/y_w+c_1'} \\
&= g^{\alpha}(k^c)^{-(y_uM+y_h)}(u^M h)^{r'}(T)^{(\nu_1\nu_3-\nu_2)(AM+B)}(k^c)^{(-\nu_1\pi+y_\tau)(AM+B)}w_1^{c_1'}, \\
W_{1,2} &= w_2^{c_1} = (k^{y_w(-\nu_3 b+\pi)})^{c(AM+B)/y_w+c_1'} = (T)^{-\nu_3(AM+B)}(k^c)^{\pi(AM+B)}w_2^{c_1'}, \\
W_{1,3} &= w_3^{c_1} = (k^{y_w b})^{c(AM+B)/y_w+c_1'} = (T)^{(AM+B)}w_3^{c_1'}, \\
W_{1,4} &= w^{c_1} = (k^{y_w})^{c(AM+B)/y_w+c_1'} = (k^c)^{(AM+B)}w^{c_1'}.
\end{aligned}$$

The semi-functional verification components are correctly distributed as

$$\begin{aligned}
V_{2,1} &= (\hat{u}^{M^*}\hat{h})^t = (\hat{k}^{(aA+y_u)M^*}\hat{k}^{aB+y_h})^{bx+t'} = (\hat{k}^{abx})^{AM^*+B}(\hat{k}^{bx})^{y_uM^*+y_h}(\hat{u}^{M^*}\hat{h})^{t'}, \\
V_{2,2} &= ((\hat{u}^{\nu_1})^{M^*}\hat{h}^{\nu_1})^t\hat{v}^{s_2} = (\hat{k}^{(aA+y_u)\nu_1M^*}\hat{k}^{(aB+y_h)\nu_1})^{bx+t'}\hat{v}^{s_2} \\
&= (\hat{k}^{abx})^{(AM^*+B)\nu_1}(\hat{k}^{bx})^{(y_uM^*+y_h)\nu_1}(\hat{u}^{M^*}\hat{h})^{\nu_1 t'}\hat{v}^{s_2}, \\
V_{2,3} &= ((\hat{u}^{\nu_2})^{M^*}\hat{h}^{\nu_2})^t(\hat{v}^{\nu_3})^{s_2}\hat{f}^{s_c z_c} = (\hat{k}^{(aA+y_u)\nu_2M^*}\hat{k}^{(aB+y_h)\nu_2})^{bx+t'}(\hat{v}^{\nu_3})^{s_2}\hat{k}^{-a^2x(AM^*+B)} \\
&= (\hat{k}^{abx})^{(AM^*+B)\nu_2}(\hat{k}^{bx})^{(y_uM^*+y_h)\nu_2}(\hat{u}^{M^*}\hat{h})^{\nu_2 t'}\hat{v}^{\nu_3 s_2}(\hat{k}^{a^2x})^{-(AM^*+B)}, \\
V_{2,4} &= ((\hat{u}^{-\tau})^{M^*}\hat{h}^{-\tau})^t(\hat{v}^{-\pi})^{s_2}(\hat{f}^{-\phi_3})^{s_c z_c} \\
&= (\hat{k}^{-(aA+y_u)(a+y_\tau)M^*}\hat{k}^{-(aB+y_h)(a+y_\tau)})^{bx+t'}(\hat{v}^{-\pi})^{s_2}\hat{k}^{-b(-a^2x)(AM^*+B)} \\
&= (\hat{k}^{abx})^{-(AM^*+B)y_\tau-(y_uM^*+y_h)}(\hat{k}^{bx})^{-(y_uM^*+y_h)y_\tau}(\hat{k}^{a^2})^{-(AM^*+B)t'}(\hat{k}^{a})^{-(y_uM^*+y_h)t'}(\hat{u}^{M^*}\hat{h})^{-y_\tau t'}\hat{v}^{-\pi s_2}.
\end{aligned}$$

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 the simulator 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 $W_{1,1}, W_{1,2}, W_{1,3}, W_{2,1}, W_{2,2}, W_{2,3}$ that have T in the simulation additionally have $(k^d)^{(\nu_1\nu_3-\nu_2)(AM+B)}$, $(k^d)^{-\nu_3(AM+B)}$, $(k^d)^{(AM+B)}$, $(k^d)^{(\nu_1\nu_3-\nu_2)}$, $(k^d)^{-\nu_3}, k^d$ 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 the adversary cannot distinguish the nominally semi-functional signature from the semi-functional signature. The main idea of this is that the adversary cannot request a signature for the forgery message M^* in the security model. Suppose there exists an unbounded adversary, then the adversary 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 4.3. *If Assumption 3 holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_2$ and $\mathbf{G}_3$ 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. 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 $\nu_1, \nu_3,\ \phi_1, \phi_2, \phi_3 \in \mathbb{Z}_p$ and sets $\pi = \phi_2 + \nu_3\phi_3$. It selects random exponents $y_g, x, y, y_w, y_v \in \mathbb{Z}_p$ and sets $g = k^{y_g}, u = g^x, h = g^y, w = k^{y_w}, \hat{g} = \hat{k}^{y_g}, \hat{u} = \hat{g}^x, \hat{h} = \hat{g}^y, \hat{v} = \hat{k}^{y_v}$. It implicitly sets $\nu_2 = a, \tau = \phi_1 + \nu_1\phi_2 + a\phi_3, \alpha = ab$ and publishes a public key as

$$
\begin{aligned}
&g, u, h,\ w_1 = w^{\phi_1}, w_2 = w^{\phi_2}, w_3 = w^{\phi_3}, w,\ \hat{g}, \hat{g}^{\nu_1}, \hat{g}^{\nu_2} = (\hat{k}^a)^{y_g}, \hat{g}^{-\tau} = \hat{k}^{-y_g(\phi_1+\nu_1\phi_2)}(\hat{k}^a)^{-y_g\phi_3},\\
&\hat{u}, \hat{u}^{\nu_1}, \hat{u}^{\nu_2} = (\hat{g}^{\nu_2})^x, \hat{u}^{-\tau} = (\hat{g}^{-\tau})^x,\ \hat{h}, \hat{h}^{\nu_1}, \hat{h}^{\nu_2} = (\hat{g}^{\nu_2})^y, \hat{h}^{-\tau} = (\hat{g}^{-\tau})^y,\\
&\hat{v}, \hat{v}^{\nu_3}, \hat{v}^{-\pi},\ \Omega = e(k^a, \hat{k}^b)^{y_g^2}.
\end{aligned}
$$

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

$$
\begin{aligned}
&W_{1,1} = (u^M h)^r w_1^{c_1} (k^b)^{\nu_1\nu_3 y_g} k^{\nu_1\nu_3 s_k z_k'} (k^a)^{-s_k z_k'},\\
&W_{1,2} = w_2^{c_1} (k^b)^{-\nu_3 y_g} k^{-\nu_3 s_k z_k'},\ W_{1,3} = w_3^{c_1} (k^b)^{y_g} k^{s_k z_k'},\ W_{1,4} = w^{c_1},\\
&W_{2,1} = g^r w_1^{c_2} k^{\nu_1\nu_3 s_k} (k^a)^{-s_k},\ W_{2,2} = w_2^{c_2} k^{-\nu_3 s_k},\ W_{2,3} = w_3^{c_2} k^{s_k},\ W_{2,4} = w^{c_2}.
\end{aligned}
$$

The simulator 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 $\sigma^* = (W_{1,1}^*, \ldots, W_{2,4}^*)$ 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

$$
\begin{aligned}
&V_{1,1} = (\hat{k}^c)^{y_g},\ V_{1,2} = (\hat{k}^c)^{y_g\nu_1} \hat{v}^{s_1},\ V_{1,3} = \hat{v}^{\nu_3 s_1} \hat{k}^{s_c'},\ V_{1,4} = (\hat{k}^c)^{-y_g(\phi_1+\nu_1\phi_2)} \hat{v}^{-\pi s_1} \hat{k}^{-\phi_3 s_c'},\\
&V_{2,1} = (\hat{k}^c)^{y_g(xM^*+y)},\ V_{2,2} = (\hat{k}^c)^{y_g(xM^*+y)\nu_1} \hat{v}^{s_2},\ V_{2,3} = \hat{v}^{\nu_3 s_2} \hat{k}^{z_c'},\\
&V_{2,4} = (\hat{k}^c)^{-y_g(xM^*+y)(\phi_1+\nu_1\phi_2)} \hat{v}^{-\pi s_2} \hat{k}^{-\phi_3 z_c'}.
\end{aligned}
$$

Next, it verifies that $\prod_{i=1}^4 e(W_{1,i}^*, V_{1,i}) \cdot \prod_{i=1}^4 e(W_{2,i}^*, V_{2,i})^{-1} \stackrel{?}{=} (T)^{y_g^2}$. 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 blinding values y_g, y_w, y_v are used. The semi-functional signature is correctly distributed as

$$
\begin{aligned}
W_{1,1} &= g^\alpha (u^M h)^r w_1^{c_1} (f^{\nu_1\nu_3 - \nu_2})^{s_k z_k} = k^{y_g ab} (u^M h)^r w_1^{c_1} (k^{\nu_1\nu_3 - a})^{s_k(by_g/s_k + z_k')}\\
&= (u^M h)^r w_1^{c_1} (k^b)^{\nu_1\nu_3 y_g} k^{\nu_1\nu_3 s_k z_k'} (k^a)^{-s_k z_k'}.
\end{aligned}
$$

The semi-functional verification components are correctly distributed as

$$
\begin{aligned}
V_{1,3} &= (\hat{g}^{\nu_2})^t (\hat{v}^{\nu_3})^{s_1} \hat{f}^{s_c} = (\hat{k}^{y_g a})^c \hat{v}^{\nu_3 s_1} \hat{k}^{-acy_g + s_c'} = \hat{v}^{\nu_3 s_1} \hat{k}^{s_c'},\\
V_{1,4} &= (\hat{g}^{-\tau})^t (\hat{v}^{-\pi})^{s_1} (\hat{f}^{-\phi_3})^{s_c} = (\hat{k}^{-y_g(\phi_1+\nu_1\phi_2+a\phi_3)})^c \hat{v}^{-\pi s_1} \hat{k}^{-\phi_3(-acy_g + s_c')}\\
&= (\hat{k}^c)^{-y_g(\phi_1+\nu_1\phi_2)} \hat{v}^{-\pi s_1} \hat{k}^{-\phi_3 s_c'},
\end{aligned}
$$

$$
\begin{aligned}
V_{2,3} &= (\hat{u}^{\nu_2 M^*}\hat{h}^{\nu_2})^t(\hat{v}^{\nu_3})^{s_2}\hat{f}^{s_c z_c} = (\hat{k}^{y_g a(xM^*+y)})^c(\hat{v}^{\nu_3})^{s_2}\hat{k}^{-acy_g(xM^*+y)+z'_c} = \hat{v}^{\nu_3 s_2}\hat{k}^{z'_c},\\
V_{2,4} &= (\hat{u}^{-\tau M^*}\hat{h}^{-\tau})^t(\hat{v}^{-\pi})^{s_2}(\hat{f}^{-\phi_3})^{s_c z_c}\\
&= (\hat{k}^{-y_g(\phi_1+\nu_1\phi_2+a\phi_3)(xM^*+y)})^c(\hat{v}^{-\pi})^{s_2}(\hat{k}^{-\phi_3})^{-acy_g(xM^*+y)+z'_c}\\
&= (\hat{k}^c)^{-y_g(xM^*+y)(\phi_1+\nu_1\phi_2)}\hat{v}^{-\pi s_2}\hat{k}^{-\phi_3 z'_c},\\
\Omega^t &= e(g,\hat{g})^{\alpha t} = e(k,\hat{k})^{y_g^2 abc} = (T_0)^{y_g^2}.
\end{aligned}
$$

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 $\Pr[\mathcal{B}_3(D,T_0)=0] = \mathbf{Adv}_{\mathcal{A}}^{G_2}$ and $\Pr[\mathcal{B}_3(D,T_1)=0] = \mathbf{Adv}_{\mathcal{A}}^{G_3}$ from the above analysis. Thus, we can easily derive the advantage of $\mathcal{B}_3$ as

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

This completes our proof. □

4.2 Proof of Theorem 3.2

Our overall proof strategy for this part follows Lu et al. [22] and adapts it to our setting. The proof uses two properties: the fact that the aggregated signature result is independent of the order of aggregation, and the fact that the simulator of the SAS system possesses the private keys of all but the target PKS.

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 $PK_{PKS} = (g,u,h,w_1,\ldots,w, \hat{g},\ldots,\hat{g}^{-\tau},\hat{u},\ldots,\hat{u}^{-\tau},\hat{h},\ldots,\hat{h}^{-\tau},\hat{v},\hat{v}^{\nu_3},\hat{v}^{-\pi},\Omega)$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}$ first constructs $PP = (g,w_1,\ldots,w,\hat{g},\ldots,\hat{g}^{-\tau},\hat{v},\hat{v}^{\nu_3},\hat{v}^{-\pi})$ and $PK^* = (u,h,\hat{u},\ldots,\hat{u}^{-\tau},\hat{h},\ldots,\hat{h}^{-\tau},\Omega = e(g,\hat{g})^{\alpha})$ 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 $PK_i = (u_i,h_i,\ldots,\Omega_i)$ 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 $1 \le i \le l-1$, 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 $AS^* = (S^*_{1,1},\ldots,S^*_{2,4})$ 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 calling **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 $PK_i = (u_i, h_i, \ldots, \Omega_i)$ 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

$$W_{1,1} = S^*_{1,1} \cdot \prod_{i=2}^{l} \left(g^{\alpha_j} (S^*_{2,1})^{x_i M_i + y_i}\right)^{-1},\ W_{1,2} = S^*_{1,2} \cdot \prod_{i=2}^{l} \left((S^*_{2,2})^{x_i M_i + y_i}\right)^{-1},$$
$$W_{1,3} = S^*_{1,3} \cdot \prod_{i=2}^{l} \left((S^*_{2,3})^{x_i M_i + y_i}\right)^{-1},\ W_{1,4} = S^*_{1,4} \cdot \prod_{i=2}^{l} \left((S^*_{2,4})^{x_i M_i + y_i}\right)^{-1},$$
$$W_{2,1} = S^*_{2,1},\ W_{2,2} = S^*_{2,2},\ W_{2,3} = S^*_{2,3},\ W_{2,4} = S^*_{2,4}.$$

3. It outputs $\sigma = (W_{1,1}, \ldots, W_{2,4})$ 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 and the public key are correctly distributed. The sequential aggregate signatures is correctly distributed since this scheme does not check the order of aggregation. Finally, we can show that the result signature $\sigma = (W_{1,1}, \ldots, W_{2,4})$ 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:

$$\begin{aligned}
&\prod_{i=1}^{4} e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{4} e(W_{2,i}, V_{2,i})^{-1} \\
&= e(S^*_{1,1}, \hat{g}^t) \cdot e(S^*_{1,2}, \hat{g}^{\nu_1 t} \hat{v}^{s_1}) \cdot e(S^*_{1,3}, \hat{g}^{\nu_2 t} \hat{v}^{\nu_3 s_1}) \cdot e(S^*_{1,4}, \hat{g}^{-\tau t} \hat{v}^{-\pi s_1}) \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g}^t)^{-1} \cdot \\
&\quad e(S^*_{2,1}, \prod_{i=2}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^t)^{-1} \cdot e(S^*_{2,2}, \prod_{i=2}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{\nu_1 t} \hat{v}^{\delta_i s_1})^{-1} \cdot e(S^*_{2,3}, \prod_{i=2}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{\nu_2 t} \hat{v}^{\delta_i s_1})^{-1} \cdot \\
&\quad e(S^*_{2,4}, \prod_{i=2}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{-\tau t} \hat{v}^{-\pi \delta_i s_1})^{-1} \cdot e(S^*_{2,1}, (\hat{u}^{M_1} \hat{h})^t)^{-1} \cdot e(S^*_{2,2}, (\hat{u}^{M_1} \hat{h})^{\nu_1 t} \hat{v}^{s_2})^{-1} \cdot \\
&\quad e(S^*_{2,3}, (\hat{u}^{M_1} \hat{h})^{\nu_2 t} \hat{v}^{\nu_3 s_2})^{-1} \cdot e(S^*_{2,4}, (\hat{u}^{M_1} \hat{h})^{-\tau t} \hat{v}^{-\pi s_2})^{-1} \\
&= e(S^*_{1,1}, C_{1,1}) \cdot e(S^*_{1,2}, C_{1,2}) \cdot e(S^*_{1,3}, C_{1,3}) \cdot e(S^*_{1,4}, C_{1,4}) \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g}^t)^{-1} \cdot \\
&\quad e(S^*_{2,1}, \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^t)^{-1} \cdot e(S^*_{2,2}, \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{\nu_1 t} \hat{v}^{\tilde{s}_2})^{-1} \cdot e(S^*_{2,3}, \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{\nu_2 t} \hat{v}^{\tilde{s}_2})^{-1} \cdot \\
&\quad e(S^*_{2,4}, \prod_{i=1}^{l} (\hat{u}_i^{M_i} \hat{h}_i)^{-\tau t} \hat{v}^{-\pi \tilde{s}_2})^{-1} \cdot \\
&= \prod_{i=1}^{4} e(S^*_{1,i}, C_{1,i}) \cdot \prod_{i=1}^{4} e(S^*_{2,i}, C_{2,i})^{-1} \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g}^t)^{-1} = \prod_{i=1}^{l} \Omega_i^t \cdot \prod_{i=2}^{l} \Omega_i^{-t} = \Omega_1^t
\end{aligned}$$

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.

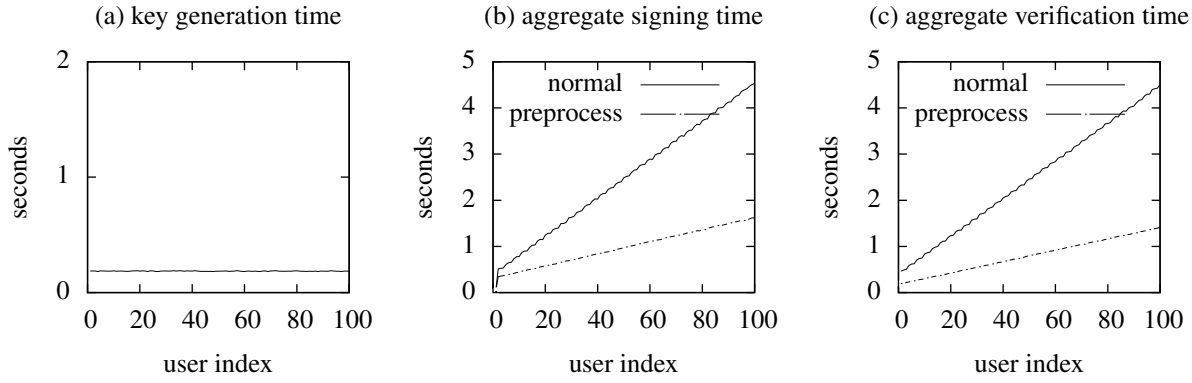

Figure 1: Performance of our SAS scheme

5 Implementation

In this section, we report on the implementation of our SAS scheme and analysis of its performance. Then, we propose a method to improve the performance of our scheme in a typical applications scenario.

We used the Pairing Based Cryptography (PBC) library of Ben Lynn [23] to implement our SAS scheme. According to the NIST recommendations for the 80-bit security [26], the key size of elliptic curve systems should be at least 160 bits and the key size of discrete logarithm systems should be at least 1024 bits. For 80-bit security, we, therefore, selected the Miyaji-Nakabayashi-Takano (MNT) curve with embedding degree 6 since this embedding degree is close to the optimal value, i.e., 1024/160=6.4 for this level of security. In the MNT curve with embedding degree 6, the group size of $\mathbb{G}$ should be at least 171 bits and the group size of $\mathbb{G}_T$ should be at least 1024 bits since the security of the $\mathbb{G}_T$ group is related to the security of the discrete logarithm [13]. Therefore, we used a 175-bit MNT curve that is generated by the MNT parameter generation program in the PBC library.

5.1 Signature and Public Key Size

We compare the signature size and the public key size of Lu et al.'s SAS scheme (the earlier scheme with non relaxed-model proof, based on a static assumption and standard model) with our SAS scheme. The original SAS scheme of Lu et al. is described using symmetric bilinear groups, but it can also be described using asymmetric bilinear groups. In the 175-bit MNT curve with point compression, the group size of $\mathbb{G}$ is about 175 bits, the group size of $\hat{\mathbb{G}}$ is about 525 bits, and the group size of $\mathbb{G}_T$ is 1050 bits respectively.

In Lu et al. system, the size of an aggregate signature is about 350 bits and the size of a public key is about 113,000 bits. Alternately, one may consider to use the method of Chatterjee and Sarkar [12] to reduce the public key size of the SAS scheme of Lu et al. However, this method obtains shorter public key size by sacrificing the security reduction of the scheme. Thus, it should use a larger size of prime for the order of groups to support the same security level of the original scheme; with this optimally increased size groups, our SAS scheme will still have 20 times shorter public key size.

5.2 Performance Measurements

We implemented and measured the performance of our SAS scheme on a notebook computer with an Intel Core i5-460M 2.53 GHz CPU. The PBC library on the test machine can compute a pairing operation in 14.0 ms, an exponentiation operation of $\mathbb{G}$ and $\hat{\mathbb{G}}$ in 1.7 ms and 20.3 ms respectively. We assume that there are 100 users who participate in the sequential aggregate signature system (indexed 1 to 100).

At first, the setup algorithm takes about 0.159 seconds to generate the public parameters since it requires three exponentiations in $\mathbb{G}$ and five exponentiations in $\hat{\mathbb{G}}$. The key generation algorithm for each user takes about 0.185 seconds since it requires six exponentiations in $\hat{\mathbb{G}}$ and one pairing. The aggregate signing algorithm mainly consists of verifying the previous aggregate signature and adding its own signature into the aggregate signature. The time to generate an aggregate signature is proportional to the index number of the user who participates in the aggregate signing algorithm. Furthermore, this algorithm spends nearly 98 percent of its time on verifying the previous aggregate signature since it should compute $4l + 14$ numbers of exponentiation in $\hat{\mathbb{G}}$ where l is the number of previous signers. For example, if a user's index is 50 in the aggregate signing algorithm, then the algorithm verifies the previous aggregate signature in 2.421 seconds (this amount will double for our last user), and adds its signature into the aggregate signature in 0.065 seconds.

Optimization: We can improve the performance of the aggregate verification algorithm by preprocessing the exponentiations in $\hat{\mathbb{G}}$. To use the preprocessing method, users should keep the public keys of the previous users. If the set of users who participate in the aggregate signature system is not changed or changed a little (as in the routing and the certification cases), then users can preprocess the public keys of previous users after running the first aggregate signing algorithm. Additionally, we can preprocess the public parameters and pre-compute elements for verification in an offline mode. If the preprocessing method is used, then the time to verify an aggregate signature is reduced to 30 percent of the original time to verify.

6 Conclusion

In this paper, we proposed a sequential aggregate signature scheme with a proof of security in the standard model and with no relaxation of assumptions (i.e., employing neither random oracles nor interactive assumptions). The proposed scheme is the first of this kind which has short (constant number of group elements) size public keys and constant number of pairing operations per message in the verification algorithm. Also, we provided an implementation and performance measurements of our scheme.

Acknowledgements

We thank Adam O'Neill and Benoît Libert for their helpful comments. We gratefully thank the anonymous reviewers of PKC 2013 for their valuable comments.

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] Ali Bagherzandi and Stanislaw Jarecki. Identity-based aggregate and multi-signature schemes based on rsa. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 480–498. Springer, 2010.

[3] 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.

[4] Mihir Bellare and Gregory Neven. Identity-based multi-signatures from rsa. In Masayuki Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 145–162. Springer, 2007.

[5] 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.

[6] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 276–285. ACM, 2007.

[7] 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`.

[8] 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.

[9] 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.

[10] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[11] 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.

[12] Sanjit Chatterjee and Palash Sarkar. Trading time for space: Towards an efficient ibe scheme with short(er) public parameters in the standard model. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *Lecture Notes in Computer Science*, pages 424–440. Springer, 2005.

[13] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[14] 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.

[15] 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.

[16] Javier Herranz. Deterministic identity-based signatures for partial aggregation. *Comput. J.*, 49(3):322–330, 2006.

[17] Jung Yeon Hwang, Dong Hoon Lee, and Moti Yung. Universal forgery of the identity-based sequential aggregate signature scheme. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS*, pages 157–160. ACM, 2009.

[18] Jonathan Katz and Andrew Y. Lindell. Aggregate message authentication codes. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2008.

[19] Kwangsu Lee. The sequential aggregate signature program code. `https://sites.google.com/site/kwangsulee07/`.

[20] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2012.

[21] 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.

[22] 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.

[23] Ben Lynn. The pairing-based cryptography library. `http://crypto.stanford.edu/pbc/`.

[24] 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.

[25] 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.

[26] NIST. Recommendation for key management, 2011. `http://csrc.nist.gov/publications/PubsSPs.html`.

[27] 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.

[28] 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.

[29] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636. Springer, 2009.

A Lewko-Waters IBE

In this section, we first describe the IBE scheme in prime order (asymmetric) bilinear groups of Lewko and Waters [21]. Next we describe the PKS scheme that is derived from the IBE scheme of Lewko and Waters. We slightly change the notation of the IBE scheme.

A.1 The LW-IBE Scheme

The IBE scheme in prime order bilinear groups of Lewko and Waters is described as follows:

IBE.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 $g \in \mathbb{G}$ and $\hat{g}, \hat{w} \in \hat{\mathbb{G}}$. Next, it chooses random exponents $\nu, \phi_1, \phi_2 \in \mathbb{Z}_p$ and sets $\tau = \phi_1 + \nu\phi_2$. It selects random exponents $\alpha, x, y \in \mathbb{Z}_p$ and sets $u = g^x, \hat{u} = \hat{g}^x, h = g^y, \hat{h} = \hat{g}^y$. It outputs a master key $MK = (\hat{g}, \hat{u}, \hat{h}, \hat{g}^\alpha, \hat{w}_1 = \hat{w}^{\phi_1}, \hat{w}_2 = \hat{w}^{\phi_2}, \hat{w})$ and public parameters as

$$PP = \big(g, g^\nu, g^{-\tau},\ u, u^\nu, u^{-\tau},\ h, h^\nu, h^{-\tau},\ \Omega = e(g,\hat{g})^\alpha \big).$$

IBE.KeyGen(ID, MK): This algorithm takes as input an identity $ID \in \{0,1\}^k$ where $k < \lambda$ and the master key MK. It selects random exponents $r, c_1, c_2 \in \mathbb{Z}_p$ and outputs a private key as

$$SK_{ID} = \big(K_{1,1} = \hat{g}^\alpha (\hat{u}^{ID}\hat{h})^r \hat{w}_1^{c_1}, K_{1,2} = \hat{w}_2^{c_1}, K_{1,3} = \hat{w}^{c_1},$$
$$K_{2,1} = \hat{g}^r \hat{w}_1^{c_2}, K_{2,2} = \hat{w}_2^{c_2}, K_{2,3} = \hat{w}^{c_2} \big).$$

IBE.Encrypt(M, ID, PP): This algorithm takes as input a message $M \in \mathbb{G}_T$, an identity ID, and the public parameters PP. It first chooses a random exponent $t \in \mathbb{Z}_p$ and outputs a ciphertext as

$$CT = \big(C = e(g,\hat{g})^{\alpha t} M,\ C_{1,1} = g^t, C_{1,2} = (g^\nu)^t, C_{1,3} = (g^{-\tau})^t,$$
$$C_{2,1} = (u^{ID}h)^t, C_{2,2} = ((u^\nu)^{ID} h^\nu)^t, C_{2,3} = ((u^{-\tau})^{ID} h^{-\tau})^t \big).$$

IBE.Decrypt(CT, SK_{ID}, PP): This algorithm takes as input a ciphertext CT, a private key SK_{ID}, and the public parameters PP. If the identities of the ciphertext and the private key are equal, then it computes

$$M \leftarrow C \cdot \prod_{i=1}^{3} e(C_{1,i}, K_{1,i})^{-1} \cdot \prod_{i=1}^{3} e(C_{2,i}, K_{2,i}).$$

A.2 The LW-PKS Scheme

To derive a PKS scheme from the IBE scheme of Lewko and Waters, we apply the transformation of Naor [8]. Additionally, we represent the signature in $\mathbb{G}$ instead of $\hat{\mathbb{G}}$ to reduce the size of signatures. The PKS scheme in prime order bilinear groups of Lewko and Waters 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 chooses random exponents $\nu, \phi_1, \phi_2 \in \mathbb{Z}_p$ and sets $\tau = \phi_1 + \nu\phi_2$. It selects random exponents $\alpha, x, y \in \mathbb{Z}_p$ and sets $u = g^x, \hat{u} = \hat{g}^x, h = g^y, \hat{h} = \hat{g}^y$. It outputs a private key $SK = (g, u, h, g^\alpha)$ and a public key as

$$PK = \big(w_1 = w^{\phi_1}, w_2 = w^{\phi_2}, w,\ \hat{g}, \hat{g}^\nu, \hat{g}^{-\tau},\ \hat{u}, \hat{u}^\nu, \hat{u}^{-\tau},\ \hat{h}, \hat{h}^\nu, \hat{h}^{-\tau},\ \Omega = e(g,\hat{g})^\alpha \big).$$

PKS.Sign(M, SK): This algorithm takes as input a message $M \in \{0,1\}^k$ where $k < \lambda$ and a private key $SK = (g, u, h, g^\alpha)$. 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.Verify(σ, M, PK): This algorithm takes as input a signature σ on a message $M \in \{0,1\}^k$ under a public key PK. It first 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 $\prod_{i=1}^3 e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^3 e(W_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$. If this equation holds, then it outputs 1. Otherwise, it outputs 0.

We can safely move the elements w_1, w_2, w from the private key to the public key since these elements are always constructed in the security proof of the IBE scheme. However, this PKS scheme does not support multi-user setting and public re-randomization since the elements g, u, h are not given in the public key.

B Multi-Signature

In this section, we construct a multi-signature (MS) scheme with short public parameters and prove its existential unforgeability under a chosen message attack.

B.1 Definitions

A multi-signature (MS) scheme consists of six PPT algorithms **Setup**, **KeyGen**, **Sign**, **Verify**, **Combine**, and **MultVerify**, which are defined as follows: The setup algorithm **Setup**(1^λ) takes as input a security parameter λ, and outputs public parameters PP. The key generation algorithm **KeyGen**(PP) takes as input the public parameters PP, and outputs a public key PK and a private key SK. The signing algorithm **Sign**(M, SK) takes as input a message M, and a private key SK. It outputs a signature σ. The verification algorithm **Verify**(σ, M, PK) 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. The combining algorithm **Combine**$(\sigma, M, \mathbf{PK})$ 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. The multi-verification algorithm **MultVerify**$(MS, M, \mathbf{PK})$ 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 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 $\mathcal{A}$:

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 $\mathcal{A}$.

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.

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.

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) $\mathbf{MultVerify}(MS^*, M^*, \mathbf{PK}^*) = 1$, 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.

B.2 Our MS Scheme

The MS scheme derived from the PKS scheme does not require to re-randomize the randomness of multi-signatures in the combine algorithm since each signer uses a fresh random value. However, it requires all signers to share the elements g, u, h in the public parameters. The MS scheme in prime order 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 $g, w \in G$ and $\hat{g}, \hat{v} \in \hat{G}$. Next, it chooses random exponents $\nu_1, \nu_2, \nu_3, \phi_1, \phi_2, \phi_3 \in \mathbb{Z}_p$ and sets $\tau = \phi_1 + \nu_1\phi_2 + \nu_2\phi_3, \pi = \phi_2 + \nu_3\phi_3$. 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 as

$$PP = \big(g, u, h,\ w_1 = w^{\phi_1}, w_2 = w^{\phi_2}, w_3 = w^{\phi_3}, w,\ \hat{g}, \hat{g}^{\nu_1}, \hat{g}^{\nu_2}, \hat{g}^{-\tau},$$
$$\hat{u}, \hat{u}^{\nu_1}, \hat{u}^{\nu_2}, \hat{u}^{-\tau},\ \hat{h}, \hat{h}^{\nu_1}, \hat{h}^{\nu_2}, \hat{h}^{-\tau},\ \hat{v}, \hat{v}^{\nu_3}, \hat{v}^{-\pi} \big).$$

MS.KeyGen(PP): This algorithm takes as input the public parameters PP. It selects a random exponent $\alpha \in \mathbb{Z}_p$ and computes $\Omega = e(g, \hat{g})^\alpha$. 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 $M \in \{0,1\}^k$ where $k < \lambda$ and a private key $SK = \alpha$. It selects random exponents $r, c_1, c_2 \in \mathbb{Z}_p$ and outputs a signature as

$$\sigma = \big(W_{1,1} = g^\alpha (u^M h)^r w_1^{c_1}, W_{1,2} = w_2^{c_1}, W_{1,3} = w_3^{c_1}, W_{1,4} = w^{c_1},$$
$$W_{2,1} = g^r w_1^{c_2}, W_{2,2} = w_2^{c_2}, W_{2,3} = w_3^{c_2}, W_{2,4} = w^{c_2} \big).$$

MS.Verify(σ, M, PK): This algorithm takes as input a signature σ on a message M under a public key PK. It first chooses random exponents $t, s_1, s_2 \in \mathbb{Z}_p$ and computes

$$V_{1,1} = \hat{g}^t, V_{1,2} = (\hat{g}^{\nu_1})^t \hat{v}^{s_1}, V_{1,3} = (\hat{g}^{\nu_2})^t (\hat{v}^{\nu_3})^{s_1}, V_{1,4} = (\hat{g}^{-\tau})^t (\hat{v}^{-\pi})^{s_1},$$
$$V_{2,1} = (\hat{u}^M \hat{h})^t, V_{2,2} = ((\hat{u}^{\nu_1})^M \hat{h}^{\nu_1})^t \hat{v}^{s_2}, V_{2,3} = ((\hat{u}^{\nu_2})^M \hat{h}^{\nu_2})^t (\hat{v}^{\nu_3})^{s_2}, V_{2,4} = ((\hat{u}^{-\tau})^M \hat{h}^{-\tau})^t (\hat{v}^{-\pi})^{s_2}.$$

Next, it verifies that $\prod_{i=1}^4 e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^4 e(W_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \Omega^t$. 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 $M \in \mathbb{Z}_p$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $PK_i = \Omega_i$. It first checks the validity of each signature σ_i by calling $\mathbf{Verify}(\sigma_i, M, PK_i)$. If any signature is invalid, then it halts. It then outputs a multi-signature for a message M as

$$MS = \big(S_{1,1} = \prod_{i=1}^{l} W_{1,1}^i,\ S_{1,2} = \prod_{i=1}^{l} W_{1,2}^i,\ S_{1,3} = \prod_{i=1}^{l} W_{1,3}^i,\ S_{1,4} = \prod_{i=1}^{l} W_{1,4}^i,$$
$$S_{2,1} = \prod_{i=1}^{l} W_{2,1}^i,\ S_{2,2} = \prod_{i=1}^{l} W_{2,2}^i,\ S_{2,3} = \prod_{i=1}^{l} W_{2,3}^i,\ S_{2,4} = \prod_{i=1}^{l} W_{2,4}^i \big).$$

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 first chooses random exponents $t, s_1, s_2 \in \mathbb{Z}_p$ and computes

$$V_{1,1} = \hat{g}^t, V_{1,2} = (\hat{g}^{v_1})^t \hat{v}^{s_1}, V_{1,3} = (\hat{g}^{v_2})^t (\hat{v}^{v_3})^{s_1}, V_{1,4} = (\hat{g}^{-\tau})^t (\hat{v}^{-\pi})^{s_1},$$
$$V_{2,1} = (\hat{u}^M \hat{h})^t, V_{2,2} = ((\hat{u}^{v_1})^M \hat{h}^{v_1})^t \hat{v}^{s_2}, V_{2,3} = ((\hat{u}^{v_2})^M \hat{h}^{v_2})^t (\hat{v}^{v_3})^{s_2}, V_{2,4} = ((\hat{u}^{-\tau})^M \hat{h}^{-\tau})^t (\hat{v}^{-\pi})^{s_2}.$$

Next, it verifies that $\prod_{i=1}^{4} e(S_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{4} e(S_{2,i}, V_{2,i})^{-1} \stackrel{?}{=} \prod_{i=1}^{l} \Omega_i^t$. If this equation holds, then it outputs 1. Otherwise, it outputs 0.

The multi-signature $MS = (S_{1,1}, \ldots, S_{2,4})$ is a valid multi-signature on a message M under public keys $\mathbf{PK}$ with randomness $\tilde{r} = \sum_{i=1}^{l} r_i, \tilde{c}_1 = \sum_{i=1}^{l} c_{i,1}, \tilde{c}_2 = \sum_{i=1}^{l} c_{i,2}$ where $r_i, c_{i,1}, c_{i,2}$ are random values in σ_i. The multi-signature on a message M under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ has the following form

$$S_{1,1} = g^{\sum_{i=1}^{l} \alpha_i} (u_i^{M_i} h_i)^{\tilde{r}} w_1^{\tilde{c}_1},\ S_{1,2} = w_2^{\tilde{c}_1},\ S_{1,3} = w_3^{\tilde{c}_1},\ S_{1,4} = w^{\tilde{c}_1},$$
$$S_{2,1} = g^{\tilde{r}} w_1^{\tilde{c}_2},\ S_{2,2} = w_2^{\tilde{c}_2},\ S_{2,3} = w_3^{\tilde{c}_2},\ S_{2,4} = w^{\tilde{c}_2}.$$

B.3 Security Analysis

Theorem B.1. *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 $\mathbf{Adv}_{\mathcal{A}}^{MS}(\lambda) \leq \mathbf{Adv}_{\mathcal{B}}^{PKS}(\lambda)$.*

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 $PK_{PKS} = (g, u, h, w_1, \ldots, \hat{v}^{-\pi}, \Omega)$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}$ first constructs $PP = (g, u, h, w_1, \ldots, \hat{v}^{-\pi})$ 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 $MS^* = (S_{1,1}^*, \ldots, S_{2,4}^*)$ 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 **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 $SK_i = \alpha_i$ of PK_i from CL. It then computes

$$W_{1,1} = S^*_{1,1} \cdot \prod_{i=2}^{l} \left(g^{\alpha_i}\right)^{-1},\ W_{1,2} = S^*_{1,2},\ W_{1,3} = S^*_{1,3},\ W_{1,4} = S^*_{1,4},$$
$$W_{2,1} = S^*_{2,1},\ W_{2,2} = S^*_{2,2},\ W_{2,3} = S^*_{2,3},\ W_{2,4} = S^*_{2,4}.$$

3. It outputs $\sigma = (W_{1,1}, \ldots, W_{2,4})$ 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 $\sigma = (W_{1,1}, \ldots, W_{2,4})$ 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

$$\begin{aligned}\prod_{i=1}^{4} e(W_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{4} e(W_{2,i}, V_{2,i})^{-1} &= \prod_{i=1}^{4} e(S^*_{1,i}, V_{1,i}) \cdot \prod_{i=1}^{4} e(S^*_{2,i}, V_{2,i})^{-1} \cdot e(\prod_{i=2}^{l} g^{\alpha_i}, \hat{g}^t)^{-1} \\ &= \prod_{i=1}^{l} \Omega_i^t \cdot \prod_{i=2}^{l} \Omega_i^{-t} = \Omega_1^t = \Omega^t.\end{aligned}$$

This completes our proof. □

C Aggregate Signature in Composite Order Groups

We construct a SAS scheme in composite order groups and prove its existential unforgeability under a chosen message attack.

C.1 Bilinear Groups of Composite Order

Let $N = p_1 p_2 p_3$ where p_1, p_2, and p_3 are distinct prime numbers. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of same composite order n and g be a generator of $\mathbb{G}$. The bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ has the following properties:

1. Bilinearity: $\forall u, v \in \mathbb{G}$ and $\forall a, b \in \mathbb{Z}_n$, $e(u^a, v^b) = e(u, v)^{ab}$.

2. Non-degeneracy: $\exists g$ such that $e(g, g)$ has order N, that is, $e(g, g)$ is a generator of $\mathbb{G}_T$.

We say that $\mathbb{G}$ is a bilinear group if the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable. Furthermore, we assume that the description of $\mathbb{G}$ and $\mathbb{G}_T$ includes generators of $\mathbb{G}$ and $\mathbb{G}_T$ respectively. We use the notation $\mathbb{G}_{p_i}$ to denote the subgroups of order p_i of $\mathbb{G}$ respectively. Similarly, we use the notation $\mathbb{G}_{T,p_i}$ to denote the subgroups of order p_i of $\mathbb{G}_T$ respectively.

C.2 Complexity Assumptions

We introduce three static assumptions under composite order bilinear groups. These three assumptions were used by Lewko and Waters [21].

Assumption C.1 (Subgroup Decision) Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The assumption is that if the challenge tuple

$$D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3}) \text{ and } T,$$

are given, no PPT algorithm $\mathcal{B}$ can distinguish $T = T_0 = X_1$ from $T = T_1 = X_1 R_1$ 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 random choices of $X_1 \in \mathbb{G}_{p_1}$ and $R_1 \in \mathbb{G}_{p_2}$.

Assumption C.2 (Generalized Subgroup Decision) Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The assumption is that if the challenge tuple

$$D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3}, X_1 R_1, R_2 Y_1) \text{ and } T,$$

are given, no PPT algorithm $\mathcal{B}$ can distinguish $T = T_0 = X_2 Y_2$ from $T = T_1 = X_2 R_3 Y_2$ 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 random choices of $X_1, X_2 \in \mathbb{G}_{p_1}$, $R_1, R_2, R_3 \in \mathbb{G}_{p_2}$, and $Y_1, Y_2 \in \mathbb{G}_{p_3}$.

Assumption C.3 (Composite Diffie-Hellman) Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The assumption is that if the challenge tuple

$$D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_2}, g_{p_3}, g^a_{p_1} R_1, g^b_{p_1} R_2) \text{ and } T,$$

are given, no PPT algorithm $\mathcal{B}$ can distinguish $T = T_0 = e(g_{p_1}, g_{p_1})^{ab}$ from $T = T_1 = e(g_{p_1}, g_{p_1})^c$ 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 random choices of $a, b, c \in \mathbb{Z}_N$, and $R_1, R_2 \in \mathbb{G}_{p_2}$.

C.3 Our PKS Scheme

We describe our composite order PKS scheme derived from the composite order IBE scheme of Lewko and Waters [21]. This PKS scheme follows the transformation of Naor [8] that the private key of an IBE scheme can be the signature of a PKS scheme. Compared to the direct PKS scheme of Lewko and Waters, this PKS scheme has x, y in the private key to support signature aggregation and Y in the public key. Note that this scheme supports multi-user setting and public re-randomization since g, u, h are given in the public key. The PKS scheme is described as follows:

PKS.KeyGen(1^λ): This algorithm first generates the bilinear group $\mathbb{G}$ of composite order $N = p_1 p_2 p_3$ where p_1, p_2 and p_3 are random primes of bit size $\Theta(\lambda)$. It chooses random elements $g \in \mathbb{G}_{p_1}, Y \in \mathbb{G}_{p_3}$ and random exponents $x, y, \alpha \in \mathbb{Z}_N$. Then it outputs a private key $SK = (\alpha, x, y)$ and a public key as

$$PK = \left(g, u = g^x, h = g^y, Y, \Omega = e(g, g)^\alpha \right).$$

PKS.Sign(M, SK)**:** This algorithm takes as input a message $M \in \{0,1\}^k$ where $k < \lambda$ and a private key $SK = (\alpha, x, y)$. It selects a random exponent $r \in \mathbb{Z}_N$ and random elements $Y_1, Y_2 \in \mathbb{G}_{p_3}$. It then outputs a signature as

$$\sigma = \big(W_1 = g^{\alpha}(u^M h)^r Y_1,\ W_2 = g^r Y_2 \big).$$

PKS.Verify(σ, M, PK)**:** This algorithm takes as input a signature σ on a message M under a public key PK. It first chooses a random exponent $t \in \mathbb{Z}_N$ and verifies that

$$e(W_1, g^t) \cdot e(W_2, (u^M h)^t)^{-1} \stackrel{?}{=} \Omega^t.$$

If this equation holds, then it outputs 1. Otherwise, it outputs 0.

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

Proof. By the transformation of Naor, the security (indistinguishability under a chosen plaintext attacks) of a slightly modified Lewko-Waters IBE scheme that has the $Y \in \mathbb{G}_{p_3}$ in the public parameters instead of the master key is reduced to the security (unforgeability under a chosen message attack) of this PKS scheme. The security of the slightly modified Lewko-Waters IBE scheme is still secure under the Assumptions C.1, C.2, and C.3 since an element g_{p_3} is always given in the assumptions. This completes our proof. □

C.4 Our SAS Scheme

The basic idea of our SAS scheme is that it uses the randomness reuse technique of Lu et al. [22] for aggregation and then publicly re-randomizes the aggregate signature. The SAS scheme in composite order groups is described as follows:

SAS.Setup(1^{λ})**:** This algorithm first generates the bilinear group $\mathbb{G}$ of composite order $N = p_1 p_2 p_3$ where p_1, p_2 and p_3 are random primes of bit size $\Theta(\lambda)$. Next, it chooses random elements $g \in \mathbb{G}_{p_1}$ and $Y \in \mathbb{G}_{p_3}$. It publishes public parameters as $PP = (g, Y)$.

SAS.KeyGen(PP)**:** This algorithm takes as input the public parameters PP. It selects random exponents $x, y, \alpha \in \mathbb{Z}_N$ and sets $u = g^x, h = g^y$. Then it outputs a private key $SK = (\alpha, x, y)$ and a public key as $PK = (u, h, \Omega = e(g,g)^{\alpha})$.

SAS.AggSign$(AS', \mathbf{M}', \mathbf{PK}', M, SK)$**:** This algorithm takes as input an aggregate-so-far $AS' = (S_1', S_2')$ on messages $\mathbf{M}' = (M_1, \ldots, M_{l-1})$ under public keys $\mathbf{PK}' = (PK_1, \ldots, PK_{l-1})$ where $PK_i = (u_i, h_i, \Omega_i)$, a message $M \in \{0,1\}^k$ where $k < \lambda$, a private key $SK = (\alpha, x, y)$ with $PK = (u, h, \Omega)$ 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 selects a random exponent $r \in \mathbb{Z}_N$ and random elements $Y_1, Y_2 \in \mathbb{G}_{p_3}$, and outputs an aggregate signature as

$$AS = \big(S_1 = S_1' g^{\alpha} (S_2')^{xM+y} \cdot \prod_{i=1}^{l-1} (u_i^{M_i} h_i)^r (u^M h)^r Y_1,\ S_2 = S_2' \cdot g^r Y_2 \big).$$

SAS.AggVerify$(AS, \mathbf{M}, \mathbf{PK})$**:** This algorithm takes as input a sequential aggregate signature $AS = (S_1, S_2)$ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $PK_i = (u_i, h_i, \Omega_i)$. 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 $t \in \mathbb{Z}_N$ and verifies that

$$e(S_1, g^t) \cdot e(S_2, \prod_{i=1}^{l} (u_i^{M_i} h_i)^t)^{-1} \stackrel{?}{=} \prod_{i=1}^{l} \Omega_i^t.$$

If this equation holds, then it outputs 1. Otherwise, it outputs 0.

The aggregate signature $AS = (S_1, S_2)$ is a valid sequential aggregate signature on messages $\mathbf{M}'||M$ under public keys $\mathbf{PK}'||PK$ with randomness $\tilde{r} = r' + r$ where $S_2' = g^{r'} Y_2'$. The sequential aggregate signature has the following form

$$S_1 = \prod_{i=1}^{l} g^{\alpha_i} \prod_{i=1}^{l} (u_i^{M_i} h_i)^{\tilde{r}} \tilde{Y}_1, \; S_2 = g^{\tilde{r}} \tilde{Y}_2.$$

C.5 Security Analysis

Theorem C.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 a non-negligible advantage ε. A simulator $\mathcal{B}$ that forges the PKS scheme is given: a challenge public key $PK_{PKS} = (g, u, h, Y, \Omega = e(g,g)^{\alpha})$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows: $\mathcal{B}$ first constructs $PP = (g, Y)$ and $PK^* = (u, h, \Omega = e(g,g)^{\alpha})$ 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 sequential aggregate signature queries. If $\mathcal{A}$ requests the certification of a public key by providing a public key $PK_i = (u_i, h_i, \Omega_i)$ 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 $AS^* = (S_1^*, S_2^*)$ 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 σ^* by calling **SAS.AggVerify**. Additionally, the forged signature should not be trivial: the challenge public key PK^* must be in $\mathbf{PK}^*$, and each 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 $PK_i = (u_i, h_i, \Omega_i)$ 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 $W_1 = S_1^* \cdot \prod_{i=2}^{l} \left(g^{\alpha_j} (S_2^*)^{x_i M_i + y_i}\right)^{-1}$ and $W_2 = S_2^*$.

3. It outputs $\sigma = (W_1, W_2)$ 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 and the public key are correctly distributed. The sequential aggregate signatures is correctly distributed since this scheme does not check the order of aggregation. Finally, we can show that the result signature $\sigma = (W_1, W_2)$ 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

$$
\begin{aligned}
& e(W_1, g^t) \cdot e(W_2, (u^{M_1} h)^t)^{-1} \\
&= e(S_1^*, g^t) \cdot e(S_2^*, (u^{M_1} h)^t)^{-1} \cdot \prod_{i=2}^{l} e(g^{\alpha_i}, g^t)^{-1} \prod_{i=2}^{l} e((S_2^*)^{x_i M_i + y_i}, g^t)^{-1} \\
&= e(S_1^*, g^t) \cdot e(S_2^*, (u^{M_1} h)^t)^{-1} \cdot \prod_{i=2}^{l} (\Omega_i^t)^{-1} \prod_{i=2}^{l} e(S_2^*, (u_i^{M_i} h_i)^t)^{-1} \\
&= e(S_1^*, g^t) \cdot e(S_2^*, \prod_{i=1}^{l} (u_i^{M_i} h_i)^t)^{-1} \cdot \prod_{i=2}^{l} (\Omega_i^t)^{-1} = \prod_{i=1}^{l} \Omega_i^t \cdot \prod_{i=2}^{l} (\Omega_j^t)^{-1} = \Omega_1^t.
\end{aligned}
$$

This completes our proof. □