Aggregating CL-Signatures Revisited: Extended Functionality and Better Efficiency

Kwangsu Lee* Dong Hoon Lee† Moti Yung‡

Abstract

Aggregate signature is public-key signature that allows anyone to aggregate different signatures generated by different signers on different messages into a short aggregate signature. Although aggregate signatures have many applications like secure routing protocols and software module authentications, it is not easy to devise a suitable aggregate signature scheme that satisfies the conditions of real applications such that the size of public keys should be short, the size of aggregate signatures should be short, and the aggregate signing and aggregate verification should be efficient.

In this paper, we propose two aggregate signature schemes based on the Camenisch-Lysyanskaya signature scheme that sufficiently satisfy the conditions of real applications. At first, we construct an efficient sequential aggregate signature scheme and prove its security under the LRSW assumption without random oracles. The proposed scheme has the shortest size of public keys among the sequential aggregate signature schemes and very short size of aggregate signatures. Additionally, the aggregate verification of this scheme is very efficient since it only requires constant number of pairing operations and l number of exponentiations where l is the number of signers in an aggregate signature. Next, we construct an efficient synchronized aggregate signature scheme and prove its security under the LRSW assumption in the random oracle model. The proposed scheme has very short size of public keys and the shortest size of aggregate signatures among the synchronized aggregate signature schemes. The signing and aggregate verification is also very efficient since these requires constant number of pairing operations and l number of exponentiations.

Furthermore, a signer of our aggregate signature schemes can use two mode of aggregation "sequential" and "synchronized" at the same time just using the same private key and the public key since the private keys and the public keys of two schemes are the same.

Keywords: Public-key signature, Aggregate signature, CL signature, Bilinear pairing.

1 Introduction

Public-key signature (PKS) that provides the authentication of a message is a primitive tool for the constructions of cryptographic systems. However, it is not easy to devise a suitable PKS scheme that is efficient, secure, and flexible enough for a range of various applications. The CL signature scheme proposed by Camenisch and Lysyanskaya [16] is one of the pairing-based signature schemes [13, 11, 16, 28] that satisfies these conditions. The CL signature scheme was widely used for basic components of various cryptographic systems such as anonymous credential systems, group signature, RFID encryption, batch verification signature, ring signature, and aggregate signature [16, 3, 2, 15, 7, 26].

*Columbia University, NY, USA. kwangsu@cs.columbia.edu

†Korea University, Korea. donghlee@korea.ac.kr

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

Pubic-key aggregate signature (PKAS) that was introduced by Boneh, Gentry, Lynn, and Shacham is a special type of PKS that enables anyone to aggregate different signatures generated by different signers on different messages into a short aggregate signature [12]. Boneh et al. proposed the first full aggregate signature scheme in bilinear groups and proved its security under the CDH assumption in the random oracle model. After the introduction of aggregate signatures, other type of aggregate signatures such as sequential aggregate signatures and synchronized aggregate signatures were proposed [23, 19, 22, 5, 8, 25, 9, 1, 14, 18, 20, 21]. PKAS has many applications like secure routing protocols, public-key infrastructure systems, sensor network systems, software module authentications, and proxy signatures [12, 9, 1, 10].

PKAS can reduce the size of signers' signatures by using the aggregation technique. However, it cannot reduce the size of signers' public keys since the public keys are not aggregated. Thus the total information of a verifier who verifies an aggregate signature is still proportional to the number of signers in the aggregate signature, since the verifier should retrieve all public keys of signers from a certificate storage. Therefore it is a very important problem to reduce the size of public keys. Note that an ideal solution for this problem is to use identity-based aggregate signature (IBAS) that represents the public key of a signer as an identity string. However, IBAS requires an additional trusted authority, and the current IBAS schemes are only secure in the random oracle model [19, 9, 20]. To construct a PKAS scheme with short public keys without random oracles, Schröder proposed a sequential aggregate signature scheme with short public keys based on the CL signature scheme [26]. In the scheme of Schröder, the public key consists of two group elements and the aggregate signature consists of four group elements, but the aggregate verification algorithm requires l pairing operations and l exponentiations where l is the number of signers in the aggregate signature. Therefore, it is still required to construct an aggregate signature scheme with short public keys without random oracles that has an efficient aggregate verification algorithm.

1.1 Our Contributions

In this paper, we solve the problem of constructing a PKAS scheme that has short public keys, short aggregate signatures, and an efficient aggregate verification algorithm. We first propose an efficient sequential aggregate signature scheme based on the CL signature scheme and prove its security under the LRSW assumption without random oracles. In this scheme, the public key consists of just one group element and the aggregate signature consists of just three group element. The size of the public key is the shorted among all sequential aggregate schemes. The aggregate verification algorithm of our scheme is very efficient since it just requires five pairing operations and l exponentiations (or multi-exponentiations). Therefore our scheme satisfies the conditions of short public keys, short aggregate signatures, and efficient aggregate verification.

Next, we propose an efficient synchronized aggregate signature scheme based on the CL signature scheme and prove its security under the LRSW assumption in the random oracle model. In this scheme, the public key consists of just one group element and the aggregate signature consists of one group element and one integer. The size of the aggregate signature is the shortest among all synchronized aggregate signature schemes. The aggregate verification algorithm of this scheme is also very efficient since it just requires three pairing operations and l exponentiations (or multi-exponentiations). Additionally, our two aggregate signature schemes can be combined to a new aggregate signature scheme that supports sequential aggregation or synchronized aggregation since the public key and the private key of two schemes are the same.

For the constructions of the sequential aggregate signature scheme and the synchronized aggregate signature scheme using the CL signature scheme, we use two techniques such that the "randomness re-use" technique of Lu et al. [22] and a newly devised "public key sharing" technique. The "public key sharing" technique is to share the element Y of the public key with all signers by placing the public key element Y of the CL signature into the public parameters. In this case, the private key and the public key of a signer are x

and $X = g^x$ instead of x, y and $X = g^x, Y = g^y$ respectively. The signer then can generate the original CL signature as $\sigma = (A = g^r, B = Y^r, C = A^x B^{xM})$. Furthermore, the signer easily can generate a sequential aggregate signature as $\sigma_\Sigma = (A = g^r, B = Y^r, C = A^{\Sigma x_i} B^{\Sigma x_i M_i})$ since he only needs to aggregate the elements related to the public keys $\{X_i\}$ by using the "randomness re-use" technique. To construct a synchronized aggregate signature scheme from the sequential aggregate signature scheme, we force all signers to use the same elements A and B by using the synchronized time period information. That is, a signer first sets $A = H(0||w)$ and $B = H(1||w)$ where H is a hash function and w is a time period, and he generates a synchronized aggregate signature as $\sigma_\Sigma = (C = A^{\Sigma x_i} B^{\Sigma x_i M_i}, w)$.

1.2 Related Works

Full Aggregation. The notion of public-key aggregate signature (PKAS) was introduced by Boneh, Gentry, Lynn, and Shacham [12]. They proposed the first PKAS scheme in bilinear groups that supports full aggregation such that anyone can freely aggregate different signatures signed by different signers on different messages into a short aggregate signature [12]. The PKAS scheme of Boneh et al. requires l number of pairing operations in the aggregate verification algorithm where l is the number of signers in the aggregate signature. Bellare et al. modified the PKAS scheme of Boneh et al. to remove the restriction such that the message should be different by hashing a message with the public key of a signer [5].

Sequential Aggregation. The concept of sequential aggregate signature was introduced by Lysyanskaya, Micali, Reyzin, and Shacham [23]. In sequential aggregate signature, a signer can generate an aggregate signature by adding his signature to the previous aggregate signature that was received from a previous signer. Lysyanskaya et al. proposed a sequential PKAS scheme using certified trapdoor permutations, and they proved its security in random oracle models [23]. Neven proposed a sequential PKAS scheme that reduces not only the size of signatures but also the size of total information that is transmitted [25]. Boldyreva et al. proposed an identity-based sequential aggregate signature (IBSAS) scheme in bilinear groups and proved its security under an interactive assumption in the random oracle model [9]. Recently, Gerbush et al. showed that a modified IBSAS scheme of Boldyreva et al. in composite order bilinear groups can be proven under static assumptions in the random oracle model [20].

The first sequential PKAS scheme without random oracles was proposed by Lu et al. [22]. They constructed a sequential PKAS scheme based on the PKS scheme of Waters and proved its security under the CDH assumption without random oracles. However, this sequential PKAS scheme has a disadvantage such that the size of public keys is very long. To reduce the size of pubic keys in PKAS schemes, Schröder proposed a sequential PKAS scheme based on the CL signature scheme that has very short public key size, and he proved its security under the LRSW assumption without random oracles [26]. Lee et al. proposed efficient sequential PKAS scheme with short public keys and proved its security under static assumptions without random oracles [21].

In sequential PKAS schemes, a signer generally should verify the validity of the previous aggregate signature (the aggregate-so-far) from a previous signer before he adds his signature into the aggregate signature. To verify the previous aggregate signature, the signer should retrieve all public keys of previous signers and should run the aggregate verification algorithm. Thus verifying the previous aggregate signature is the most expensive operation in the aggregate signing algorithm. To solve this problem, sequential PKAS schemes that do not require to verify the previous aggregate signature were proposed [14, 18].

Synchronized Aggregation. The concept of synchronized aggregate signature was introduced by Gentry and Ramzan [19]. In synchronized aggregate signature, all signers have synchronized time information and individual signatures generated by different signers on the same time period can be aggregated into a

short aggregate signature. Gentry and Ramzan proposed an identity-based synchronized aggregate signature scheme in bilinear groups and proved its security under the CDH assumption in the random oracle model [19]. Ahn et al. proposed an efficient synchronized PKAS scheme based on the PKS scheme of Hohenberger and Waters and proved its security under the CDH assumption without random oracles [1].

Interactive Aggregation. Interactive aggregate signature is aggregate signature such that a signer generates an aggregate signature after having interactive communications with other signers through a broadcast channel. Bellare and Neven proposed an identity-based multi-signature scheme under the RSA assumption in the random oracle model [6], and Bagherzandi and Jareki proposed an identity-based aggregate signature scheme and proved its security under the RSA assumption in the random oracle model [4]. However, the interactive communications between signers are expensive, and the heavy message transmission between signers eliminates the advantage of signature aggregation.

2 Preliminaries

In this section, we first define the public key signature and its security model. Next, we define bilinear groups, and introduce the LRSW assumption and the CL signature scheme.

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 a security parameter 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 $\mathsf{Adv}_{\mathcal{A}}^{PKS} = \Pr[\mathcal{C} = 1]$ 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 Bilinear Groups

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

We say that $\mathbb{G}, \mathbb{G}_T$ are bilinear groups if the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable.

2.3 Complexity Assumption

The security of our aggregate signature schemes is based on the following LRSW assumption. The LRSW assumption was introduced by Lysyanskaya, et al. [24] and it is secure under the generic group model defined by Shoup [27].

Assumption 2.1 (LRSW)**.** *Let $\mathcal{G}$ be an algorithm that on input the security parameter 1^λ, outputs the parameters for a bilinear group as $(p, \mathbb{G}, \mathbb{G}_T, e, g)$. Let $X, Y \in \mathbb{G}$ such that $X = g^x, Y = g^y$ for some $x, y \in \mathbb{Z}_p$. Let $O_{X,Y}(\cdot)$ be an oracle that on input a value $M \in \mathbb{Z}_p$ outputs a triple (a, a^y, a^{x+Mxy}) for a randomly chosen $a \in \mathbb{G}$. Then for all probabilistic polynomial time adversaries $\mathcal{A}$,*

$$\begin{aligned}\Pr[&(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda), x \leftarrow \mathbb{Z}_p, y \leftarrow \mathbb{Z}_p, X = g^x, Y = g^y, \\ &(M, a, b, c) \leftarrow \mathcal{A}^{O_{X,Y}(\cdot)}(p, \mathbb{G}, \mathbb{G}_T, e, g, X, Y) : \\ &M \notin Q \wedge M \in \mathbb{Z}_p^* \wedge a \in \mathbb{G} \wedge b = a^y \wedge c = a^{x+Mxy}] < 1/poly(\lambda)\end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $O_{X,Y}(\cdot)$.

2.4 The CL Signature Scheme

The CL signature scheme is a public-key signature scheme that was proposed by Camenisch and Lysyanskaya and the security was proven under the LRSW assumption without random oracles [16]. Although the security of the CL signature scheme is based on the interactive assumption, it is widely used for the constructions of various cryptosystems such as anonymous credential, group signature, ring signature, batch verification, and aggregate signature [24, 16, 7, 15, 26].

PKS.KeyGen(1^λ): The key generation algorithm first generates the bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p of bit size $\Theta(\lambda)$. Let g be the generator of $\mathbb{G}$. It selects two random exponents $x, y \in \mathbb{Z}_p$ and sets $X = g^x, Y = g^y$. It outputs a private key as $SK = (x, y)$ and a public key as $PK = (p, \mathbb{G}, \mathbb{G}_T, e, g, X, Y)$.

PKS.Sign(M, SK): The signing algorithm takes as input a message $M \in \mathbb{Z}_p^*$ and a private key $SK = (x, y)$. It selects a random element $A \in \mathbb{G}$ and computes $B = A^y$, $C = A^x B^{xM}$. It outputs a signature as $\sigma = (A, B, C)$.

PKS.Verify(σ, M, PK): The verification algorithm takes as input a signature $\sigma = (A, B, C)$ on a message $M \in \mathbb{Z}_p^*$ under a public key $PK = (p, \mathbb{G}, \mathbb{G}_T, e, g, X, Y)$. It verifies that $e(A, Y) \stackrel{?}{=} e(B, g)$ and $e(C, g) \stackrel{?}{=} e(A, X) \cdot e(B, X)^M$. If these equations hold, then it outputs 1. Otherwise, it outputs 0.

Theorem 2.2 ([16])**.** *The CL signature scheme is existentially unforgeable under a chosen message attack if the LRSW assumption holds.*

3 Sequential Aggregate Signature

In this section, we first define the sequential aggregate signature and its security model. After that, we propose an efficient sequential aggregate signature scheme based on the CL signature scheme, and prove its security under the LRSW assumption.

3.1 Definitions

Sequential aggregate signature (SeqAS) is a special type of public-key aggregate signature (PKAS) that allows each signer to sequentially add his signature on a different message to the aggregate signature [23]. That is, a signer with an index i receives an aggregate signature σ'_Σ from the signer of an index $i-1$, and he generates a new aggregate signature σ_Σ by aggregating his signature on a message M to the received aggregate signature. The resulting aggregate signature has the same size of the previous aggregate signature.

Formally, a sequential aggregate signature (SeqAS) scheme consists of four PPT algorithms **Setup**, **KeyGen**, **AggSign**, and **AggVerify**, which are defined as follows:

- **Setup**(1^λ). The setup algorithm takes as input a security parameter 1^λ and outputs public parameters PP.
- **KeyGen**(PP). The key generation algorithm takes as input the public parameters PP, and outputs a public key PK and a private key SK.
- **AggSign**$(\sigma'_\Sigma, \mathbf{M}, \mathbf{PK}, M, SK, PP)$. The aggregate signing algorithm takes as input an aggregate-so-far σ'_Σ on messages $\mathbf{M} = (M_1, \ldots, M_k)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_k)$, a message M, and a private key SK with PP, and outputs a new aggregate signature σ_Σ.
- **AggVerify**$(\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP)$. The aggregate verification algorithm takes as input an aggregate signature σ_Σ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ and the public parameters PP, and outputs either 1 or 0 depending on the validity of the aggregate signature.

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

The security model of SeqAS was introduced by Lysyanskaya et al. [23]. In this paper, we follow the security model that was proposed by Lu et al. [22]. The security model of Lu et al. is a more restricted model that requires the adversary to generate other signers' public keys and private keys except the challenge signer's key. To ensure the correct generation of public keys and private keys, the adversary should submit the corresponding private keys of the public keys to the challenger before using the public keys. A realistic solution of this is for the signer to prove that he knows the corresponding private key of the public key by using zero-knowledge proofs when he requests the certification of his public key.

In the security model of SeqAS, the public parameters and the challenge public key PK^* are given to the adversary. The adversary can request the certification of a public key through a certification query by providing a public key and a private key. He also can request a sequential aggregate signature query on a message under the challenge public key by providing an aggregate-so-far that was generated from the certified public keys. Finally, he outputs a forged aggregate signatures σ^*_Σ. The adversary breaks the SeqAS scheme if the forged sequential aggregate signature is valid and non-trivial.

Formally, 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 key-pair list *KeyList* 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 *KeyList* if the key pair is a valid one.

Signature Query: $\mathcal{A}$ adaptively requests a sequential aggregate signature (by providing an aggregate-so-far σ'_Σ 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 σ_Σ.

Output: Finally (after a sequence of the above queries), $\mathcal{A}$ outputs a forged sequential aggregate signature σ^*_Σ 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) $\mathbf{AggVerify}(\sigma^*_\Sigma, \mathbf{M}^*, \mathbf{PK}^*, PP) = 1$, 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 $KeyList$, 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 $\mathsf{Adv}^{SeqAS}_{\mathcal{A}} = \Pr[\mathcal{C} = 1]$ where the probability is taken over all the randomness of the experiment. A SeqAS 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.

3.2 Construction

We first describe the design idea of our SeqAS scheme. To construct a SeqAS scheme, we use the "public key sharing" technique such that the element Y in the public key of the original CL signature scheme can be shared with all signers. The modified CL signature scheme that shares the element Y of the public key is described as follows: The setup algorithm of the modified CL signature scheme first publishes the public parameters that contain the description of bilinear groups and an element Y. Each signer generates a private key $x \in \mathbb{Z}_p$ and a public key $X = g^x$. A signer who has the private key x of the public key X can generates an original CL signature $\sigma = (A, B, C)$ on a message M just using the private key x and a random r as $A = g^r, B = Y^r$, and $C = A^x B^{xM}$ since the element Y is given in the public parameters. This modified CL signature scheme is still secure under the LRSW assumption.

We construct a SeqAS scheme based on the modified CL signature scheme that supports "public key sharing" by using the "randomness re-use" technique of Lu et al. [22]. It is easy to sequentially aggregate signatures if the element Y is shared with all signers since we only need to consider the aggregation of the $\{X\}$ values of signers instead of the $\{X, Y\}$ values of signers. For instance, the first signer who has a private key x_1 generates a signature $\sigma_1 = (A_1, B_1, C_1)$ on a message M_1 as $A_1 = g^{r_1}, B_1 = Y^{r_1}$, and $C_1 = (g^{r_1})^{x_1}(Y^{r_1})^{x_1 M_1}$. The second signer with a private key x_2 generate a sequential aggregate signature $\sigma_2 = (A_2, B_2, C_2)$ on a message M_2 as $A_2 = A_1, B_2 = B_1$, and $C_2 = C_1 (A_1)^{x_2} (B_1)^{x_2 M_2}$ by using the "randomness re-use" technique. Therefore a sequential aggregate signature of signers is formed as $\sigma_\Sigma = (A = g^r, B = Y^r, C = A^{\sum x_i} B^{\sum x_i M_i})$. Additionally, each signer should re-randomize the aggregate signature to prevent a simple attack.

Our sequential aggregate signature (SeqAS) scheme is described as follows.

SeqAS.Setup(1^λ): The setup algorithm first generates the bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p of bit size $\Theta(\lambda)$. Let g be the generator of $\mathbb{G}$. It chooses a random element $Y \in \mathbb{G}$ and outputs public parameters as $PP = (p, \mathbb{G}, \mathbb{G}_T, e, g, Y)$.

SeqAS.KeyGen(PP): The key generation algorithm takes as input the public parameters PP. It selects a random exponent $x \in \mathbb{Z}_p$ and sets $X = g^x$. Then it outputs a private key as $SK = x$ and a public key as $PK = X$.

SeqAS.AggSign$(\sigma'_\Sigma, \mathbf{M}', \mathbf{PK}', M, SK, PP)$: The aggregate signing algorithm takes as input an aggregate-so-far $\sigma'_\Sigma = (A', B', C')$ on messages $\mathbf{M}' = (M_1, \ldots, M_k)$ under public keys $\mathbf{PK}' = (PK_1, \ldots, PK_k)$ where

$PK_i = X_i$, a message $M \in \mathbb{Z}_p^*$, and a private key $SK = x$ with PP. It first checks the validity of σ'_Σ by calling **AggVerify**$(\sigma'_\Sigma, \mathbf{M}', \mathbf{PK}', PP)$. If σ'_Σ is not valid, then it halts. It checks that the public key PK of SK does not already exist in $\mathbf{PK}'$. If the public key already exists, then it halts. Note that if $k = 0$, then $\sigma'_\Sigma = (1, Y, 1)$. It selects a random exponent $r \in \mathbb{Z}_p$ and computes

$$A = (A')^r,\ B = (B')^r,\ C = \left(C' \cdot (A')^x \cdot (B')^{xM}\right)^r.$$

It outputs an aggregate signature as $\sigma_\Sigma = (A, B, C)$.

SeqAS.AggVerify$(\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP)$: The aggregate verification algorithm takes as input an aggregate signature $\sigma_\Sigma = (A, B, C)$ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $PK_i = X_i$. 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 $\sigma_\Sigma = (1, Y, 1)$, 0 otherwise. Next, it verifies that

$$e(A, Y) \stackrel{?}{=} e(B, g) \ \text{ and } \ e(C, g) \stackrel{?}{=} e(A, \prod_{i=1}^{l} X_i) \cdot e(B, \prod_{i=1}^{l} X_i^{M_i}).$$

If these equations hold, then it outputs 1. Otherwise, it outputs 0.

A sequential aggregate signature $\sigma_\Sigma = (A, B, C)$ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ has the following form

$$A = g^r,\ B = Y^r,\ C = \left(g^r\right)^{\sum_{i=1}^{k} x_i} \left(Y^r\right)^{\sum_{i=1}^{k} x_i M_i}$$

where $PK_i = X_i = g^{x_i}$.

3.3 Security Analysis

We prove that our SeqAS scheme is existentially unforgeable under a chosen message attack if the CL signature scheme is existentially unforgeable under a chosen message attack. Therefore, our SeqAS scheme is also existentially unforgeable under a chosen message attack under the LRSW assumption since the security of the CL signature scheme is proven under the LRSW assumption,

Theorem 3.1. *The above SeqAS scheme is existentially unforgeable under a chosen message attack if the CL signature scheme is existentially unforgeable under a chosen message attack. That is, for any PPT adversary $\mathcal{A}$ for the above SeqAS scheme, there exists a PPT algorithm $\mathcal{B}$ for the CL signature scheme such that $Adv_{\mathcal{A}}^{SeqAS}(\lambda) \leq Adv_{\mathcal{B}}^{CL}(\lambda)$.*

Proof. The main idea of the security proof is that the aggregated signature of our SeqAS scheme is independent of the order of aggregation, and the simulator of the SeqAS scheme possesses the private keys of all signers except the private key of the challenge public key. That is, if the adversary request a sequential aggregate signature, then the simulator first obtains a CL signature from the target scheme's signing oracle and runs the aggregate signing algorithm to generates a sequential aggregate signature. If the adversary finally outputs a forged sequential aggregate signature that is non-trivial, then the simulator extracts the CL signature of the challenge public key from the forged aggregate signature by using the private keys of other signers.

Suppose there exists an adversary $\mathcal{A}$ that forges the above SeqAS scheme with non-negligible advantage ε, then a simulator $\mathcal{B}$ that forges the CL signature scheme is given: a challenge public key $PK_{CL} = (p, \mathbb{G}, \mathbb{G}_T, e, g, X, Y)$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ first constructs $PP = (p, \mathbb{G}, \mathbb{G}_T, e, g, Y)$ and $PK^* = X$ from PK_{CL}. Next, it initializes a key-pair list *KeyList* as an empty one and gives PP and PK^* to $\mathcal{A}$.

Certification Query: $\mathcal{A}$ adaptively requests the certification of a public key by providing a public key $PK_i = X_i$ and its private key $SK_i = x_i$. $\mathcal{B}$ checks the private key and adds the key pair (PK_i, SK_i) to *KeyList*.

Signature Query: $\mathcal{A}$ adaptively requests a sequential aggregate signature query by providing an aggregate-so-far σ'_Σ on messages $\mathbf{M}' = (M_1, \ldots, M_k)$ under public keys $\mathbf{PK}' = (PK_1, \ldots, PK_k)$, and a message M to sign under the challenge private key of PK^*. $\mathcal{B}$ proceeds the aggregate signature query as follows:

1. It first checks that the signature σ'_Σ is valid and that each public key in $\mathbf{PK}'$ exits in *KeyList*.
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 $1 \leq i \leq k$, it constructs an aggregate signature on message M_i using **SeqAS.AggSign** since it knows the private key that corresponds to PK_i. The resulting 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 σ_Σ to $\mathcal{A}$.

Output: $\mathcal{A}$ outputs a forged aggregate signature $\sigma^*_\Sigma = (A^*, B^*, C^*)$ 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^*_Σ by calling **SeqAS.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 $PK_i = X_i$ from $\mathbf{PK}^*$, and it retrieves the private key $SK_i = x_i$ of PK_i from *KeyList*. It then computes
$$A = A^*, \;\; B = B^*, \;\; C = C^* \cdot \left((A^*)^{\sum_{i=2}^{l} x_i} (B^*)^{\sum_{i=2}^{l} x_i M_i} \right)^{-1}.$$
3. It outputs $\sigma^* = (A, B, C)$ on a message $M^* = M_1$ as a non-trivial forgery of the CL signature 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 distribution of the sequential aggregate signatures is correct since this scheme does not check the order of aggregation. Finally, we can show that the resulting signature $\sigma^* = (A, B, C)$ of the simulator is a valid signature for the CL signature scheme on the message M_1 under the public key PK^* since it satisfies the following equation:

$$\begin{aligned}
e(C, g) &= e(C^* \cdot \left((A^*)^{\sum_{i=2}^{l} x_i} (B^*)^{\sum_{i=2}^{l} x_i M_i} \right)^{-1}, g) \\
&= e((A^*)^{\sum_{i=1}^{l} x_i} (B^*)^{\sum_{i=1}^{l} x_i M_i} \cdot (A^*)^{-\sum_{i=2}^{l} x_i} (B^*)^{-\sum_{i=2}^{l} x_i M_i}, g) \\
&= e((A^*)^{x_1} (B^*)^{x_1 M_1}, g) \\
&= e(A^*, g^{x_1}) \cdot e(B^*, g^{x_1 M_1}) \\
&= e(A, X) \cdot e(B, X^{M^*}).
\end{aligned}$$

This completes our proof. □

3.4 Discussions

Efficiency. The public key of our SeqAS scheme consists of just one group element and the aggregate signature consists of three group elements, since the public key element Y of the CL signature scheme is moved to the public parameters of our scheme. The aggregate signing algorithm of our SeqAS scheme requires one aggregate verification and five exponentiations, and the aggregate verification algorithm requires five pairing operations and l exponentiations where l is the number of signers in the aggregate signature. In the SeqAS scheme of Schröder, the public key consists of two group elements, the aggregate signature consists of four group elements, and the aggregate verification algorithm requires l pairing operations and l exponentiations. Therefore, our SeqAS scheme is more efficient than the SeqAS scheme of Schröder.

Asymmetric Bilinear Groups. We can use asymmetric bilinear groups instead of symmetric bilinear groups to reduce the size of aggregate signatures. Let $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ be the cyclic groups of prime order p. We say that $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ are asymmetric bilinear groups if there exists the bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ that has bilinearity and non-degeneracy properties. The SeqAS scheme in asymmetric bilinear groups is described as follows: the public parameters is $PP = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}, Y, \hat{Y})$, the private key and the public key are $SK = x$ and $PK = \hat{X} \in \hat{\mathbb{G}}$ respectively, and the aggregate signature is $\sigma_\Sigma = (A = g^r, B = Y^r, C = A^{\Sigma x_i} B^{\Sigma x_i M_i}) \in \mathbb{G}^3$. For instance, if we instantiate the asymmetric bilinear groups using the 175-bit MNT curve with embedding degree 6 to guarantee 80-bit security level, the size of public key is 525 bit and the size of aggregate signature is 525 bit. In the 175-bit MNT curve, the SeqAS scheme of Lu et al. [22] has 113 kilo-bits size of the public key and 350 bit size of the aggregate signature, and the SeqAS scheme of Schröder [26] has 1050 bit size of the public key and 700 bit size of the aggregate signature.

Public-Key Signature. We can easily derive a new PKS scheme from our SeqAS scheme. Compared to the CL signature scheme, the new PKS scheme has an advantage such that the public key just consists of one group element X instead of two group elements X, Y since the element Y is moved to the public parameters. A signer who has a private key x can generate a signature $\sigma = (A, B, C)$ as $A = g^r, B = Y^r$, and $C = A^x B^{xM}$. A verifier can verify the signature by checking that $e(A, Y) = e(B, g)$ and $e(C, g) = e(A, X) \cdot e(B, X^M)$. This new PKS scheme is also secure under the LRSW assumption.

4 Synchronized Aggregate Signature

In this section, we first define the synchronized aggregate signature and its security model. Next, we propose an efficient synchronized aggregate signature scheme based on the CL signature scheme, and prove its security in the random oracle model under the LRSW assumption.

4.1 Definitions

Synchronized aggregate signature (SyncAS) is a special type of public-key aggregate signature (PKAS) that allows anyone to aggregate signer's signatures on different messages with a same time period into a short aggregate signature if all signers have the synchronized time period information like a clock [19, 1]. In SyncAS scheme, each signer has a synchronized time period or has an access to public time information. Each signer can generate an individual signature on a message M and a time period w. Note that the signer can generate just one signature per one time period. After that, anyone can aggregate individual signatures of other signers into a short aggregate signature σ_Σ if the individual signatures are generated on the same time period w. The resulting aggregate signature has the same size of the individual signature.

Formally, a synchronized aggregate signature (SyncAS) scheme consists of six PPT algorithms **Setup**, **KeyGen**, **Sign**, **Verify**, **Aggregate**, and **AggVerify**, which are defined as follows:

- **Setup**(1^λ). The setup algorithm takes as input a security parameter 1^λ and outputs public parameters PP.
- **KeyGen**(PP). The key generation algorithm takes as input the public parameters PP, and outputs a public key PK and a private key SK.
- **Sign**(M, w, SK, PP). The signing algorithm takes as input a message M, a time period w, and a private key SK with PP, and outputs an individual signature σ.
- **Verify**(σ, M, PK, PP). The verification algorithm takes as input a signature σ on a message M under a public key PK, and outputs either 1 or 0 depending on the validity of the signature.
- **Aggregate**$(\mathbf{S}, \mathbf{M}, \mathbf{PK})$. The aggregation algorithm takes as input individual signatures $\mathbf{S} = (\sigma_1, \ldots, \sigma_l)$ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$, and outputs an aggregate signature σ_Σ.
- **AggVerify**$(\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP)$. The aggregate verification algorithm takes as input an aggregate signature σ_Σ 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 aggregate signature.

The correctness requirement is that for each PP output by **Setup**, for all (PK, SK) output by **KeyGen**, any M, we have that **AggVerify**(**Aggregate**$(\mathbf{S}, \mathbf{M}, \mathbf{PK}), \mathbf{M}, \mathbf{PK}, PP) = 1$ where $\mathbf{S}$ is individual signatures on messages $\mathbf{M}$ under public keys $\mathbf{PK}$.

The security model of SyncAS was introduced by Gentry and Ramzan [19]. In this paper, we follow the security model that was proposed by Ahn et al. [1]. The security model of Ahn et al. is a more restricted model that requires the adversary to generate other signers' public keys and private keys except the challenge signer's key. To ensure the correct generation of public keys and private keys, the adversary should submit the private key of the public key, or he should prove that he knows the corresponding private key by using zero-knowledge proofs.

In the security model of SyncAS, the public parameters and the challenge public key PK^* are given to the adversary at first. The adversary can request the certification of a public key through a certification query by providing the public key and the corresponding private key. He also can request a signature query on a message M and a time period w that was not used before under the challenge public key. Finally, he outputs a forged synchronized aggregate signature σ_Σ^*. The adversary breaks the SyncAS scheme if the forged synchronized aggregate signature of the adversary is valid and non-trivial.

Formally, 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 key-pair list *KeyList* 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 *KeyList* if the key pair is a valid one.

Hash Query: $\mathcal{A}$ adaptively requests a hash on a string for various hash functions, and receives a hash value.

Signature Query: $\mathcal{A}$ adaptively requests a signature on a message M and a time period w that was not used before to sign under the challenge public key PK, and receives an individual signature σ.

Output: Finally (after a sequence of the above queries), $\mathcal{A}$ outputs a forged synchronized aggregate signature σ_Σ^* 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) $\mathbf{AggVerify}(\sigma_\Sigma^*, \mathbf{M}^*, \mathbf{PK}^*, PP) = 1$, 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 $KeyList$, 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 signing oracle.

The advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}_{\mathcal{A}}^{SyncAS} = \Pr[\mathcal{C} = 1]$ where the probability is taken over all the randomness of the experiment. A SyncAS 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.

4.2 Construction

We first describe the design idea of our SyncAS scheme. In the previous section, we proposed a modified CL signature scheme that shares the element Y in the public parameters. The signature of this modified CL signature scheme is formed as $\sigma = (A = g^r, B = Y^r, C = A^x B^{xM})$. If we can force signers to use the same $A = g^r$ and $B = Y^r$ in signatures, then we easily obtain an aggregate signature as $\sigma_\Sigma = (A = g^r, B = Y^r, C = A^{\Sigma x_i} B^{\Sigma x_i M_i})$ by just multiplying individual signatures of signers. In synchronized aggregate signatures, it is possible to force signers to use the same A and B since all signers have the same time period w. Therefore, each signer first sets $A = H(0||w)$ and $B = H(1||w)$ using the hash function H and the time period w, and then he generates an individual signature $\sigma = (C = A^x B^{xM}, w)$.

Our synchronized aggregate signature (SyncAS) scheme is described as follows.

SyncAS.Setup(1^λ): The setup algorithm first generates the bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p of bit size $\Theta(\lambda)$. Let g be the generator of $\mathbb{G}$. It chooses two hash functions $H_1 : \{0,1\} \times \mathcal{W} \to \mathbb{G}$ and $H_2 : \mathcal{M} \times \mathcal{W} \to \mathbb{Z}_p$. It outputs public parameters as $PP = (p, \mathbb{G}, \mathbb{G}_T, e, g, H_1, H_2)$.

SyncAS.KeyGen(PP): The key generation algorithm takes as input the public parameters PP. It selects a random exponent $x \in \mathbb{Z}_p$ and sets $X = g^x$. Then it outputs a private key as $SK = x$ and a public key as $PK = X$.

SyncAS.Sign(M, w, SK, PP): The signing algorithm takes as input a message $M \in \mathcal{M}$, a time period $w \in \mathcal{W}$, and a private key $SK = x$ with PP. It first sets $A = H_1(0||w), B = H_1(1||w), h = H_2(M||w)$ and computes $C = A^x B^{xh}$. It outputs a signature as $\sigma = (C, w)$.

SyncAS.Verify(σ, M, PK, PP): The verification algorithm takes as input a signature $\sigma = (C, w)$ on a message M under a public key $PK = X$. It first checks that the public key has been certified. If these checks fail, then it outputs 0. Next, it sets $A = H_1(0||w), B = H_1(1||w), h = H_2(M||w)$ and verifies that $e(C, g) \stackrel{?}{=} e(AB^h, X)$. If this equation holds, then it outputs 1. Otherwise, it outputs 0.

SyncAS.Aggregate$(\mathbf{S}, \mathbf{M}, \mathbf{PK}, PP)$: The aggregation algorithm takes as input signatures $\mathbf{S} = (\sigma_1, \ldots, \sigma_l)$ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $\sigma_i = (C_i', w_i')$ and $PK_i = X_i$. It first checks that that w_1' is equal to w_i' for $i = 2$ to l. If it fails, it halts. Next, it sets $w = w_1'$ and computes $C = \prod_{i=1}^{l} C_i'$. It outputs an aggregate signature as $\sigma_\Sigma = (C, w)$.

SyncAS.AggVerify$(\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP)$: The aggregate verification algorithm takes as input an aggregate signature $\sigma_\Sigma = (C, w)$ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ where $PK_i = X_i$. 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. Next, it sets $A = H_1(0||w), B = H_1(1||w), \{h_i = H_2(M_i||w)\}$ and verifies that

$$e(C, g) \stackrel{?}{=} e(A, \prod_{i=1}^{l} X_i) \cdot e(B, \prod_{i=1}^{l} X_i^{h_i}).$$

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

A synchronized aggregate signature $\sigma_\Sigma = (C, w)$ on messages $\mathbf{M} = (M_1, \ldots, M_l)$ under public keys $\mathbf{PK} = (PK_1, \ldots, PK_l)$ has the following form

$$C = H_1(0||w)^{\sum_{i=1}^{l} x_i} H_1(1||w)^{\sum_{i=1}^{l} x_i H_3(M_i||w)}$$

where $PK_i = X_i = g^{x_i}$.

4.3 Security Analysis

We prove that our SyncAS scheme is existentially unforgeable under a chosen message attack if the CL signature scheme is existentially unforgeable under a chosen message attack. Therefore, our SyncAS scheme is also existentially unforgeable under a chosen message attack under the LRSW assumption since the CL signature is proven under the LRSW assumption.

Theorem 4.1. *The above SyncAS scheme is existentially unforgeable under a chosen message attack if the CL signature scheme is existentially unforgeable under a chosen message attack. That is, for any PPT adversary $\mathcal{A}$ for the above SyncAS scheme, there exists a PPT algorithm $\mathcal{B}$ for the CL signature scheme such that $Adv_{\mathcal{A}}^{SyncAS}(\lambda) \leq q_{H_1} q_{H_2} \cdot Adv_{\mathcal{B}}^{CL}(\lambda)$ where q_{H_1} and q_{H_2} are the maximum number of hash queries.*

Proof. The main idea of the security proof is that the random oracle model supports the programmability of hash functions, the adversary can request just one signature query per one time period in this security model, and the simulator possesses the private keys of all signers except the private key of the challenge public key. In the proof, the simulator first guesses the time period w' of the forged synchronized aggregate signature and selects a random query index k of the hash function H_2. After that, if the adversary requests a signature query on a message M and a time period w such that $w \neq w'$, then he can easily generates the signature by using the programmability of the random oracle model. If the adversary requests a signature query for the time period $w = w'$, then he can generates the signature if the query index i is equal to the index k. Otherwise, the simulator should abort the simulation. Finally, if the adversary outputs a forged synchronized aggregate signature that is non-trivial on the time period w', then the simulator extracts the CL signature of the challenge public key from the forged aggregate signature by using the private keys of other signers.

Suppose there exists an adversary $\mathcal{A}$ that forges the above SyncAS scheme with non-negligible advantage ε, then a simulator $\mathcal{B}$ that forges the CL signature scheme is given: a challenge public key $PK_{CL} = (p, \mathbb{G}, \mathbb{G}_T, e, g, X, Y)$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ first constructs $PP = (p, \mathbb{G}, \mathbb{G}_T, e, g, H_1, H_2)$ and $PK^* = X$ from PK_{CL}. It chooses a random value $h' \in \mathbb{Z}_p^*$ and queries **PKS.Sign** to obtain $\sigma' = (A', B', C')$. Let q_{H_1} and q_{H_2} be the maximum number of H_1 and H_2 hash queries respectively. It chooses a random index k such that $1 \leq k \leq q_{H_2}$ and guesses a random time period $w' \in \mathcal{W}$ of the forged signature. Next, it initializes a key-pair list *KeyList*, hash lists H_1-*List*, H_2-*List* as an empty one and gives PP and PK^* to $\mathcal{A}$.

Certification Query: $\mathcal{A}$ adaptively requests the certification of a public key by providing a public key $PK_i = X_i$ and its private key $SK_i = x_i$. $\mathcal{B}$ checks the private key and adds the key-pair (PK_i, SK_i) to *KeyList*.

Hash Query: $\mathcal{A}$ adaptively requests a hash query for H_1 and H_2 respectively. If this is a H_1 hash query on a bit b and a time period w_i, then $\mathcal{B}$ treats the query as follows:

- If $b = 0$ and $w_i \neq w'$, then it selects a random exponent $r_i \in \mathbb{Z}_p$ and sets $H_1(0||w_i) = g^{r_i}$.
- If $b = 0$ and $w_i = w'$, then it sets $H_1(0||w_i) = A'$.
- If $b = 1$ and $w_i \neq w'$, then it selects a random exponent $s_i \in \mathbb{Z}_p$ and sets $H_1(1||w_i) = g^{s_i}$.
- If $b = 1$ and $w_i = w'$, then it sets $H_1(1||w_i) = B'$.

If this is a H_2 hash query on a message M_i and a time period w_j, then $\mathcal{B}$ treats the query as follows:

- If $i \neq k$ or $w_j \neq w'$, then it selects a random value $h_{i,j} \in \mathbb{Z}_p$ and sets $H_2(M_i||w_j) = h_{i,j}$.
- If $i = k$ and $w_j = w'$, then it sets $H_2(M_i||w_j) = h'$.

Note that the simulator keeps the tuple $(b_i, w_i, r_i, H_1(b_i||w_i))$ in H_1-*List* and the tuple $(M_i, w_j, h_{i,j})$ in H_2-*List*.

Signature Query: $\mathcal{A}$ adaptively requests a signature query by providing a message M_i and a time period w_j to sign under the challenge private key of PK^*. $\mathcal{B}$ proceeds the signature query as follows:

- If $w_i \neq w'$, then it responses $\sigma_{i,j} = (X^{r_i} X^{s_i h_{i,j}}, w_j)$ where r_i, s_i, and $h_{i,j}$ are retrieved from the H_1-*List* and H_2-*List*.
- If $w_i = w'$ and $i = k$, then it responses $\sigma_{i,j} = (C', w_j)$.
- If $w_i = w'$ and $i \neq k$, it aborts the simulation.

Output: $\mathcal{A}$ outputs a forged aggregate signature $\sigma_\Sigma^* = (C^*, w^*)$ 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. It checks the validity of σ_Σ^* by calling **SyncAS.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. If $w^* \neq w'$, then it aborts the simulation since it fails to guess the correct state value.
3. For each $2 \leq i \leq l$, it retrieves the private key $SK_i = x_i$ of PK_i from *KeyList* and sets $h_{i,*} = H_2(M_i||w^*)$. Next, it computes

$$A = A^*,\ B = B^*,\ C = C^* \cdot \left(\left(A^*\right)^{\sum_{i=2}^{l} x_i} \left(B^*\right)^{\sum_{i=2}^{l} x_i h_{i,*}} \right)^{-1}.$$

4. If $H_2(M_1||w^*) = h'$, then it also aborts the simulation.
5. It outputs $\sigma^* = (A, B, C)$ on a message $h_{1,*}$ as a non-trivial forgery of the CL signature scheme since $h_{1,*} \neq h'$ where $h_{1,*} = H_2(M_1||w^*)$.

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 distribution of the signatures is also correct. Next, we show that the resulting signature $\sigma^* = (A, B, C)$ of the simulator is a valid signature for the CL signature scheme on the message $h_{1,*} \neq h'$ under the public key PK^* since it satisfies the following equation:

$$\begin{aligned}
e(C, g) &= e(C^* \cdot \left((A^*)^{\sum_{i=2}^{l} x_i} (B^*)^{\sum_{i=2}^{l} x_i H_2(M_i||w^*)}\right)^{-1}, g) \\
&= e((A^*)^{\sum_{i=1}^{l} x_i} (B^*)^{\sum_{i=1}^{l} x_i h_{i,*}} \cdot (A^*)^{-\sum_{i=2}^{l} x_i} (B^*)^{-\sum_{i=2}^{l} x_i h_{i,*}}, g) \\
&= e((A^*)^{x_1} (B^*)^{x_1 h_{1,*}}, g) \\
&= e(A^*, g^{x_1}) \cdot e(B^*, g^{x_1 h_{1,*}}) \\
&= e(A', X) \cdot e(B', X^{h_{1,*}}).
\end{aligned}$$

We now analyze the success probability of the simulator $\mathcal{B}$. At first, $\mathcal{B}$ succeeds the simulation if he does not abort in the simulation of signature query and he correctly guesses the time period w^* such that $w^* = w'$ in the forged aggregate signature from the adversary $\mathcal{A}$. $\mathcal{B}$ aborts the simulation of signature query if the time period w' is given from $\mathcal{A}$ and he incorrectly guessed the index k since he cannot generate a signature. Thus $\mathcal{B}$ succeeds the simulation of signature query at least $1/q_{H_2}$ probability since the outputs of H_2 are independently random. Next, $\mathcal{B}$ can correctly guess the time period w^* of the forged aggregate signature with at least $1/q_{H_1}$ probability since he randomly chooses a random w'. Note that the probability $H_2(M_2||w^*) = h'$ is negligible. Therefore, the success probability of $\mathcal{B}$ is at least $1/q_{H_1} q_{H_2} \cdot Adv_{\mathcal{A}}^{SyncAS}$ where $Adv_{\mathcal{A}}^{SyncAS}$ is the success probability of $\mathcal{A}$. This completes our proof. □

4.4 Discussions

Efficiency. The public key of our SyncAS scheme consists of just one group element since our SyncAS scheme is derived from the SeqAS scheme of the previous section, and the synchronized aggregate signature consists of one group element and one integer since anyone can compute A, B using the hash functions. The signing algorithm requires two group hash operations and two exponentiations, and the aggregate verification algorithm requires two group hash operations, three pairing operations, and l exponentiations where l is the number of signers in the aggregate signature. Our SyncAS scheme provides the shortest aggregate signature size compared to the previous SyncAS schemes [19, 1] since the aggregate signature of previous SyncAS schemes consists of two group elements and one integer. Additionally the signing and verification algorithms of our scheme are efficient compared to the previous SyncAS schemes.

Asymmetric Bilinear Groups. To reduce the size of aggregate signatures, we can use asymmetric bilinear groups instead of symmetric bilinear groups. The SyncAS scheme in asymmetric bilinear groups is described as follows: the public parameters is $PP = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}, H_1, H_2)$, the private key and the public key are $SK = x$ and $PK = \hat{X}$ respectively, the individual signature is $\sigma = (C = H_1(0||w)^x H_1(1||w)^{xH_2(M||w)}, w)$, and the aggregate signature is $\sigma_\Sigma = (C = H_1(0||w)^{\sum x_i} H_1(1||w)^{\sum x_i H_2(M_i||w_i)}, w)$. For instance, if we instantiate the asymmetric bilinear groups using the 175-bit MNT curve with embedding degree 6 to guarantee 80-bit security level, the size of aggregate signature is 207 bit since the size of a time period can be 32 bit.

Combined Aggregate Signature. We can construct a combined aggregate signature scheme that supports sequential aggregation and synchronized aggregation at the same time by combining our SeqAS scheme and our SyncAS scheme since the private key and the public key of our two schemes are the same. In the combined aggregate signature scheme, the public parameters is $PP = (p, \mathbb{G}, \mathbb{G}_T, e, g, Y, H_1, H_2)$, the private

key and the public key are $SK = x$ and $PK = X$ respectively. A signer with a private key can generate a sequential aggregate signature or a synchronized aggregate signature.

Public-Key Signature. We can also derive another PKS scheme from our SyncAS scheme. The derived PKS scheme has a restriction such that a signer should use a random value w that was not used before. This derived PKS scheme is the same as the CL* signature scheme that was proposed by Camenisch et al. [15]. The CL* signature scheme supports batch verification that enables a verifier to quickly check the validity of many signatures on different messages and different signers. Compared to the CL signature scheme, the signature of the CL* signature scheme consists of one group element and one integer instead of three group elements, and the signature verification algorithm of the CL* signature scheme just requires two pairing operations instead of five pairing operations.

Removing Random Oracles. If the size of message space is less than a polynomial number, then Camenisch et al. showed that random oracles can be removed in the CL* signature scheme by using the universal one-way hash function of Canetti et al. [17, 15]. We also can use the universal one-way hash function in our SyncAS scheme if the size of message space is less than a polynomial number. However, the SeqAS scheme using the universal one-way hash function of Canetti et al. is inefficient since it requires polynomial number of exponentiations.

5 Conclusion

In this paper, we proposed a new sequential aggregate signature scheme and a new synchronized aggregate signature scheme using a newly devised "public key sharing" technique, and we proved their security under the LRSW assumption. Our two aggregate signature schemes in this paper sufficiently satisfy the efficiency properties of aggregate signatures such that the size of public keys should be short, the size of aggregate signatures should be short, and the aggregate verification should be efficient.

An interesting problem is to prove the security of our schemes under static assumptions instead of the interactive LRSW assumption. Recently, Gerbush et al. proposed a modified CL signature scheme in composite order bilinear groups and proved its security under static assumptions [20]. One may consider to use the modified CL signature scheme of Gerbush et al. for aggregate signature schemes, but it is not easy to apply our techniques to the modified CL signature scheme in composite order bilinear groups.

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] Giuseppe Ateniese, Jan Camenisch, and Breno de Medeiros. Untraceable rfid tags via insubvertible encryption. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 92–101. ACM, 2005.

[3] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. `http://eprint.iacr.org/2005/385`.

[4] 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.

[5] 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.

[6] 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.

[7] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. *J. Cryptology*, 22(1):114–138, 2009.

[8] 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.

[9] 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`.

[10] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. *J. Cryptology*, 25(1):57–115, 2012.

[11] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2004.

[12] 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.

[13] 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.

[14] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations. Cryptology ePrint Archive, Report 2011/222, 2011. `http://eprint.iacr.org/2011/222`.

[15] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.

[16] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[17] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.

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

[19] 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.

[20] Michael Gerbush, Allison B. Lewko, Adam O'Neill, and Brent Waters. Dual form signatures: An approach for proving security from static assumptions. Cryptology ePrint Archive, Report 2012/261, 2012. `http://eprint.iacr.org/2012/261`.

[21] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Sequential aggregate signatures with short public keys: Design, analysis, and implementation studies. Cryptology ePrint Archive, Report 2012/518, 2012. `http://eprint.iacr.org/2012/518`.

[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] 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.

[24] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.

[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] 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.

[27] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[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.