# Aggregating CL-Signatures Revisited: Extended Functionality and Better Efficiency

Kwangsu Lee[1,2⋆], Dong Hoon Lee[1⋆⋆], and Moti Yung[2,3]

[1] CIST, Korea University, Korea
{guspin,donghlee}@korea.ac.kr
[2] Columbia University, USA
{kwangsu,moti}@cs.columbia.edu
[3] Google Inc., USA

**Abstract.** Aggregate signature is public-key signature that allows anyone to aggregate different signatures generated by different signers on different messages into a short (called aggregate) signature. The notion has many applications where compressing the signature space is important: in infrastructure: secure routing protocols, in security: compressed certificate chain signature, in signing incrementally changed data: such as software module authentications, and in transaction systems: like in secure high-scale repositories and logs, typical in financial transactions. In spite of its importance, the state of the art of the primitive is such that it has not been easy to devise a suitable aggregate signature scheme that satisfies the conditions of real applications, with reasonable parameters: short public key size, short aggregate signatures size, and efficient aggregate signing/verification. In this paper, we propose two aggregate signature schemes based on the Camenisch-Lysyanskaya (CL) signature scheme whose security is reduced to that of CL signature (i.e., secure under the LRSW assumption) which substantially improve efficiency conditions for real applications. The first scheme is an "efficient sequential aggregate signature" scheme with the shortest size public key, to date, and very efficient aggregate verification. The second scheme is an "efficient synchronized aggregate signature" scheme with a very short public key size, and with the shortest (to date) size of aggregate signatures among synchronized aggregate signature schemes. Signing and aggregate verification are very efficient. Furthermore, our schemes are compatible: a signer of our aggregate signature schemes can dynamically use two modes of aggregation "sequential" and "synchronized," employing the same private/public key.

**Keywords:** Public-key signature, Aggregate information applications, Aggregate signature, CL signature, Bilinear map.

⋆ Supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the C-ITRC (Convergence Information Technology Research Center) support program (NIPA-2013-H0301-13-3007) supervised by the NIPA (National IT Industry Promotion Agency).

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

## 1 Introduction

Public-key signature (PKS) is a central cryptographic primitive with numerous applications. However, constructing a PKS scheme that is efficient, secure, and flexible enough for a range of possible applications is not easy. Among such schemes, CL signature, proposed by Camenisch and Lysyanskaya [12], is one of the pairing-based signature schemes [8, 10, 12, 24] that satisfies these conditions. It was widely used as a basic component in various cryptosystems such as anonymous credential systems, group signature, RFID encryption, batch verification signature, ring signature [2, 3, 5, 11, 12], as well as in aggregate signature [22].

Pubic-key aggregate signature (PKAS), introduced by Boneh, Gentry, Lynn, and Shacham [9], 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. Boneh et al. proposed the first full aggregate signature scheme in bilinear groups and proved its security in the random oracle model under the CDH assumption. After the introduction of aggregate signatures, various types of aggregate signatures such as sequential aggregate signatures [6, 15, 17–19] and synchronized aggregate signatures [1, 14] were proposed. PKAS has numerous applications. In network and infrastructure: secure routing protocols, public-key infrastructure systems (signing certificate chains), sensor network systems, proxy signatures, as well as in applications: dynamically changing document composition (in particular, secure updating of software modules), secure transaction signing, secure work flow, and secure logs and repositories [1, 6, 7, 9]. In all these applications, compressing the space consumed by signatures is the major advantage. Note that in the area of financial transactions, in particular, logs and repositories are very large due to regulatory requirements to hold records for long time periods. The effect of compressing signatures by aggregation increases with the number of data items; thus it is quite plausible that the financial sector may find variations of aggregate signature most useful.

Though PKAS can reduce the size of signers' signatures by using the aggregation technique, it cannot reduce the size of signers' public keys since the public keys are not aggregated. Thus, the total information the verifier needs to access 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 very important to reduce the size of public keys. 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 a trust structure different from public key infrastructure, namely, the existence of an additional trusted authority, (the current IBAS schemes are in [6, 14, 15] and are all secure in the random oracle model.) To construct a PKAS scheme with short public keys, Schröder proposed a sequential aggregate signature scheme with short public keys based on the CL signature scheme [22]. 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, this work, while nicely pointing at the

CL signature as a source of efficiency for the context of aggregate signatures, still leaves out desired properties to build upon while exploiting the flexibility of the CL signature: can we make the public key shorter? can we require substantially less work in verification? and, can we build other modes of aggregate signatures? While asking such questions, we revisit the subject of aggregate signature based on CL signatures.

### 1.1 Our Contributions

In this paper, we indeed 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 based on that of CL signature (i.e., the LRSW assumption) without random oracles. A sequential aggregate signature assumes that the aggregation mode is done in linear order: signed message after signed message. 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 shortest among all sequential aggregate schemes to date (except IBAS schemes). The aggregate verification algorithm of our scheme is quite efficient since it just requires five pairing operations and $l$ exponentiations (or multi-exponentiations). Therefore our scheme simultaneously 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 based on the CL signature security in the random oracle model (the random oracle can be removed if the number of messages is restricted to be polynomial). Synchronized aggregate signature is a mode where the signers of messages to be aggregated are synchronized, but aggregation can take any order. 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 to date. The aggregate verification algorithm of this scheme is also quite efficient since it just requires three pairing operations and $l$ exponentiations (or multi-exponentiations).

Additionally, our two aggregate signature schemes can be combined to give a new notion of *aggregate "multi-modal" signature scheme*: A scheme which supports, both, sequential aggregation or synchronized aggregation, since the public key and the private key of two schemes are the same. This property can increase the utility and flexibility of the suggested scheme(s).

### 1.2 Related Work

Given the importance of aggregation to saving signature space, much work has been invested in the various notions allowing aggregation.

**Full Aggregation.** The notion of public-key aggregate signature (PKAS) was introduced by Boneh, Gentry, Lynn, and Shacham [9]. 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 [9]. 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 [4].

**Sequential Aggregation.** The concept of sequential aggregate signature was introduced by Lysyanskaya, Micali, Reyzin, and Shacham [19]. 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 [19]. Boldyreva et al. proposed an identity-based sequential aggregate signature (IBSAS) scheme (in the trust model of identity-based schemes with a trusted private keys authority), in bilinear groups and proved its security in the random oracle model under an interactive assumption [6]. Recently, Gerbush et al. showed that a modified IBSAS scheme of Boldyreva et al. in composite order bilinear groups can be secure in the random oracle model under static assumptions [15].

The first sequential PKAS scheme without random oracles was proposed by Lu et al. [18]. They constructed a sequential PKAS scheme based on the PKS scheme of Waters and proved its security without random oracles under the CDH assumption. 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 the CL signature based scheme discussed above [22]. Recently, Lee et al. proposed an efficient sequential PKAS scheme with short public keys and proved its security without random oracles under static assumptions [17].

**Synchronized Aggregation.** The concept of synchronized aggregate signature was introduced by Gentry and Ramzan [14]. In synchronized aggregate signature, all signers have synchronized time information and individual signatures generated by different signers within the same time period can be aggregated into a short aggregate signature. They proposed an identity-based synchronized aggregate signature scheme in bilinear groups and proved its security in the random oracle model under the CDH assumption [14]. Ahn et al. proposed an efficient synchronized PKAS scheme based on the PKS scheme of Hohenberger and Waters and proved its security without random oracles under the CDH assumption [1].

## 2 Preliminaries

In this section, we define bilinear groups, and introduce the LRSW assumption which is associated with the security of the CL signature scheme, which is, then, presented as well.

### 2.1 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.

### 2.2 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. [20] and it is secure under the generic group model defined by Shoup [23] (and adapted to bilinear groups in [12]).

**Assumption 1 (LRSW)** *Let $\mathcal{G}$ be an algorithm that on input the security parameter $1^\lambda$, 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 $O_{X,Y}(\cdot)$.*

### 2.3 The CL Signature Scheme

The CL signature scheme is a public-key signature scheme that was proposed by Camenisch and Lysyanskaya [12] and the security was proven without random oracles under the LRSW assumption. Although the security of the CL signature scheme is based on this interactive assumption, it is flexible and widely used for the constructions of various cryptosystems [5, 11, 12, 20, 22] (this is so, perhaps due to its relatively elegant and simple algebraic structure).

**PKS.KeyGen($1^\lambda$):** 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**$(\sigma, M, PK)$**:** 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 ( [12]).** *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 propose an efficient sequential aggregate signature (SeqAS) scheme based on the CL signature scheme, and prove its security without random oracles.

### 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 [19]. That is, a signer with an index $i$ receives an aggregate signature $\sigma'_\Sigma$ from the signer of an index $i-1$, and he generates a new aggregate signature $\sigma_\Sigma$ 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 SeqAS scheme consists of four PPT algorithms **Setup**, **KeyGen**, **AggSign**, and **AggVerify**, which are defined as follows:

- **Setup**$(1^\lambda)$. The setup algorithm takes as input a security parameter $1^\lambda$ and outputs public parameters $PP$.
- **KeyGen**$(PP)$. The key generation algorithm takes as input the public parameters $PP$, and outputs a public key $PK$ and a private key $SK$.
- **AggSign**$(\sigma'_\Sigma, \mathbf{M}, \mathbf{PK}, M, SK, PP)$. The aggregate signing algorithm takes as input an aggregate-so-far $\sigma'_\Sigma$ 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 $\sigma_\Sigma$.
- **AggVerify**$(\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP)$. The aggregate verification algorithm takes as input an aggregate signature $\sigma_\Sigma$ 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 $\sigma'_\Sigma$ 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. [19]. In this paper, we follow the security model that was proposed by Lu et al. [18]. The security model of Lu et al. is a more restricted model that requires the

adversary to correctly 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.

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 $\sigma'_{\Sigma}$ 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 $\sigma_{\Sigma}$.

**Output**: Finally (after a sequence of the above queries), $\mathcal{A}$ outputs a forged sequential aggregate signature $\sigma^*_{\Sigma}$ 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 exist 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 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 generate 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.

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. [18]. 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 $\{X_i\}$ values of signers instead of the $\{X_i, Y_i\}$ 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 $x_2$ generates 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 SeqAS scheme is described as follows:

**SeqAS.Setup($1^\lambda$):** This 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($PP$):** This 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($\sigma'_\Sigma, \mathbf{M}', \mathbf{PK}', M, SK, PP$):** This 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 $\sigma'_\Sigma$ by calling **AggVerify**$(\sigma'_\Sigma, \mathbf{M}', \mathbf{PK}', PP)$. If $\sigma'_\Sigma$ 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 $\sigma'_\Sigma = (g, Y, 1)$. 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($\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP$):** This 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 $M_i$ is in $\mathbb{Z}_p^*$, any public key does not appear twice in $\mathbf{PK}$, and 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

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

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

### 3.3 Security Analysis

We prove the security of our SeqAS scheme based on the security of the CL signature scheme without random oracles.

**Theorem 3.** *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.*

*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 requests 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 generate 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 $\epsilon$. A simulator $\mathcal{B}$ that forges the CL signature scheme is first 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 by providing an aggregate-so-far $\sigma'_\Sigma$ 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 $\sigma'_\Sigma$ is valid by calling **SeqAS.AggVerify** 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 $\sigma$.

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 $\sigma_\Sigma$ 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. It first checks the validity of $\sigma^*_\Sigma$ 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 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 [22], 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.

## 4 Synchronized Aggregate Signature

In this section, we propose an efficient synchronized aggregate signature (SyncAS) scheme based on the CL signature scheme, and prove its security in the random oracle model.

### 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 [1, 14]. 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 $\sigma_\Sigma$ 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 SyncAS scheme consists of six PPT algorithms **Setup**, **KeyGen**, **Sign**, **Verify**, **Aggregate**, and **AggVerify**, which are defined as follows:

- **Setup**$(1^\lambda)$. The setup algorithm takes as input a security parameter $1^\lambda$ and outputs public parameters $PP$.
- **KeyGen**$(PP)$. The key generation algorithm takes as input the public parameters $PP$, and outputs a public key $PK$ and a private key $SK$.
- **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 $\sigma$.
- **Verify**$(\sigma, M, PK, PP)$. The verification algorithm takes as input a signature $\sigma$ 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 $\sigma_\Sigma$.
- **AggVerify**$(\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP)$. The aggregate verification algorithm takes as input an aggregate signature $\sigma_\Sigma$ 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**(**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 [14]. 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 correctly 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.

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 $\sigma$.

**Output**: Finally (after a sequence of the above queries), $\mathcal{A}$ outputs a forged synchronized aggregate signature $\sigma^*_\Sigma$ 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 exist 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 $\mathsf{Adv}^{SyncAS}_{\mathcal{A}} = \Pr[\mathcal{C} = 1]$ 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 $\sigma_\Sigma = (A = g^r, B = Y^r, C = A^{\sum x_i} B^{\sum x_i M_i})$ 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 $\sigma = (C = A^x B^{xM}, w)$. We need to hash a message for the proof of security.

Let $\mathcal{W}$ be a set of time periods where $|\mathcal{W}|$ is fixed polynomial in the security parameter[4]. Our SyncAS scheme is described as follows:

**SyncAS.Setup($1^\lambda$):** This 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 $H_2 : \{0,1\}^* \times \mathcal{W} \to \mathbb{Z}_p^*$. It outputs public parameters as $PP = (p, \mathbb{G}, \mathbb{G}_T, e, g, H_1, H_2)$.

**SyncAS.KeyGen($PP$):** This 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($M, w, SK, PP$):** This algorithm takes as input a message $M \in \{0,1\}^*$, 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($\sigma, M, PK, PP$):** This 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($\mathbf{S}, \mathbf{M}, \mathbf{PK}, PP$):** This 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($\sigma_\Sigma, \mathbf{M}, \mathbf{PK}, PP$):** This 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 any public key in $\mathbf{PK}$ has been certified. If these checks fail, then it outputs 0. Next, it sets $A = H_1(0||w), B = H_1(1||w)$, $h_i = H_2(M_i||w)$ for all $1 \leq i \leq l$ and ver-

[4] The set $\mathcal{W}$ does not need to be included in $PP$ since an integer $w$ in the range $[1, T]$ can be used where $T$ is fixed polynomial in the security parameter. In practice, we can set $T = 2^{32}$ if the maximum time period of certificates is 10 years and a signer generates a signature per each second. The previous SyncAS schemes [1,14] support exponential size of time periods while our SyncAS scheme supports polynomial size of time periods.

ifies 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

$$C = H_1(0||w)^{\sum_{i=1}^{l} x_i} H_1(1||w)^{\sum_{i=1}^{l} x_i H_2(M_i||w)}$$

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

### 4.3 Security Analysis

We prove the security of our SyncAS scheme based on the security of the CL signature scheme in the random oracle model.

**Theorem 4.** *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.*

*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 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 on a message $M$ and a time period $w$ such that $w \neq w'$, then he can easily generate the signature by using the programmability of the random oracle model. If the adversary requests a signature for the time period $w = w'$, then he can generate 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 $\epsilon$. A simulator $\mathcal{B}$ that forges the CL signature scheme is first 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 its signing oracle **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 $H_1\text{-}List, H_2\text{-}List$ 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 value for $H_1$ and $H_2$ respectively. If this is a $H_1$ hash query on a bit $b \in \{0,1\}$ 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 $r_{0,i} \in \mathbb{Z}_p$ and sets $H_1(0||w_i) = g^{r_{0,i}}$.
- 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 $r_{1,i} \in \mathbb{Z}_p$ and sets $H_1(1||w_i) = g^{r_{1,i}}$.
- 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 $\mathcal{B}$ keeps the tuple $(b, w_i, r_{b,i}, H_1(b||w_i))$ 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 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 responds $\sigma_{i,j} = (X^{r_{0,i}} X^{r_{1,i} h_{i,j}}, w_j)$ where $r_{0,i}, r_{1,i}$, and $h_{i,j}$ are retrieved from the $H_1$-$List$ and $H_2$-$List$.
- If $w_i = w'$ and $i = k$, then it responds $\sigma_{i,j} = (C', w_j)$.
- 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 $\sigma^*_\Sigma$ 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 forged time period.
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

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

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 queries 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 queries 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 queries at least $q_{H_2}^{-1}$ 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 $|\mathcal{W}|^{-1}$ 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 $|\mathcal{W}|^{-1} \cdot q_{H_2}^{-1} \cdot \mathbf{Adv}_{\mathcal{A}}^{SyncAS}$ where $\mathbf{Adv}_{\mathcal{A}}^{SyncAS}$ 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 [1, 14]

**Combined (Multi-Modal) 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. The security model of the combined aggregate signatures can be defined by combining the security models of SeqAS schemes and

SyncAS schemes. The details of this scheme are given in the full version of this paper [16].

**Removing Random Oracles.** If the number of messages is restricted to be polynomial, then we can use the universal one-way hash function [13, 21]. However, the SeqAS scheme using the universal one-way hash function of Canetti et al. is inefficient since it requires large number of exponentiations.

## 5 Conclusion

In this paper we concentrated on the notion of aggregate signatures which applications are in reducing space of signatures for large repositories (such as in the legal, financial, and infrastructure areas). 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.

## Acknowledgements

We thank the anonymous reviewers of FC 2013 for their valuable comments.

## References

1. Ahn, J.H., Green, M., Hohenberger, S.: Synchronized aggregate signatures: new definitions, constructions and applications. In: ACM Conference on Computer and Communications Security. (2010) 473–484
2. Ateniese, G., Camenisch, J., de Medeiros, B.: Untraceable rfid tags via insubvertible encryption. In Atluri, V., Meadows, C., Juels, A., eds.: ACM Conference on Computer and Communications Security, ACM (2005) 92–101
3. Ateniese, G., Camenisch, J., Hohenberger, S., de Medeiros, B.: Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385 (2005) `http://eprint.iacr.org/2005/385`.
4. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A., eds.: ICALP. Volume 4596 of LNCS, Springer (2007) 411–422
5. Bender, A., Katz, J., Morselli, R.: Ring signatures: Stronger definitions, and constructions without random oracles. J. Cryptology **22**(1) (2009) 114–138
6. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. Cryptology ePrint Archive, Report 2007/438 (2010) `http://eprint.iacr.org/2007/438`.
7. Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. J. Cryptology **25**(1) (2012) 57–115

8. Boneh, D., Boyen, X.: Short signatures without random oracles. In Cachin, C., Camenisch, J., eds.: EUROCRYPT. Volume 3027 of LNCS, Springer (2004) 56–73
9. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In Biham, E., ed.: EUROCRYPT. Volume 2656 of LNCS, Springer (2003) 416–432
10. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In Boyd, C., ed.: ASIACRYPT. Volume 2248 of LNCS, Springer (2001) 514–532
11. Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. In Naor, M., ed.: EUROCRYPT. Volume 4515 of LNCS, Springer (2007) 246–263
12. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In Franklin, M.K., ed.: CRYPTO. Volume 3152 of LNCS, Springer (2004) 56–72
13. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In Biham, E., ed.: EUROCRYPT. Volume 2656 of LNCS, Springer (2003) 255–271
14. Gentry, C., Ramzan, Z.: Identity-based aggregate signatures. In Yung, M., Dodis, Y., Kiayias, A., Malkin, T., eds.: Public Key Cryptography. Volume 3958 of LNCS, Springer (2006) 257–273
15. Gerbush, M., Lewko, A.B., O'Neill, A., Waters, B.: Dual form signatures: An approach for proving security from static assumptions. In Wang, X., Sako, K., eds.: ASIACRYPT. Volume 7658 of LNCS, Springer (2012) 25–42
16. Lee, K., Lee, D.H., Yung, M.: Aggregating cl-signatures revisited: Extended functionality and better efficiency. Cryptology ePrint Archive, Report 2012/562 (2012) `http://eprint.iacr.org/2012/562`.
17. Lee, K., Lee, D.H., Yung, M.: Sequential aggregate signatures with short public keys: Design, analysis, and implementation studies. In Kurosawa, K., Hanaoka, G., eds.: PKC. Volume 7778 of LNCS, Springer (2013) 423–442
18. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In Vaudenay, S., ed.: EUROCRYPT. Volume 4004 of LNCS, Springer (2006) 465–485
19. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In Cachin, C., Camenisch, J., eds.: EUROCRYPT. Volume 3027 of LNCS, Springer (2004) 74–90
20. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In Heys, H.M., Adams, C.M., eds.: Selected Areas in Cryptography. Volume 1758 of LNCS, Springer (1999) 184–199
21. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In Johnson, D.S., ed.: STOC, ACM (1989) 33–43
22. Schröder, D.: How to aggregate the cl signature scheme. In Atluri, V., Díaz, C., eds.: ESORICS. Volume 6879 of LNCS, Springer (2011) 298–314
23. Shoup, V.: Lower bounds for discrete logarithms and related problems. In Fumy, W., ed.: EUROCRYPT. Volume 1233 of LNCS, Springer (1997) 256–266
24. Waters, B.: Efficient identity-based encryption without random oracles. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of LNCS, Springer (2005) 114–127