An Efficient Synchronized Aggregate Signature Scheme From Standard RSA Assumption

Xinshun Guo\(^1\) and Zhiwei Wang\(^2\)

\(^1\)School of Business Information Management, Shanghai University of International Business and Economics, Shanghai 201620, P.R.China
\(^2\)College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, P.R.China
gxs@suibe.edu.cn, zhwwang@njupt.edu.cn

Abstract

An aggregate is a digital signature that supports aggregation where any one given \(n\) signatures on \(n\) distinct messages from \(n\) different users can aggregate all these signatures into a single one. We proposed an efficient synchronized aggregate signature scheme from the standard RSA assumption. Our construction is based on the Hohenburger and Waters RSA signature (Crypto 09). This scheme can be proved existentially unforgeable against adaptive chosen-message attacks without random oracles.

Keywords: aggregate signature, HW signature, RSA assumption, standard model

1. Introduction

The notion of aggregate signature is introduced by Boneh et al., [1], which allows an efficient algorithm to aggregate \(n\) signatures of \(n\) distinct messages from \(n\) different users into one single signature. The result aggregate signature can convince the verifier that \(n\) different users did indeed sign \(n\) distinct messages. This primitive has many applications where storage or bandwidth is restricted, and thus one wants to reduce the total cryptographic overhead.

Boneh et al., presented the first aggregate signature [1], which was based on BLS signature[2] in groups with bilinear maps. Subsequently, Lysyanskaya et al. presented a sequential RSA-based scheme [3] that, while more limited, could be instantiated using more general assumptions. In Eurocrypt 2006, Lu et al., proposed another sequential aggregate signature [4] from Waters signature [5]. A sequential aggregate signature is that signatures can only be aggregated by sequentially passing the aggregate from one signer to the next. However, it is in appropriate for many important applications where signers cannot be conveniently arranged in sequence. Furthermore, the schemes in [1-3] are proved secure in the random oracle model, only the scheme in [4] is proved secure in the standard model.

In order to eliminate the need for signer interaction, Gentry et al. proposed an aggregate signature scheme [6] where no interaction between signers was required, provide that all signers have a global strategy for choosing a value \(\omega\) that is used during signing. Only the signature with the same \(\omega\) can be aggregated. This kind of aggregate signature is called synchronized aggregate signature, and Gentry et al., scheme is under CDH assumption in the random oracle model. In CCS 2010, Ahn et al. proposed an novel synchronized aggregate signature [11] based on the stateful signature of Hohenburger and Waters (Eurocrypt 2009) [7]. They define \(\omega\) as a time period. Although their scheme can be proved secure under
the CDH assumption in the standard model, the computational overhead is heavy. To verify an aggregate of \( N \) signatures, requires \( k + 3 \) pairings, (where \( k \) is a security parameter,) plus \( N + 2 \) exponentiations. The computational cost of pairing is usually considered very heavy.

We present a synchronized aggregate signature under Hohenburger and Waters’s RSA signature [8] (Crypto 09). Our scheme is provably secure based on the standard RSA assumption in the standard model. Compared with Ahn et al., scheme [11], our scheme is efficient and without pairing.

In Section 2, we review the standard RSA assumption and define some notations. In Section 3, we review the secure definition of synchronized signature. Section 4 presents the synchronized signature scheme under the standard RSA assumption. Section 5 presents the security analysis to our scheme. Section 6 gives the conclusion.

2. Preliminary

**RSA Assumption.** Let \( \kappa \) be the security parameter. Let integer \( n \) be the product of two \( \kappa \) -bits, distinct odd primes \( p, q \). Let \( e \) be a randomly chosen positive integer less than and relatively prime to \( \phi(n) = (p - 1)(q - 1) \). Given \((n, e)\) and randomly chosen \( y \in \mathbb{Z}_n^* \), we say that the \((\tau, e)\)-RSA assumption holds if for all \( \tau \)-time adversary \( \mathcal{A} \)

\[
\Pr[(x) \leftarrow A(n, y, e), x \in \mathbb{Z}_n^*, x^e = y \text{ mod } n] \leq \varepsilon
\]

where the probability is over the random choices of \( y, n, e \), and the random coins of \( \mathcal{A} \).

**Notation.** For \( B \in \{0,1\}^{l_\kappa} \), let \( B = b_1 b_2 \cdots b_{l_\kappa} \) be the binary presentation of \( B \) with \( b_i \in \{0,1\} \) for \( i \in [1,\cdots, l_\kappa] \). We denote by \( B^{(i)} = 0^{l_\kappa-i} b_1 \cdots b_i \in \{0,1\}^{l_\kappa} \) the prefix of \( B \) that consists of \( l_\kappa - i \) zeros and the first \( i \) bits of \( B \). We use \( QR_n \) to denote the set of quadratic residues modulo \( n \in \mathbb{N} \), i.e., \( QR_n = \{ x \mid \exists y \in \mathbb{Z}_n^*: y^2 = x \text{ mod } n \} \). We use \( \lambda \in \mathbb{N} \) to indicate the security parameter while \( 1^\lambda \) describes the string that consist of \( \lambda \) ones. Let \( l = \ell(\lambda) \) and \( l_\kappa = \ell_\kappa(\lambda) \) be polynomials with \( l_\kappa \leq l \). We define a prime map function called \textit{nearprime}: \( \{0,1\}^{l_\kappa} \to P_{l_\kappa} \), \( P_{l_\kappa} \) is a set of primes belongs to \( \{1,\cdots, 2^{l_\kappa}\} \). The function \textit{nearprime} maps to the smallest prime that is equal or greater than the input value.

**Nearprime Function.** We will construct a \textit{nearprime} function: \( r : \{0,1\}^{l_\kappa} \to P_{l_\kappa} \), that is the same prime mapping function used by [9]. Firstly, we choose a random key \( K \) for a key space \( K \) for the pseudo-random function (PRF) \( F : \{0,1\}^k \to \{0,1\}^{l_\kappa} \), and a random value \( s \in_k \{0,1\}^{l_\kappa} \). Secondly, we establish a function as \( H_k(X) = F_k(X) \oplus s \). Then the final definition of \( r \) is \( r(X) = \text{nearprime}(H_k(X)) \).

For convenience, we additionally define a new function \( E : \{0,1\}^{l_\kappa} \to \mathbb{N} \) as \( E(B) = \prod_{i=1}^{l_\kappa} r(B^{(i)}) \).
3. Secure Definition of Synchronized Aggregate Signature

In an aggregate signature, Suppose that there are \( N \) signers, each chooses a public-private key pair \( (pk_i, sk_i) \) in the same system parameters. Signer \( u \) signs a message \( M \), to obtain a signature \( \sigma_i \). Then there is a public aggregation algorithm that takes as input all of individual signatures \( \sigma_1, \cdots, \sigma_N \) and outputs a short compressed signature \( \sigma_{agg} \). Anyone can aggregate the signatures. Moreover, the aggregation can be performed incrementally. There is also an aggregate verification algorithm that takes as input \( pk_1, \cdots, pk_N, M_1, \cdots, M_N \) and \( \sigma_{agg} \), and decides whether the aggregate signature is valid. This is also true for synchronized aggregate signature, except that all signers have a synchronized clock and two restrictions, which can be described as follows:

1. A signer can sign at most one signature in each time period (one value of clock denotes a time period).
2. Only the signatures with the same clock value can be aggregated.

Then, we give the formal definition of synchronized aggregate signature as Gentry et al.[6] and Ahn et al.[11].

Definition 1. (Synchronized Aggregate Signatures) A synchronized aggregate signature scheme is consisted with six algorithms (\( \text{ParaGen, Setup, Sign, Verify, Aggregate, AggVerify} \)).

\( \text{ParaGen}(1^\lambda) \): This algorithm generates the public parameters \( \text{par} \).

\( \text{Setup}(1^\lambda, \text{par}) \): This algorithm takes in \( 1^\lambda \) and \( \text{par} \), outputs a keypair \( (pk, sk) \).

\( \text{Sign}(sk, M, t) \): This algorithm takes input as a secret key \( sk \), a message \( M \), and the current clock value \( t \), and produces a signature \( \sigma \).

\( \text{Verify}(pk, M, \sigma) \): This algorithm takes in a public key \( pk \), a message \( M \), and a corresponding signature \( \sigma \), and verifies whether \( \sigma \) is a valid signature. If valid, it outputs 1, otherwise 0.

\( \text{Aggregate}( (pk_1, M_1, \sigma_1), \cdots, (pk_N, M_N, \sigma_N) ) \): On input a sequence of public keys \( (pk_1, \cdots, pk_N) \), messages \( (M_1, \cdots, M_N) \), and corresponding signatures \( (\sigma_1, \cdots, \sigma_N) \), this algorithm outputs an aggregate signature \( \sigma_{agg} \).

\( \text{AggVerify}( (pk_1, \cdots, pk_N), (M_1, \cdots, M_N), \sigma_{agg} ) \): On input a sequence of public keys \( (pk_1, \cdots, pk_N) \), messages \( (M_1, \cdots, M_N) \), and corresponding aggregate signature \( \sigma_{agg} \), this algorithm outputs 1 if \( \sigma_{agg} \) is a valid aggregate signature and 0 otherwise.

Next, we recall Ahn et al.’s secure definition of aggregate signature[11].

Exist Unforgeability According to the standard definition of security of Goldwasser, Micali and Rivest [10], a signature scheme \( S = ( \text{ParaGen, Setup, Sign, Verify} ) \) is existentially unforgeable under an adaptive chosen messages attack if no forger that has \( q \) times access to a signing oracle \( O(sk..) \) can produce a new valid signature on a new
message. The security definition of aggregate signature is an extension to the normal digital signature, which can be described as a game between a challenger $C$ and an adversary $A$.

**Setup:** The challenger $C$ runs $\text{ParaGen}(1^\lambda)$ algorithm to obtain the public parameters $\text{param}$, then it runs the $\text{Setup}(1^\lambda)$ algorithm for $N$ times to obtain $N$ distinct key pairs $(pk_1, sk_1), \ldots, (pk_N, sk_N)$. Finally, $C$ sends $(pk_1, (pk_2, sk_2), \ldots, (pk_N, sk_N))$ to $A$.

**Queries:** The adversary $A$ can request a signature on a message of its adaptively choice under $sk_i$ for each time period $1$ to $q$, provided that at most one query is made per time period. The challenger responses $\text{Sign}(sk_1, M_i, t_i)$ to a query for message $M_i$ at time period $t_i$.

**Forgery:** Finally, the adversary $A$ outputs $((p_k_1, \ldots, p_k_N), (M_1', \ldots, M_N'), \sigma^*)$ and wins the game if

1. $A$ did not make a signature query on $M_i'$;
2. $\text{AggVerify}((p_k_1, \ldots, p_k_N), (M_1', \ldots, M_N'), \sigma^*) = 1$.

We define $AggAdv_A$ to be the probability that the adversary $A$ wins the above game, taken over the coin tosses made by $A$ and the challenger $C$.

**Definition 2. (Exist Unforgeability)** An adversary $A (\tau, q, n, e)$ - breaks an $N$ users aggregate signature scheme if $A$ runs at most in time $\tau$, $A$ makes at most $q$ signature queries and $AggAdv_A$ is at least $\epsilon$. If no forger can $(\tau, q, n, e)$ - break this scheme, we call this aggregate signature scheme is a $(\tau, q, n, e)$ - existentially unforgeable under an adaptive chosen message attack.

### 4. Proposed Synchronized Signature Scheme

#### 4.1. Our Construction

We now define our new synchronized signature scheme. Our construction is based on the Hohenburger and Waters RSA signature[8]. The drapdoor of our scheme is factorization of integer, so $N$ users in our scheme should have the same modulo $n$ and $p, q$. Thus, $N$ users have the same secret key $sk = (p, q)$, however, their public keys are partially distinct.

**ParaGen($1^\lambda$).** Let $l = l(\lambda)$, $l_m = l_m(\lambda)$, and $l_p = l_p(\lambda)$ be polynomials and $l_s = 2l_p$, $l_m \leq l \leq l_p$. Messages can be interpreted as values in $\{0,1\}^l$.

**Setup($1^\lambda$, param).** The setup algorithm chooses a RSA modulus $n = pq$. The length of $n$ is $l_n$, and the length of $p, q$ is $l_p$. Next, the setup algorithm chooses a generator $g \in QR_n$. Finally, it draws $K \in K$ and $s \in_s \{0,1\}^l$ and publishes $pk = (n, K, s, g)$ and keeps the secret key $sk = (p, q)$.
(Remark: $N$ different users have the corresponding secret/public key pairs as $(sk_1 = (p, q), pk_1 = (n, K, s, g))$, $\ldots$, $(sk_N = (p, q), pk_N = (n, K, s, g))$.)

Sign($sk, M, t$): We assume the signer is given the clock value $t = \text{clock}(\cdot) \in \{0,1\}^k$, as input to the algorithm. It keeps an internal state $t_{\text{prev}}$ that denotes the last time period on which it issued a signature. If $t_{\text{prev}} = t$ or $t \geq 2^k$, then it aborts. Otherwise, it record the current time period as $t_{\text{prev}} = t$. The sign algorithm computes a signature on $M$ under secret key $sk$ and time period $t$ as

$$\sigma = (\sigma' = (g^M)^{1/E(t)} \mod n, t).$$

Verify($pk, M, \sigma$): The verify algorithm firstly parses $\sigma$ as $(\sigma', t)$, and checks whether $0 < t < 2^k$. If this is false, it rejects. Then, it verifies the signature by checking that

$$\sigma'^{E(t)} = g^M \mod n.$$

Aggregate(($pk_1, M_1, \sigma_1), \ldots, (pk_N, M_N, \sigma_N)$): This algorithm first parses $\sigma_i$ as $(\sigma'_i, t)$, and checks the second element of $\sigma_i$ (for $i = 1, \ldots, N$). If they are distinct, it outputs “failure”. Otherwise, if they are the same, the algorithm computes the aggregate signature as

$$\sigma_{\text{agg}} = (\gamma = \prod_{i=1}^N \sigma'_i, t).$$

AggVerify(($pk_1, \ldots, pk_N), (M_1, \ldots, M_N), \sigma_{\text{agg}}$): This algorithm parses $\sigma_{\text{agg}}$ as $(\gamma, t)$, and checks that $0 < t < 2^k$, if this false, it rejects. This algorithm verifies the signature by checking that

$$\gamma^{E(t)} = \prod_{i=1}^N g_i^M \mod n.$$

4.2. Efficiency Analysis

Our scheme is practical, since an aggregate signature only requires one element in group of $QR_n$ and a small integer. To verify an aggregate of $N$ signatures, requires only one modular exponentiation and one modular $N$ multi-exponentiations. In general, if $N$ is relatively small, the computational cost of one modular $N$ multi-exponentiations and one modular exponentiation are almost equal. In the following table, we compare our scheme with other aggregate signature schemes in type, assumption, model, aggregate signature size and the computational cost of aggregate verifying.
### Table 1. Summary of Full, Sequential and Synchronized Aggregate Signatures.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Type</th>
<th>Assumption</th>
<th>Model</th>
<th>Agg Size</th>
<th>Agg Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boneh scheme[1]</td>
<td>full</td>
<td>CDH</td>
<td>ROM</td>
<td>1</td>
<td>$N + 1$</td>
</tr>
<tr>
<td>Lysyanskaya ya scheme[3]</td>
<td>seq</td>
<td>Cert TDP</td>
<td>ROM</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Lu scheme[4]</td>
<td>seq</td>
<td>CDH</td>
<td>Standard</td>
<td>2</td>
<td>2 pairings</td>
</tr>
<tr>
<td>Gentry scheme[6]</td>
<td>sync</td>
<td>CDH</td>
<td>ROM</td>
<td>3</td>
<td>3 pairings</td>
</tr>
<tr>
<td>Our scheme</td>
<td>sync</td>
<td>RSA</td>
<td>Standard</td>
<td>2</td>
<td>About 2 modular exponentiations</td>
</tr>
</tbody>
</table>

From the above table, we can see that prior aggregate signatures, Lysyanskaya et al.’s scheme and Lu et al.’s scheme are sequential aggregate signatures. Prior synchronized or full schemes, due to Boneh et al.’s scheme, Gentry et al.’s scheme are only offered heuristic security arguments in the random oracle model. Only one synchronized aggregate signature scheme with a standard security proof exists, due to Ahn et al.’s scheme, and its security is based the CDH assumption. Furthermore, it require $k + 3$ pairings in the verification of $N$ aggregated signatures.

Our scheme can be proved secure under the standard RSA assumption in the standard model, and only requires About 2 modular exponentiations in the aggregate verification (If $N$ is relatively small, the cost of one modular $N$ multi-exponentiation is almost equal to one modular exponentiation). Thus, our scheme is more practical than theirs.

### 5. Security Analysis

In this section, we will prove our scheme secure under the standard RSA assumption in the standard model.

**Theorem 1.** Assume the $(\tau_w, \tau_k)$ -RSA assumption holds, then our synchronized aggregate signature scheme is $(q, \tau_w, \varepsilon_w, N)$ -secure against adaptive chosen message attacks provide that

$$
\varepsilon_w \leq \frac{3q2^{l_x}(2^{l_x} - q + l_y)}{2^{l_x} - q} \varepsilon_k + N \cdot 2^{-(l_x - 2)} , \text{ and } \tau_w \approx \tau_k .
$$

($l_y$ is the length of clock value $t$, and it is relatively small.)
Proof. Assume that \( A \) is forger that \( (q, \tau, \varepsilon, N) \)-breaks the exist unforgeability of our scheme. Then we can construct a challenger \( C \) that breaks the RSA assumption in time \( \tau \) with probability \( \varepsilon \). Given the RSA challenge \( (n, y, e) \), \( C \)'s goal is to compute \( x \) such that \( x^e = y \mod n \).

\( C \) denotes \( 1, \cdots, q \) in the binary presentation by \( t_i, \cdots, t_q \) (clock values), and all the length of \( t_i (i = 1, \cdots, q) \) is \( \ell \). We assume that the adversary makes \( q \) queries for each time period \( 1 \) to \( q \) and receives \( q \) responses to a query for message \( M_i \) at time period \( t_i \in \{ t_1, \cdots, t_q \} \).

Finally, the adversary returns a forgery of \( ((p_k_1, \cdots, p_k_q), (M'_1, \cdots, M'_N), \sigma'_a = (\gamma', t')) \).

We should differentiate two types of forgeries.

- **Type I Forgery**, there exists non-empty set \( J \subseteq \{1, \cdots, q\} \) such that all \( j \in J \) it holds that \( E(t') = E(t_j) \). We note that if \( r() \) is a collision free function for all \( t_i (i \in \{1, \cdots, q\} \) and their prefixes, then \( |J| = 1 \) and \( t' = t_j \).

- **Type II Forgery**, there exists no \( j \in \{1, \cdots, q\} \), such that \( E(t') = E(t_j) \).

**Type I Forgery.**

**Setup:** \( C \) guesses \( 1 \leq j \leq q \) (such that \( E(t') = E(t_j) \) ). Next, \( C \) draws \( K \in K \) and \( s \in \{0, 1\}^s \) uniformly at random from the set of values with \( r(t'_j) = \text{nearprime}(F_k(t'_j) \oplus s) = e \). Now, \( C \) chooses \( N \) random numbers \( \omega_1, \cdots, \omega_N \) from \( Z_{(n-1)/4} \), and computes

\[
R_i = y^{2^j \prod_{t \in \{1, \cdots, t_j\}} E(t_i)} \mod n
\]

\[
R_j = y^{2^j \prod_{t \in \{1, \cdots, t_j\}} E(t_i)} \mod n
\]

\[
R_N = y^{2^N \prod_{t \in \{1, \cdots, t_j\}} E(t_i)} \mod n
\]

Obviously, \( g_1, \cdots, g_N \in QR_n \). Finally, \( C \) sends the \( N \) public keys \( p_k_1 = (n, K, s_i, g_1), \cdots, p_k_N = (n, K, s_i, g_N) \) to \( A \). Since \( s_k = s_k \cdots = s_k \) \( C \) can not send \( s_k, \cdots, s_k \) to the adversary \( A \).

**Queries:** The adversary \( A \) requests a signature on a message of its adaptively choice under \( (sk_1, p_k_1) \) for each time period \( 1 \) to \( q \), provide that at most one query is made per time period. On receiving the signature query on \( M_j, j \in \{1, \cdots, q\} \), \( C \) computes

\[
\sigma'_j = y^{2^j \prod_{t \in \{1, \cdots, t_j\}} E(t_j)} \mod n,
\]
and \((\sigma'_j, t_j)\) is a correct signature on message \(M_j\) in time period \(j\). \(C\) sends \((\sigma'_j, t_j)\) as a response to \(A\).

**Forgery:** Finally, the adversary \(A\) outputs \(((pk_1, \cdots, pk_N), (M'_1, \cdots, M'_N), \sigma'_{\text{agg}} = (\gamma^*, t^*)\). If \(t^* \in \{t_1, \cdots, t_q\}\) and \(t^* = t_j\) \((j \in \{1, \cdots, q\}\), then \(C\) guesses correctly. From the aggregate verification equation, we have that

\[
\gamma^* = \prod_{i=1}^{N} S_i \mod n
\]

\[
= y^{E(t_i, \sum_{i=1, i \neq j}^{N} \omega_i M'_i) + 2 \omega_j M'_j \prod_{i=1, i \neq j}^{N} E(t_i)} \mod n
\]

If \(GCD(e, 2 \prod_{i=1}^{q} E(t_i, \sum_{i=1, i \neq j}^{N} \omega_i M'_i) + 2 \omega_j M'_j \prod_{i=1, i \neq j}^{N} E(t_i)) = 1\), we can compute two values \(\alpha_1, \alpha_2 \in Z\) such that

\[
\alpha_1 e + 2 \alpha_2 \left( \prod_{i=1}^{q} E(t_i, \sum_{i=1, i \neq j}^{N} \omega_i M'_i) + \omega_j M'_j \prod_{i=1, i \neq j}^{N} E(t_i) \right) = 1,
\]

and find a response to the RSA challenge as

\[
y^{\alpha_1 / e} = y^{\alpha_1 \prod_{i=1}^{q} \omega_i^{-1} r_i(\gamma^*)} \mod n
\]

We have completed the simulation of the scheme. In the following step, we will analyze the bounded probability of \(C\).

1) **In the Setup step**, if \(C\) guesses correctly, the probability is \(1 / q\).

2) **In the Setup step**, \(C\) chooses \(\omega_1, \cdots, \omega_N\) from \(Z_{(n-1)/4}\). It holds that \((n-1)/4 = p'q' + (p' + q')/2 > p'q'\), (assume \(q' > p'\)) which implies that

\[
Pr[X \in Z_{(n-1)/4}, X \notin Z_{p'q'}] < \frac{2}{p' + q'} < \frac{1}{2^{-(j,-1)}}.
\]

3) **In the Forgery step**, if \(C\) does not abort, which means that \(t^* \in \{t_1, \cdots, t_q\}\) and \(t^* = t_j\) \((j \in \{1, \cdots, q\}\), the probability is \(\frac{1}{2^j}\).

4) **In the Forgery step**, since \(\omega_1, \cdots, \omega_N\) are hidden from the adversary’s view, the probability of \(GCD(e, 2 \prod_{i=1}^{q} E(t_i, \sum_{i=1, i \neq j}^{N} \omega_i M'_i) + 2 \omega_j M'_j \prod_{i=1, i \neq j}^{N} E(t_i)) = 1\) is at least \(2 / 3\).

Putting the above together, we get that

\[
e_w \leq \frac{3q 2^{2j}}{2} e_w + N \cdot 2^{-(j,-1)}, \text{ and } \tau_w \approx \tau_R.
\]
Type II Forgery.

Setup: $C$ guesses $1 \leq i^* \leq q$ (a random number with the longest prefix in common with the random number $t^*$ in the forgery signature) and $1 \leq j^* \leq l^*_b$ (the length of the longest common prefix plus one). Let the guessed prefix be $\bar{t}$ and Next, $C$ draws $K \in \mathbb{K}$ and $s \in \{0,1\}^b$ uniformly at random from the set of values with $r(t^*) = \text{nearprime}(F_x(t^*) \oplus s) = e$. Now, $C$ chooses $N$ random numbers $\omega_1, \cdots, \omega_N$ from $Z_{(n-1)/4}$, and computes

$$g_1 = y^{2\omega_1 \prod_{i=1}^{t^*} E(t_i)} \mod n$$
$$\vdots$$
$$g_N = y^{2\omega_N \prod_{i=1}^{t^*} E(t_i)} \mod n$$

Obviously, $g_1, \cdots, g_N \in QR_n$. Finally, $C$ sends the $N$ public keys $pk_1 = (n, K, s, g_1), \cdots, pk_N = (n, K, s, g_N)$ to $A$. Since $sk_1 = sk_2 = \cdots = sk_N$, $C$ can not send $sk_2, \cdots, sk_N$ to the adversary $A$.

Queries: The adversary $A$ requests a signature on a message of its adaptively choice under $sk_1$ for each time period $1$ to $q$, provide that at most one query is made per time period. On receiving the signature query on $M_j$, $j \in \{1, \cdots, q\}$, $C$ computes

$$\sigma'_j = y^{2\omega_1 M_j \prod_{i=1}^{\alpha_j^*} E(t_i)} \mod n,$$

and $(\sigma'_j, j)$ is a correct signature on message $M_j$ in time period $j$. $C$ sends $(\sigma'_j, j)$ as a response to $A$.

Forgery: Finally, the adversary $A$ outputs $((pk_1, \cdots, pk_N), (M'_1, \cdots, M'_N), \sigma^*_x = (\gamma^*, t^*))$. If $t^* \notin \{t_1^*, \cdots, t_q^*\}$, then $C$ aborts. Otherwise, if $\bar{t} = t^{*(j^*)}$, $C$ guesses correctly. From the aggregate verification equation, we have that

$$\gamma^{*E(t^*)} = \prod_{i=1}^{N} g_{t^*_i} \mod n$$
$$= y^{2\prod_{i=1}^{t^{*(j^*)}} E(t_i)} \mod n$$

If $GCD(e, 2 \prod_{i=1}^{q} E(t_i)(\sum_{i=1}^{N} \omega_i M'_i)) = 1$, we can compute two values $\alpha_1, \alpha_2 \in \mathbb{Z}$ such that

$$\alpha_1 e + 2\alpha_2 \prod_{i=1}^{q} E(t_i)(\sum_{i=1}^{N} \omega_i M'_i) = 1,$$

and find a response to the RSA challenge as
\[ y^{1/e} = y^{x_1 \gamma^{s_{n,1}} \prod_{j=2}^{s_{n,2}} (e^{n,j})} \mod n \]

We have completed the simulation of the scheme. In the following step, we will analyze the bounded probability of C.

1) In the Setup step, if C guesses correctly, the probability is \( \frac{1}{q^l_b} \).

2) In the Setup step, C chooses \( \omega_1, \ldots, \omega_N \) from \( \mathbb{Z}_{(n-1)/4} \). It holds that \( (n-1)/4 = p'q' + (p'+q')/2 > p'q' \), (assume \( q' > p' \)) which implies that

\[
\Pr [ \chi \in \mathbb{Z}_{(n-1)/4}, \chi \notin \mathbb{Z}_{p'q'} ] < \frac{2}{p'q'} < \frac{1}{p'} < 2^{-l_b - 1}.
\]

3) In the Forgery step, if C does not abort, which means that \( t \notin \{t_1, \ldots, t_q \} \), the probability is \( \frac{2^{l_b} - q}{2^{l_b}} \).

4) In the Forgery step, since \( \omega_1, \ldots, \omega_N \) are hidden from the adversary’s view, the probability of \( \text{GCD} (e, 2 \prod_{i=1}^{q} E(t_i) (\sum_{j=1}^{N} \omega_j M_j)) = 1 \) is at least \( 2/3 \).

Putting the above together, we get that

\[
\varepsilon_u \leq \frac{3q^l_b 2^{l_b}}{2(2^{l_b} - q)} \varepsilon_x + N \cdot 2^{-l_b - 1}, \text{ and } \tau_u \approx \tau_R.
\]

6. Conclusion

An aggregate signature scheme is a digital signature scheme where anyone given \( N \) signatures on \( n \) messages from \( N \) users can aggregate all these signatures into a single short signature. Thus, it is a good candidate for a variety of communication applications where routing flexibility, speed and low bandwidth are needed. For example, in the wireless sensor network (WSN), sensors often run on battery power and must minimize radio communications. Digital signature offer better security properties, but can add significant bandwidth overhead. Aggregate signature can greatly reduce the bandwidth requirements the message signing imposes on WSN. Rather than carry all signature data, intermediate routing nodes can perform signature aggregation at any point where multiple signatures must be routed towards the collector.

In this paper, we propose the first synchronized aggregate signature from the standard RSA assumption without random oracles. It requires that signers have access to a synchronized clock and only signatures from the same period can be aggregated. Compared with Ahn et al.’s construction, our scheme is efficient, and needs not compute pairings in the aggregate verification.
Acknowledgements

This work was supported by Shanghai 085 Project for Municipal Universities, the Innovation Program of Shanghai Municipal Education Commission under grant No. 14YZ134 and the NSF of China under grants No. 61373006.

References
