Analysis and Improvements of Two Identity-Based Perfect Concurrent Signature Schemes

Zhenjie Huang, Kefei Chen, Yumin Wang

Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai 200240, China
State Key Lab of Integrated Service Networks,
Xidian University, Xi'an, Shaanxi 710071, China

Abstract. The notion of concurrent signatures was introduced by Chen, Kudla and Paterson in their seminal paper in Eurocrypt 2004. In concurrent signature schemes, two entities can produce two signatures that are not binding, until an extra piece of information (namely the keystone) is released by one of the parties. Upon release of the keystone, both signatures become binding to their true signers concurrently. In ICICS 2005, two identity-based perfect concurrent signature schemes were proposed by Chow and Susilo. In this paper, we show that these two schemes are unfair, in which the initial signer can cheat the matching signer. We present a formal definition of ID-based concurrent signatures which redress the flaw of Chow et al.'s definition and then propose two simple but significant improvements to fix our attacks.

Keywords: Concurrent signature, Identity-Based, Bilinear pairings, Cryptoanalysis, Fair exchange.

1 Introduction

Background. The problem of fair exchange of signatures is a fundamental and well-studied problem in cryptography, with potential application in a wide range of scenarios in which the parties involved are mutually distrustful. Early work on solving the problem of fair exchange of signatures was based on the idea of timed release or timed fair exchange of signatures [2, 7, 8]. Such protocols are highly interactive with many message flows and may be too interactive for many applications. Another approach to solving this problem involves the use of a trusted (or semi-trusted) third party or arbitrator who can be called upon to handle disputes between signers [1, 3, 4, 9, 10]. The main problem with such an approach is the requirement for a dispute-resolving third party with functions beyond those required of a normal Certification Authority. In general, appropriate third parties may not be available.

Concurrent signatures were introduced as an alternative approach to solving the problem of fair exchange of signatures by Chen, Kudla and Paterson [5] in their seminal paper in Eurocrypt 2004. In concurrent signature schemes, two entities can produce two signatures that are not binding, until an extra piece of information (namely the *keystone*) is released by one of the parties. Upon release of the keystone, both signatures become binding to their true signers concurrently. Concurrent signatures have a benefit that they have none of the disadvantages of previous fair exchange protocols: they require neither special trusted third party nor highly interactive.

Previous Works. In [5], Chen et al. proposed a concrete concurrent signature scheme based on a variant of Schnorr based ring signature scheme. In their scheme, before the keystone is released, any third party cannot be convinced that a signature has indeed been signed by one particular signer, since any signer can always generate this signature by himself/herself. Later, Susilo et al. [13] pointed out that in a situation where the initial signer and the matching signer are known to be honest players, in Chen et al.'s scheme, any third party can be sure that both signers have signed the messages even before the keystone is released. Then, they extended the notion of concurrent signatures to a stronger notion of *perfect concurrent signatures*, which will allow full ambiguity of the concurrent signatures, even both signers are known to be trustworthy. They proposed two concrete schemes to satisfy this model. The first scheme is based on a variant of Schnorr ring signature scheme, and the second scheme is based on bilinear pairing. In 2005, there are four concurrent signature schemes have been proposed. In [12], Susilo and Mu proposed a tripartite concurrent signature scheme from bilinear pairings. In tripartite concurrent signatures, three parties can exchange their signatures in such a way that their signatures will be binding concurrently. Chow and Susilo [6] proposed two identity-based (simply ID-based) perfect concurrent signature schemes based on two major paradigms of ID-based ring signature schemes. Previous concurrent signature schemes use the concept of ring signatures in their construction. Nguyen [11] proposed a new concurrent signature (namely *asymmetric concurrent signatures*) scheme which is independent of the ring signature concept. This scheme based on Schnorr signature scheme and Schnorr-like signature scheme. Recently, Tonien et al. [14] proposed a multi-party concurrent signature scheme using techniques of ring signatures and bilinear pairings.

Contribution. In this paper, we show that Chow et al.'s two ID-based perfect concurrent signature schemes [6] are unfair, in their schemes the initial signer Alice can cheat the matching signer Bob. We will show that, in their schemes, by carefully choosing some communication values, Alice can perform the signature protocol with Bob on messages m_A and m_B, but outputs a valid signature pair $(\widetilde{\sigma}, \sigma_B)$ on $(\widetilde{m}, m_B)$ with $\widetilde{m} \neq m_A$. We give two attacks for each Chow et al.'s schemes, respectively.

For the case of one keystone only, Chen et al. presented a formal definition of fairness in [5], but it is no longer appropriate in the case of two or more keystones. Chen et al. [6] did not present any new definition of fairness. As a result, their schemes are unfair. Furthermore, the definition of ID-based concurrent signatures given by [6] implies that two keystones k_I and k_M are chose by the initial signer. As mentioned above, it may cause unfair. In this paper, we present a formal definition of ID-based concurrent signatures which redress the flaw of Chow et al.'s definition. We then propose two simple but significant improvements to fix our attacks.

Organization. The rest of this paper is organized as follows. In the next section, we review the notions of Bilinear Pairings and Chow et al.'s ID-based perfect concurrent signature schemes. Attacks on the fairness of Chow et al.'s schemes are given in section 3. We give a modified definition of perfect concurrent signatures in section 4. In section 5, we propose two improved identity-based perfect concurrent signature schemes with proofs of their securities. Section 6 concludes this paper.

2 Review of Chow et al.'s Schemes

2.1 Bilinear Pairings and Complexity Assumption

Let $\mathbb{G}_1$ be a cyclic additive group generated by P with order prime q and $\mathbb{G}_2$ be a cyclic multiplicative group with the same order q. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ with the following properties:

Bilinear: For all $P, P_1, P_2, Q, Q_1, Q_2 \in \mathbb{G}_1$,

$$\hat{e}(P_1 + P_2, Q) = \hat{e}(P_1, Q)\hat{e}(P_2, Q),$$
$$\hat{e}(P, Q_1 + Q_2) = \hat{e}(P, Q_1)\hat{e}(P, Q_2).$$

Non-degenerate: There exists $P, Q \in \mathbb{G}_1$ such that $\hat{e}(P, Q) \neq 1$;

Computable: There is an efficient algorithm to compute $\hat{e}(P, Q)$ for all $P, Q \in \mathbb{G}_1$.

Modified Weil pairing and Tate pairings are examples of bilinear maps.

Computational Co-Diffie-Hellman (Co-CDH) Problem. Given a randomly chosen (P_1, P_2, aP_1, bP_2), where $P_1, P_2 \in \mathbb{G}_1, a, b \in \mathbb{Z}_q^*$, and a, b are unknown, compute $abP_2 \in \mathbb{G}_2$.

Co-CDH Assumption. For every probabilistic polynomial-time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ to solve Co-CDH-Problem is negligible.

2.2 Chow et al.'s Scheme 1

Concurrent Signature Algorithms

- SETUP: Choose $(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, P)$ as Section 2.1. The *Private Key Generator* (PKG) selects a random number $s \in \mathbb{Z}_q^*$ and sets $P_{pub} = sP$. It selects three cryptographic hash functions $H_0 : \{0,1\}^* \to \mathbb{G}_1$ and $H_1 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \to \mathbb{G}_1$. It publishes system parameters *params* $= \{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, P, P_{pub}, H_0, H_1, H_2\}$, and keeps s as the *master private key.* The algorithm also sets $\mathcal{M} = \mathcal{K}_I = \mathcal{K}_M = \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{K}' = \mathbb{G}_1$.
- EXTRACT: The EXTRACT algorithm is defined as follows.
  - A user $\mathcal{U}_i$ submits his or her identity ID_i to the PKG.

  - The PKG generates $\mathcal{U}_i$'s private key as $S_{ID_i} = sQ_{ID_i}$, where $Q_{ID_i} = H_0(ID_i)$.
- FIX-INITIAL-KEYSTONE: Assuming a keystone $k_I \in \mathbb{Z}_q$ is randomly selected, this algorithm outputs $f_I = H_1(k_I)$ as the keystone fix.
- ASIGN: The ASIGN algorithm accepts the following parameters $(ID_i, ID_j, S_{ID_i}, \alpha, f, m)$, where S_{ID_i} is the private key associated with $Q_{ID_i}, \alpha, f \in \mathcal{F}$ and $m \in \mathcal{M}$. The algorithm will perform the following.
  - Select a random point $Z \in \mathbb{G}_1$.
  - Set $u_j = \alpha \cdot f$.
  - Compute $u_0 = H_1(H_2(m)||(ID_i \oplus ID_j)||\hat{e}(Z, P)\hat{e}(u_j Q_{ID_j}, P_{pub}))$.
  - Compute $V = u_0^{-1}(Z - (u_0 - u_j)S_{ID_i})$.
  - Output $\sigma = (u_i = u_0 - u_j, u_j, V)$ as the signature on message m.
- ENC-MATCHING-KEYSTONE: Assuming a keystone $k_M \in \mathbb{Z}_q$ is randomly selected, this algorithm outputs $K_M = k_M P$ as the encrypted keystone.
- FIX-SECRET-KEYSTONE: This algorithm returns $f_S = H_1(\hat{e}(K_M, S_{ID_j}))$.
- AVERIFY: The algorithm accepts (σ, ID_i, ID_j, m), where $\sigma = (u_i, u_j, V)$, and verifies whether

$$u_i + u_j \overset{?}{=} H_1(H_2(m)||(ID_i \oplus ID_j)||\hat{e}(V, P)^{u_i+u_j}\hat{e}(u_i Q_{ID_i}, P_{pub})\hat{e}(u_j Q_{ID_j}, P_{pub}))$$

  holds with equality. If so, then output *accept*. Otherwise, output *reject*.
- VERIFY-INITIAL-KEYSTONE: This algorithm outputs *accept* if $f_I = H_1(k_I)$, *reject* otherwise.
- VERIFY-SECRET-KEYSTONE: It outputs *accept* if $f_S = H_1(\hat{e}(P_{pub}, Q_{ID_j})^{k_M})$, *reject* otherwise.
- VERIFY-CONNECTION: This algorithm outputs *accept* if $u_j = f_I$ and $u_i' = u_j \cdot f_S$, *reject* otherwise.
- VERIFY: The algorithm accepts (k_I, k_M, S'), where $k_I \in \mathcal{K}_I$ and $k_M \in \mathcal{K}_M$ are the keystones and $S' = (\sigma_i, \sigma_j, ID_i, ID_j, m_i, m_j)$. The algorithm verifies whether (k_I, k_M) is valid and the connection between σ_i and σ_j is valid by using the above three algorithms. If it does not hold, then output *reject*. Otherwise, run AVERIFY(S). The output of VERIFY is the output of AVERIFY algorithm.

Concurrent Signature Protocol

1. Alice performs the following
   - Picks a random keystone $(k_I, k_M) \in \mathbb{Z}_q \times \mathbb{Z}_q$.
   - Computes keystone fix $f_I = H_1(k_I)$.
   - Selects a message $m_A \in \mathcal{M}$, computes her ambiguous signature as $\sigma_A = (u_A, u_B, V) \leftarrow \mathsf{ASIGN}(ID_A, ID_B, S_{ID_A}, 1, f_I, m_A)$.
   - Computes encrypted keystone $K_M = k_M P$.
   - Sends σ_A and K_M to Bob.
2. Bob performs the following
   - Verifies the signature σ_A by testing whether $\mathsf{AVERIFY}(\sigma_A, ID_A, ID_B, m_A) =$ *accept*. Aborts if the equation does not hold.

  - Computes secret matching keystone fix $f_S = H_1(\hat{e}(K_M, S_{ID_B}))$
  - Selects a message $m_B \in \mathcal{M}$, and computes his ambiguous signature as $\sigma_B = (u'_B, u'_A, V') \leftarrow \mathsf{ASIGN}(ID_B, ID_A, S_{ID_B}, u_B, f_S, m_B)$.
  - Sends σ_B and f_S to Alice.

3. Alice verifies σ_B by testing whether $f_S = H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})$, $u'_A = u_B \cdot f_S$, and $\mathsf{AVERIFY}(\sigma_B, ID_B, ID_A, m_B) = accept$ are held. If not, then Alice aborts. Otherwise, Alice releases the keystone (k_I, k_M) to Bob and both signatures are binding concurrently.

2.3 Chow et al.'s Scheme 2

Concurrent Signature Algorithms

- SETUP: Basically it is the same as Scheme 1, but the description of spaces becomes $\mathcal{M} = \mathcal{K}_I = \mathcal{K}_M = \mathbb{Z}_q, \mathcal{F} = \mathcal{K}' = \mathbb{G}_1$.
- EXTRACT: The same as Scheme 1.
- FIX-INITIAL-KEYSTONE: Assuming a keystone $k_I \in \mathbb{Z}_q$ is randomly selected, this algorithm outputs $f_I = H_2(k_I)$ as the keystone fix.
- ASIGN: The input of this algorithm includes two identities ID_i and ID_j, a private key S_{ID_i}, a message m and $\alpha, f \in \mathbb{G}_1$.
  - Compute $U_j = \alpha + f$ and $h_j = H_1(m||(ID_i \oplus ID_j)||U_j)$.
  - Choose $r'_i \in \mathbb{Z}_q^*$ randomly, compute $U_i = r'_i Q_{ID_i} - U_j - h_j Q_{ID_j}$.
  - Compute $h_i = H_1(m||(ID_i \oplus ID_j)||U_i)$ and $V = (h_i + r'_i) S_{ID_i}$.
  - Output the signature $\sigma = \{U_i, U_j, V\}$.
- ENC-MATCHING-KEYSTONE: The same as Scheme 1.
- FIX-SECRET-KEYSTONE: This algorithm returns $f_S = H_2(\hat{e}(K_M, S_{ID_j}))$.
- AVERIFY: The input of this algorithm includes two identities ID_i and ID_j, a message m, and a ring signature $\sigma = \{U_i, U_j, V\}$.
  - Compute $h_i = H_1(m||(ID_i \oplus ID_j)||U_i)$ and $h_j = H_1(m||(ID_i \oplus ID_j)||U_j)$.
  - Return *accept* if $\hat{e}(P_{pub}, U_i + h_i Q_{ID_i} + U_j + h_j Q_{ID_j}) = \hat{e}(P, V)$, *reject* otherwise.
- VERIFY-INITIAL-KEYSTONE: This algorithm outputs *accept* if $f_I = H_2(k_I)$, *reject* otherwise.
- VERIFY-SECRET-KEYSTONE: It outputs *accept* if $f_S = H_2(\hat{e}(P_{pub}, Q_{ID_j})^{k_M})$, *reject* otherwise.
- VERIFY-CONNECTION: This algorithm outputs *accept* if $U_j = f_I$ and $U'_i = U_j + f_S$, *reject* otherwise.
- VERIFY: The algorithm accepts (k_I, k_M, S'), where $k_I \in \mathcal{K}_I$ and $k_M \in \mathcal{K}_M$ are the keystones and $S' = (\sigma_i, \sigma_j, ID_i, ID_j, m_i, m_j)$. The algorithm verifies whether (k_I, k_M) is valid and the connection between σ_i and σ_j is valid by using the above three algorithms. If it does not hold, then output *reject.* Otherwise, run $\mathsf{AVERIFY}(S)$. The output of VERIFY is the output of AVERIFY algorithm.

Concurrent Signature Protocol

1. Alice performs the following
   - Picks a random keystone $(k_I, k_M) \in \mathcal{K}_I \times \mathcal{K}_M$.
   - Computes keystone fix $f_I = H_2(k_I)$.
   - Selects a message $m_A \in \mathcal{M}$, computes her ambiguous signature as $\sigma_A = (U_A, U_B, V) \leftarrow \mathsf{ASIGN}(ID_A, ID_B, S_{ID_A}, \mathcal{O}_{\mathcal{F}}, f_I, m_A)$, where $\mathcal{O}_{\mathcal{F}}$ denotes the identity element of the group $\mathcal{F}$.
   - Computes encrypted keystone $K_M = k_M P$.
   - Sends σ_A and K_M to Bob.
2. Bob performs the following
   - Verifies the signature σ_A by testing whether $\mathsf{AVERIFY}(\sigma_A, ID_A, ID_B, m_A) \overset{?}{=}$ *accept* holds. Aborts if the equation does not hold.
   - Computes secret matching keystone fix $f_S = H_2(\hat{e}(K_M, S_{ID_B}))$
   - Selects a message $m_B \in \mathcal{M}$, and computes his ambiguous signature as $\sigma_B = (U'_B, U'_A, V') \leftarrow \mathsf{ASIGN}(ID_B, ID_A, S_{ID_B}, U_B, f_S, m_B)$.
   - Sends σ_B and f_S to Alice.
3. Alice verifies σ_B by testing whether $f_S = H_2(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})$, $U'_A = U_B + f_S$ and $\mathsf{AVERIFY}(\sigma_B, ID_B, ID_A, m_B) =$ *accept* are held. If not, then Alice aborts. Otherwise, Alice releases the keystone (k_I, k_M) to Bob and both signatures are binding concurrently.

3 Attacks on the Fairness of Chow et al.'s Schemes

In Chow et al.'s schemes, both keystones k_I and k_M are chose by the initial signer Alice, so she can cheat the matching signer Bob by carefully choosing some communication values. We will show that Alice can perform the signature protocol with Bob on messages m_A and m_B, but outputs a valid signature pair $(\widetilde{\sigma}, \sigma_B)$ on $(\widetilde{m}, m_B)$ with $\widetilde{m} \neq m_A$. Bob can still obtain a signature pair (σ_A, σ_B) on (m_A, m_B), but (σ_A, σ_B) can not be accepted by verifying algorithm VERIFY.

3.1 Attacks against Chow et al.'s Scheme 1

Attack 1 against Chow et al.'s Scheme 1 In Chow et al.'s Scheme 1, if Alice let f_I be equal to $H_1(k_I) f'_S f^{-1}$ instead of $H_1(k_I)$ and let $K_M = k'P$, where $f'_S = H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})$, $f = H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k'})$, then Bob will send a signature $\sigma_B = (u'_B, u'_A, V')$ with $u'_A = H_1(k_I) f'_S$ back to Alice. Then Alice can generate a new signature $\widetilde{\sigma} = (\widetilde{u}_A, \widetilde{u}_B, \widetilde{V})$ on a new message $\widetilde{m}$ with $\widetilde{u}_B = H_1(k_I)$. The result is: the signature pair $(\widetilde{\sigma}, \sigma_B)$ with keystone (k_I, k_M) should be accepted by VERIFY while the signature pair (σ_A, σ_B) with keystone (k_I, k_M) should be rejected by VERIFY since the outputs of $\mathsf{VERIFY\text{-}INITIAL\text{-}KEYSTONE}$ and $\mathsf{VERIFY\text{-}CONNECTION}$ are "*reject*". Following is the detail.

1. Alice performs the following
   - Picks three random keystones $k_I, k_M, k' \in \mathbb{Z}_q$.

  - Computes $f'_S = H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})$, $f = H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k'})$.
  - Computes keystone fix $f_I = H_1(k_I)f'_S f^{-1}$.
  - Selects a message $m_A \in \mathcal{M}$, computes her ambiguous signature as $\sigma_A = (u_A, u_B, V) \leftarrow \mathsf{ASIGN}(ID_A, ID_B, S_{ID_A}, 1, f_I, m_A)$.
  - Computes encrypted keystone $K_M = k'P$.
  - Sends σ_A and K_M to Bob.
2. Bob performs the same as that of the original scheme. Note that, in this case, the secret matching keystone fix $f_S = H_1(\hat{e}(K_M, S_{ID_B})) = f$, so Bob will return a signature $\sigma_B = (u'_B, u'_A, V')$ with $u'_A = H_1(k_I)f'_S$.
3. Alice verifies Bob's ambiguous signature σ_B. If it is invalid, then she aborts. Otherwise, she performs following attack.
  - Computes keystone fix $\widetilde{f} = H_1(k_I)$.
  - Selects a new message $\widetilde{m} \in \mathcal{M}$, computes her ambiguous signature as $\widetilde{\sigma} = (\widetilde{u}_A, \widetilde{u}_B, \widetilde{V}) \leftarrow \mathsf{ASIGN}(ID_A, ID_B, S_{ID_A}, 1, \widetilde{f}, \widetilde{m})$.
  - Releases the keystone (k_I, k_M) publicly, and both signatures $\widetilde{\sigma}$ and σ_B are binding concurrently.

Remark: By releasing $k_I, k_M, K = \hat{e}(K_M, S_{ID_B})$ such that $f_I = H_1(k_I)$ $H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})H_1(K)^{-1}$, Bob can prove that the signature σ_A was indeed issued by Alice, but he can not make σ_A accepted by VERIFY. According to the definition, σ_A is invalid. We would like to point out that the attack above is harmful. After received a valid signature pair $(\widetilde{\sigma}, \sigma_B)$, one may not have his wits about the potential cheat.

Attack 2 against Chow et al.'s Scheme 1 We can also implement a attack against Scheme 1 by letting $f_I = H_1(k_I)H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})^{-1}$. In this case, Bob will send a signature $\sigma_B = (u'_B, u'_A, V')$ with $u'_A = H_1(k_I)$ back to Alice. In the end, Alice produces a new signature $\widetilde{\sigma} = (\widetilde{u}_A, \widetilde{u}_B, \widetilde{V})$ on a new message $\widetilde{m}$ with $\widetilde{u}_B = H_1(k_I)H_1(\hat{e}(P_{pub}, Q_{ID_A})^{\widetilde{k}})$. The signature pair $(\sigma_B, \widetilde{\sigma})$ with keystone $(k_I, \widetilde{k})$ should be accepted by VERIFY. Here Bob is the initial signer while Alice is the matching signer. Whereas the signature pair (σ_A, σ_B) with keystone $(k_I, \widetilde{k})$ should be rejected by VERIFY. The detail is as follows.

1. Alice performs the following
  - Picks a random keystone $(k_I, k_M) \in \mathbb{Z}_q \times \mathbb{Z}_q$.
  - Computes $f = H_1(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})$ and keystone fix $f_I = H_1(k_I)f^{-1}$.
  - Selects a message $m_A \in \mathcal{M}$, computes her ambiguous signature as $\sigma_A = (u_A, u_B, V) \leftarrow \mathsf{ASIGN}(ID_A, ID_B, S_{ID_A}, 1, f_I, m_A)$.
  - Computes encrypted keystone $K_M = k_M P$.
  - Sends σ_A and K_M to Bob.
2. Bob performs the same as that of the original scheme. Note that, in this case, Bob will return a signature $\sigma_B = (u'_B, u'_A, V')$ with $u'_A = H_1(k_I)$.
3. Alice verifies Bob's ambiguous signature σ_B. If it is invalid, then she aborts. Otherwise, she performs following attack.
  - Picks a new random keystone $\widetilde{k} \in \mathbb{Z}_q$.

- Computes keystone fix $\widetilde{f} = H_1(\hat{e}(P_{pub}, Q_{ID_A})^{\tilde{k}})$.
- Selects a new message $\widetilde{m} \in \mathcal{M}$, computes her ambiguous signature as $\widetilde{\sigma} = (\widetilde{u}_A, \widetilde{u}_B, \widetilde{V}) \leftarrow \mathsf{ASIGN}(ID_A, ID_B, S_{ID_A}, u'_A, \widetilde{f}, \widetilde{m})$.
- Releases the keystone $(k_I, \widetilde{k})$ publicly, and both signatures σ_B and $\widetilde{\sigma}$ are binding concurrently.

3.2 Attacks against Chow et al.'s Scheme 2

Attack 1 against Chow et al.'s Scheme 2 Similar to the Attack 1 against Scheme 1, we can also implement a attack against Scheme 2. In this case, we let f_I be equal to $H_2(k_I) + f'_S - f$ instead of $H_2(k_I)$ and let $K_M = k'P$, where $f'_S = H_2(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})$, $f = H_2(\hat{e}(P_{pub}, Q_{ID_B})^{k'})$. Then Bob will send a signature $\sigma_B = (U'_B, U'_A, V')$ with $U'_A = H_2(k_I) + f'_S$ back to Alice. Finally, Alice generates a new signature $\widetilde{\sigma} = (\widetilde{U}_A, \widetilde{U}_B, \widetilde{V})$ on a new message $\widetilde{m}$ with $\widetilde{U}_B = H_1(k_I)$. One can see that the signature pair $(\widetilde{\sigma}, \sigma_B)$ with keystone (k_I, k_M) should be accepted by VERIFY while the signature pair (σ_A, σ_B) with keystone (k_I, k_M) should be rejected by VERIFY. The detail is similar to that of Attack 1 of Scheme 1. Due to space limitation, we omit it here.

Attack 2 against Chow et al.'s Scheme 2 Similar to the Attack 2 against Scheme 1, let $f_I = H_2(k_I) - H_2(\hat{e}(P_{pub}, Q_{ID_B})^{k_M})$. Then Bob will send a signature $\sigma_B = (U'_B, U'_A, V')$ with $U'_A = H_2(k_I)$ back to Alice. In the end, Alice produces a new signature $\widetilde{\sigma} = (\widetilde{U}_A, \widetilde{U}_B, \widetilde{V})$ on a new message $\widetilde{m}$ with $\widetilde{U}_B = H_1(k_I)H_1(\hat{e}(P_{pub}, Q_{ID_A})^{\tilde{k}})$. The signature pair $(\sigma_B, \widetilde{\sigma})$ with keystone $(k_I, \widetilde{k})$ is valid while the signature pair (σ_A, σ_B) with keystone $(k_I, \widetilde{k})$ is invalid. We omit the detail due to the same reason.

4 Definition of Perfect ID-Based Concurrent Signatures

The definition of ID-based concurrent signatures given by [6] implies that two keystones k_I and k_M are chose by the initial signer. As we have shown above, it may cause unfair. In this section, we present a formal definition of ID-based concurrent signatures which redress the flaw of Chow et al.'s definition.

4.1 Concurrent ID-Based Signature Algorithm

A concurrent signature protocol involves two parties Alice and Bob. Since one party needs to create the keystone fix and send the first ambiguous signature, we call this party the *initial signer*. The party who responds to this initial signature by creating another ambiguous signature we call the *matching signer*.

Definition 1. *A perfect ID-based concurrent signature scheme is a digital signature scheme that consists of the following algorithms:*

- SETUP*: A probabilistic algorithm that on input a security parameter l, outputs the system parameters* `params` *which is the descriptions of the message space $\mathcal{M}$, the signature space $\mathcal{S}$, the private key space $\mathcal{K}_{sk}$, the keystone-pair space $\mathcal{K}_I \times \mathcal{K}_M$, the keystone fix space $\mathcal{F}$, the encrypted keystone space $\mathcal{K}'$ and any additional system parameters π. The algorithm also outputs an initial-keystone-fix function $F_I : \mathcal{K}_I \rightarrow \mathcal{F}$, a matching-keystone-fix function $F_M : \mathcal{K}_M \times \mathcal{F} \rightarrow \mathcal{F}$, a keystone encryption function $Enc : \mathcal{K}_M \rightarrow \mathcal{K}'$ and a keystone decryption function $Dec : \mathcal{K}' \times \mathcal{K}_{sk} \rightarrow \mathcal{K}_M$. (Note that we do not include* params *explicitly as the input in the following descriptions.)*
- $\mathsf{EXTRACT}$*: A probabilistic algorithm that on inputs a participant's identity ID, outputs a public key Q_{ID} and the corresponding private key S_{ID}.*
- ASIGN*: A probabilistic algorithm that on inputs $(ID_i, ID_j, S_{ID_i}, f, m)$, where $f \in \mathcal{F}, ID_i, ID_j$ are the identities of the participants, S_{ID_i} is the private key associated with ID_i and $m \in \mathcal{M}$, outputs an ambiguous signature $\sigma = (u_i, u_j, V)$ on m.*
- $\mathsf{AVERIFY}$*: A deterministic algorithm that takes as input $S = (\sigma, ID_i, ID_j, m)$ and outputs* accept *or* reject*.*
- VERIFY*: A deterministic algorithm that takes as input (k_I, k_M, S'), where $(k_I, k_M) \in \mathcal{K}_I \times \mathcal{K}_M, S' = (\sigma_I, \sigma_M, ID_I, ID_M, m_I, m_M)$, and outputs* accept *or* reject*.*

4.2 Concurrent Signature Protocol

The concurrent signature protocol works as follows.

- The initial signer performs the following.
  - Picks a random keystone $k_I \in \mathcal{K}_I$, and computes the corresponding keystone fix $f_I = F_I(k_I)$.
  - Picks a message $m_I \in \mathcal{M}$ and computes her ambiguous signature $\sigma_I = (u_I, u_M, V) = \mathsf{ASIGN}(ID_I, ID_M, S_{ID_I}, f_I, m_I)$.
  - Sends σ_I to the matching signer.
- The matching signer performs the following.
  - Verifies σ_I by checking whether $\mathsf{AVERIFY}(\sigma_I, ID_I, ID_M, m_I) = accept$. If not, he aborts.
  - Picks a random keystone $k_M \in \mathcal{K}_M$, and computes the corresponding keystone fix $f_M = F_M(k_M, u_M)$.
  - Picks a message $m_M \in \mathcal{M}$ and computes his ambiguous signature $\sigma_M = (u'_M, u'_I, V') = \mathsf{ASIGN}(ID_M, ID_I, S_{ID_M}, f_M, m_M)$.
  - Computes the encrypted keystone $K_M = Enc(k_M)$.
  - Sends σ_M and K_M back to the initial signer.
- The initial signer performs the following.
  - Computes the matching keystone $k_M = Dec(K_M, S_{ID_I})$.
  - Verifies σ_M by checking whether $\mathsf{AVERIFY}$ $(\sigma_M, ID_M, ID_I, m_M) =$ *accept* and $u'_I = F_M(k_M, f_I)$ are held. If not, she aborts. Otherwise, she releases the keystone pair (k_I, k_M).

4.3 Security Model for Perfect Concurrent Signatures

A secure perfect concurrent signature scheme should have five properties: *correctness, unforgeability, ambiguity, unlinkability* and *fairness.*

Correctness. If a signature σ has been generated correctly by invoking ASIGN algorithm on a message $m \in \mathcal{M}$, $\mathsf{AVERIFY}$ algorithm will return *accept* with an overwhelming probability, given a signature σ on m and a security parameter l. After the keystone pair $(k_I, k_M) \in \mathcal{K}_I \times \mathcal{K}_M$ is released, the output of VERIFY algorithm will be *accept* with an overwhelming probability.

Unforgeability. Unforgeability for a concurrent signature under a chosen message attack is defined by the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

- **Setup:** $\mathcal{C}$ runs SETUP for a given security parameter l to obtain the system parameters *params*.
- **Queries:** $\mathcal{A}$ can make the following types of query to the challenger $\mathcal{C}$:
  - **Hash Function Query:** $\mathcal{A}$ can request a value of Hash function for any input. $\mathcal{C}$ computes and outputs the value of the Hash function for the requested input.
  - **EXTRACT Query:** $\mathcal{A}$ can request a private key for any input ID. $\mathcal{C}$ runs $\mathsf{EXTRACT}$ and outputs the corresponding private key S_{ID}.
  - F_I **Query:** $\mathcal{A}$ can request that $\mathcal{C}$ select a keystone k_I and return the keystone fix $f_I = F_I(k_I)$. If $\mathcal{A}$ wishes to choose his own keystone k_I, then he can request the keystone fix $f_I = F_I(k_I)$ by a F_I Query with input k_I.
  - F_M **Query:** $\mathcal{A}$ can request a matching keystone fix corresponding to a keystone fix $f \in \mathcal{F}$. $\mathcal{C}$ selects a keystone k_M and returns the keystone fix $f_M = F_M(k_M, f)$. If $\mathcal{A}$ wishes to choose his own keystone k_M, then he can request the keystone fix $f_M = F_M(k_M, f)$ by a F_M Query with input (k_M, f).
  - F_I **Reveal Query:** $\mathcal{A}$ can request that $\mathcal{C}$ reveal the keystone k_I that he used to produce a keystone fix $f_I \in \mathcal{F}$ in a previous F_I query. If f_I was not a previous F_I output, then $\mathcal{C}$ outputs invalid. Otherwise, $\mathcal{C}$ outputs k_I where $f_I = F_I(k_I)$.
  - F_M **Reveal Query:** $\mathcal{A}$ can request that $\mathcal{C}$ reveal the keystone k_M and the keystone fix f that he used to produce a keystone fix $f_M \in \mathcal{F}$ in a previous F_M query. If f_M was not a previous F_M output, then $\mathcal{C}$ outputs invalid. Otherwise, $\mathcal{C}$ outputs k_M and f such that $f_M = F_I(k_M, f)$.
  - **ASIGN Query:** $\mathcal{A}$ can request an ambiguous signature for any input of the form (ID_i, ID_j, f_i, m_i) where $f_i \in \mathcal{F}$, $ID_i, ID_j \neq ID_i$ are the identities of the participants and $m_i \in \mathcal{M}$. $\mathcal{C}$ responds with an ambiguous signature $\sigma = (u_i, u_j, V)$.
- **Output:** Finally, $\mathcal{A}$ outputs a tuple $\sigma = (u_c, u_d, V)$ where $u_c, u_d \in \mathcal{F}$, along with identities ID_c and ID_d, and a message $m \in \mathcal{M}$.
  The adversary wins the game if $\mathsf{AVERIFY}(\sigma, ID_c, ID_d, m)=$ *accept*, and if either of the following two cases hold:

- No ASIGN query with input either of the tuples (ID_c, ID_d, u_d, m) or (ID_d, ID_c, u_c, m) was made by $\mathcal{A}$ and no EXTRACT query was made by $\mathcal{A}$ on either ID_c or ID_d.
- No ASIGN query with input of the tuples (ID_c, ID_i, u_d, m) was made by $\mathcal{A}$ for any $ID_i \neq ID_c$, no EXTRACT query was made by $\mathcal{A}$ on ID_c and either u_d was a previous output from a F_I (or F_M) query or $\mathcal{A}$ produces a keystone k such that $u_d = F_I(k)$ (or (k, k') such that $u_d = F_M(k, F_I(k')))$.

Definition 2. *We say that an ID-based concurrent signature scheme is existentially unforgeable under a chosen message attack if the probability of success of any polynomially bounded adversary in the above game is negligible.*

Ambiguity. Ambiguity for a concurrent signature scheme is defined by the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

- **Setup:** This is as above in the unforgeability game.
- **Phase 1:** $\mathcal{A}$ makes a sequence of EXTRACT, F_I, F_M, F_I Reveal, F_M Reveal, ASIGN queries. These queries are answered by $\mathcal{C}$ as in the above unforgeability game.
- **Challenge:** $\mathcal{A}$ selects a challenge tuple (ID_i, ID_j, m) where ID_i, ID_j are the identities of the participants and $m \in \mathcal{M}$. In response, $\mathcal{C}$ randomly selects $k \in \mathcal{K}_I$ and computes $f = F_I(k)$ or randomly selects $(k, k') \in \mathcal{K}_M \times \mathcal{K}_I$ and computes $f = F_M(k, F_I(k'))$ (each with probability of 1/2), then randomly selects a bit $b \in \{0, 1\}$. $\mathcal{C}$ outputs $\sigma = \mathsf{ASIGN}(ID_i, ID_j, S_{ID_i}, f, m)$ if $b = 0$; otherwise $\mathcal{C}$ outputs $\sigma = \mathsf{ASIGN}(ID_j, ID_i, S_{ID_j}, f, m)$. Denoted the outputed signature σ by (u_1, u_2, V)
- **Phase 2:** $\mathcal{A}$ may make another sequence of queries as in Phase 1; these are handled by $\mathcal{C}$ as before.
- **Output:** Finally, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$. $\mathcal{A}$ wins if $b' = b$ and $\mathcal{A}$ has made neither F_I Reveal nor F_M Reveal query on any of the values u_1, u_2.

Definition 3. *We say that an ID-based concurrent signature scheme is ambiguous if no polynomially bounded adversary has non-negligibly advantage of winning in the above game.*

Unlinkability. Unlinkability for a concurrent signature scheme is defined by the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

- **Setup:** This is as above in the unforgeability game.
- **Phase 1:** $\mathcal{A}$ makes a sequence of EXTRACT, F_I, F_M, F_I Reveal, F_M Reveal, ASIGN queries. These queries are answered by $\mathcal{C}$ as in the above unforgeability game.
- **Challenge:**
  - $\mathcal{A}$ selects a challenge tuple $(ID_i, ID_j, m_{i0}, m_{i1}, \sigma_{i0}, \sigma_{i1})$ such that $\mathsf{AVERIFY}(\sigma_{ia}, ID_i, ID_j, m_{ia})=$ *accept* for $a = 0, 1$, where $\sigma_{ia} = (u_{ia}, u_{ja}, V_{ia})$.

  - $\mathcal{C}$ randomly selects $b \in \{0,1\}, k_j \in \mathcal{K}_M$ and computes $f_j = F_M(k_j, u_{jb})$.
  - $\mathcal{C}$ selects $m_j \in \mathcal{M}$, outputs $\sigma_j = (u'_j, u'_i, V') = \mathsf{ASIGN}(ID_j, ID_i, S_{ID_j}, f_j, m_j)$.
- **Phase 2:** $\mathcal{A}$ may make another sequence of queries as in Phase 1; these are handled by $\mathcal{C}$ as before.
- **Output:** Finally $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{A}$ wins if $b' = b$ and $\mathcal{A}$ has not made any F_M Reveal query on f_j.

Definition 4. *We say that an ID-based concurrent signature scheme is unlinkable if no polynomially bounded adversary has non-negligibly advantage of winning in the above game.*

Fairness. For the case of one keystone only, Chen et al. presented a formal definition of fairness in [5], but it is no longer appropriate in the case of two or more keystones. Chow et al. [6] did not present any new definition of fairness, as a result, their schemes are unfair. Here we present a formal definition of fairness for the case of two keystones. Our definition uses the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

- **Setup:** This is as above in the unforgeability game.
- **EXTRACT, F_I, F_M, F_I Reveal, F_M Reveal Queries:** These queries are answered by $\mathcal{C}$ as in the above unforgeability game.
- **ASIGN Queries:** In the case of fairness, we have two ASIGN Queries as follows.
  - **IASIGN Query:** $\mathcal{A}$ can request an ambiguous signature for any input of the form (ID_i, ID_j, m_i) where $ID_i, ID_j \neq ID_i$ are the identities of the participants and $m_i \in \mathcal{M}$. $\mathcal{C}$ first gets a keystone-fix f_i by a F_I Query and then returns an ambiguous signature $\sigma = (u_i, u_j, V)$.
  - **MASIGN Query:** $\mathcal{A}$ can request an ambiguous signature for any input of the form (ID_i, ID_j, m_i, f) where $ID_i, ID_j \neq ID_i$ are the identities of the participants, $m_i \in \mathcal{M}$ and $f \in \mathcal{F}$. $\mathcal{C}$ first gets a keystone-fix f_i by a F_M Query with input f and then returns an ambiguous signature $\sigma = (u_i, u_j, V)$.
- **Output:** Finally, $\mathcal{A}$ chooses the challenge identities ID_c and ID_d, outputs a signature $\sigma = (u, f, V)$ along with a message m.
  The adversary wins the game if $\mathsf{AVERIFY}(\sigma, ID_c, ID_d, m) = accept$ and if either of the following cases hold:
  - σ is a previous output from a IASIGN Query. No F_I Reveal query on f was made and $\mathcal{A}$ produces a keystone k such that $f = F_I(k)$ or produces a keystone pair (k, k') such that $f = F_M(k, F_I(k'))$.
  - σ is a previous output from a MASIGN Query. $\mathcal{A}$ produces a keystone k such that $f = F_I(k)$ or produces keystones k_1, k_2 and k' such that $f = F_M(k_1, F_I(k_2)) = F_M(k', f')$ but $f' \neq F_I(k_2)$, where f' is the keystone-fix in the input of MASIGN Query, in which $\mathcal{A}$ gets the signature σ.

Definition 5. *We say that an ID-based concurrent signature scheme is fair if a polynomially bounded adversary's probability of success in the above game is negligible.*

Definition 6. *A ID-based concurrent signature scheme is secure if it is existentially unforgeable under a chosen message attack, correct, ambiguous, unlinkable and fair.*

5 Improved ID-Based Perfect Concurrent Signature Schemes

In Chow et al.'s schemes, both keystones k_I and k_M are chose by Alice, as a result, Alice can cheat Bob by carefully choosing the keystone fix f_I. If keystones k_I and k_M are chose by Alice and Bob, respectively, all attacks above can be avoided. We present two simple but significant improvements as follows.

5.1 Improved Scheme 1

Concurrent Signature Algorithms

- SETUP:
  - $\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, P, P_{pub}, H_0, H_1, \mathcal{M}, \mathcal{F}, \mathcal{K}'$ are the same as that of the original scheme.
  - Sets $\mathcal{K}_I = \mathcal{K}_M = \mathbb{G}_2$.
  - Sets $F_I : \mathbb{G}_2 \to \mathbb{Z}_q$ be a one-way permutation.
  - Sets $F_M(x, y) = F_I(x) + y \pmod{q}$.
  - Sets $Enc(k) = kP$.
  - Sets $Dec(K', K'') = \hat{e}(K', K'')$.
- EXTRACT: The same as that of the original scheme.
- ASIGN: The algorithm accepts $(ID_i, ID_j, S_{ID_i}, f_i, m_i)$ and performs the following.
  - Selects a random point $Z \in \mathbb{G}_1$.
  - Computes $u_0 = H_1(H_0(m)||(ID_i \oplus ID_j)||\hat{e}(Z, P)\hat{e}(f_i Q_{ID_j}, P_{pub}))$.
  - Computes $V = u_0^{-1}(Z - (u_0 - u_j)S_{ID_i})$.
  - Sets $u_i = u_0 - f_i \pmod{q}, u_j = f_i$.
  - Outputs $\sigma = (u_i, u_j, V)$ as the signature on message m.
- AVERIFY: The same as that of the original scheme.
- VERIFY: The algorithm accepts (k_i, k_j, S'), where $k_i \in \mathcal{K}_I$ and $k_j \in \mathcal{K}_M$ are the keystones and $S' = (\sigma_i, \sigma_j, ID_i, ID_j, m_i, m_j)$. The algorithm verifies whether $f_i = F_I(k_i), f_j = F_I(k_j) + f_i \pmod{q}$. If not, then outputs *reject*. Otherwise, run AVERIFY on σ_i and σ_j respectively. If both outputs are *accept*, then outputs *accept*. Otherwise, outputs *reject*.

Concurrent Signature Protocol

1. Alice performs the following
   - Picks a random keystone $k_I \in \mathbb{G}_2$, computes keystone fix $f_I = F_I(k_I)$.
   - Selects a message $m_I \in \mathcal{M}$, computes her ambiguous signature as $\sigma_I = (u_I, u_M, V) \leftarrow \text{ASIGN}(ID_I, ID_M, S_{ID_I}, f_I, m_I)$.

- Sends σ_I to Bob.

2. Bob performs the following
   - Verifies the signature σ_I by testing whether $\mathsf{AVERIFY}(\sigma_I,ID_I,ID_M,m_I) =$ *accept*. Aborts if the equation does not hold.
   - Picks a random number $k \in \mathbb{Z}_q$, computes keystone $k_M = \hat{e}(P_{pub}, Q_{ID_I})^k$.
   - Computes encrypted keystone $K_M = kP$.
   - Computes matching keystone fix $f_M = F_I(k_M) + u_j \pmod q$.
   - Selects a message $m_M \in \mathcal{M}$, and computes his ambiguous signature as $\sigma_M = (u'_M, u'_I, V') \leftarrow \mathsf{ASIGN}(ID_M, ID_I, S_{ID_M}, f_M, m_M)$.
   - Sends σ_M and K_M to Alice.
3. Alice verifies σ_M by testing whether
   - $u'_I = F_I(\hat{e}(K_M, S_{ID_I}) + u_M \pmod q)$
   - $\mathsf{AVERIFY}(\sigma_M, ID_M, ID_I, m_M) =$ *accept.*

   If not, then Alice aborts. Otherwise, Alice computes keystone $k_M = \hat{e}(K_M,S_{ID_I})$ and releases the keystone (k_I, k_M), then both signatures are binding concurrently.

5.2 The security

The **correctness** of the improved concurrent signature scheme 1 can easily be verified.

Since we do not make any change to the ASIGN algorithm, the unforgeability of our improved scheme is kept the same as the original scheme. So, the same as that of [6], we have following lemma. Its proof follows the proof of the Lemma 1 in [5] given by Chen et al. Due to space limitation, the proof is omitted.

Lemma 1. ***(Unforgeability)*** *The improved concurrent signature scheme 1 is existentially unforgeable under a chosen message attack in the random oracle model, assuming the hardness of Co-CDH problem.*

Lemma 2. ***(Ambiguity)*** *The improved concurrent signature scheme 1 is ambiguous in the random oracle model.*

Proof. We consider the following distributions:

$$\xi = \left\{ (u_1, u_2, V) \left| \begin{array}{l} Z \in_R \mathbb{G}_1, k \in_R \mathbb{G}_2 \\ u_2 = F_I(k) \\ u_0 = H_1(H_0(m)||(ID_i \oplus ID_j)||\hat{e}(Z,P)\hat{e}(u_2 Q_{ID_j}, P_{pub})) \\ u_1 = u_0 - u_2 \\ V = u_0^{-1}(Z - u_1 S_{ID_i}) \end{array} \right. \right\},$$

and

$$\zeta = \left\{ (u'_2, u'_1, V') \left| \begin{array}{l} Z \in_R \mathbb{G}_1, k \in_R \mathbb{G}_2 \\ u'_2 = F_I(k) \\ u'_0 = H_1(H_0(m)||(ID_i \oplus ID_j)||\hat{e}(Z,P)\hat{e}(u'_2 Q_{ID_i}, P_{pub})) \\ u'_1 = u'_0 - u'_2 \\ V' = u_0'^{-1}(Z - u'_1 S_{ID_j}) \end{array} \right. \right\}.$$

In the random oracle model, the distributions of the outputs of F_I and H_1 are uniform, so two distributions above are the same.

The distribution of $F_I(k) + F_I(k')$ is the same as the distribution of $F_I(k)$, so the case of $f = F_I(k) + F_I(k')$ is the same as that of $f = F_I(k)$.

Hence, the adversary wins the game of Definition 3 with probability exactly 1/2, so the scheme is ambiguous. $\square$

Lemma 3. ***(Unlinkability)*** *The improved concurrent signature scheme 1 is unlinkable.*

Proof. Since $F_I : \mathbb{G}_2 \to \mathbb{Z}_q$ is a one-way permutation, given f_j, f_{i0}, f_{i1}, there exist k_0 and k_1 such that $f_j = F_I(k_0) + f_{i0}$ and $f_j = F_I(k_1) + f_{i1}$, respectively. Such k_0 and k_1 always exist regardless of the values of f_j and (f_{i0}, f_{i1}), so f_j and f_{ib} have exactly the same relation defined by $F_M, (b = 0, 1)$. Therefore, even an infinitely powerful adversary wins the game of Definition 4 with probability exactly 1/2, so the scheme is unconditional unlinkable. $\square$

Lemma 4. ***(Fairness)*** *The improved concurrent signature scheme 1 is fair in the random oracle model.*

Proof. Suppose that there exists an algorithm $\mathcal{A}$ that with probability δ wins the game in Definition 5, we show that δ is negligible. Let μ_I and μ_M are the numbers of F_I and F_M queries made by $\mathcal{A}$ and let μ_{IS} and μ_{MS} are the numbers of IASIGN and MASIGN queries made by $\mathcal{A}$, respectively.

If case 1 of the output conditions occurs, then σ is a previous output from a IASIGN Query and f is a previous output of a F_I query. We discuss the following two cases.

- $\mathcal{A}$ has found a keystone k such that $f = F_I(k)$ but without making any F_I Reveal query on input f. In the random oracle model, $\mathcal{A}$'s probability of producing such a k_c is at most $\mu_{IS}\mu_I/q$.
- $\mathcal{A}$ has found a keystone pair (k, k') such that $f = F_I(k) + F_I(k')$. In the random oracle model, $\mathcal{A}$ can get such (k, k') by F_M query with probability at most $\mu_{IS}\mu_M/q$ or by F_I query with probability at most $\mu_{IS}\mu_I(\mu_I + 1)/2q$, where $\mu_I(\mu_I + 1)/2q$ is the probability of two outputs of F_I queries f' and f'' such that $f = f' + f''$. So, $\mathcal{A}$'s probability of producing such (k, k') is at most $\mu_{IS}(\mu_I(\mu_I + 1) + 2\mu_M)/2q$.

If case 2 of the output conditions occurs, then σ is a previous output from a MASIGN Query. In this case, f is a previous output of a F_M query on input of some f', so $\mathcal{A}$ can get k' and f' such that $f = F_I(k') + f'$ by a F_M Reveal query. We would like to emphasize the fact that, in our scheme, the f' was set before choosing the k', which is the significant difference between our scheme and that of [6]. Thus $\mathcal{A}$ can neither find the required keystone k by setting $f' = F_I(k) - F_I(k')$, nor find the required keystones k_1 and k_2 by setting $f' = F_I(k_1) + F_I(k_2) - F_I(k')$. So the case 2 was reduced to the case 1.

Since μ_I, μ_M, μ_{IS} and μ_{MS} are polynomially bounded in the security parameter $\lceil \log_2 q \rceil$, the probability δ is negligible. $\square$

Theorem 1. *The improved concurrent signature scheme 1 is secure in the random oracle model, assuming the hardness of Co-CDH problem.*

Proof. The proof follows directly from above lemmas. □

5.3 Improved Scheme 2

Concurrent Signature Algorithms

- SETUP:
  - $\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, P, P_{pub}, H_0, H_1, \mathcal{M}, \mathcal{F}, \mathcal{K}'$ are the same as that of the original scheme.
  - Sets $\mathcal{K}_I = \mathcal{K}_M = \mathbb{G}_2$.
  - Sets $F_I : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ be a one-way permutation.
  - Sets $F_M(x, y) = F_I(x) + y \pmod q$.
  - Sets $Enc(k) = kP$.
  - Sets $Dec(K', K'') = \hat{e}(K', K'')$.
- EXTRACT: The same as that of the original scheme.
- ASIGN: The algorithm accepts $(ID_i, ID_j, S_{ID_i}, f_i, m_i)$ and performs the following.
  - Sets $U_j = f_i$, computes $h_j = H_1(m||(ID_i \oplus ID_j)||U_j)$.
  - Chooses $r'_i \in \mathbb{Z}_q^*$ randomly, computes $U_i = r'_i Q_{ID_i} - U_j - h_j Q_{ID_j}$.
  - Computes $h_i = H_1(m||(ID_i \oplus ID_j)||U_i)$ and $V = (h_i + r'_i)S_{ID_i}$.
  - Outputs the signature $\sigma = (U_i, U_j, V)$.
- AVERIFY: The same as that of the original scheme.
- VERIFY: The same as that of the improved scheme 1.

Concurrent Signature Protocol

1. Alice performs the following
   - Picks a random keystone $k_I \in \mathcal{K}_I$, computes keystone fix $f_I = F_I(k_I)$.
   - Selects a message $m_I \in \mathcal{M}$, computes her ambiguous signature as $\sigma_I = (U_I, U_M, V) \leftarrow \mathsf{ASIGN}(ID_I, ID_M, S_{ID_I}, f_I, m_I)$.
   - Sends σ_I to Bob.
2. Bob performs the following
   - Verifies the signature σ_I by testing whether $\mathsf{AVERIFY}(\sigma_I, ID_I, ID_M, m_I) \stackrel{?}{=}$ *accept* holds. Aborts if the equation does not hold.
   - Picks a random number $k \in \mathbb{Z}_q$.
   - Computes keystone $k_M = \hat{e}(P_{pub}, Q_{ID_I})^k$.
   - Computes encrypted keystone $K_M = kP$, computes matching keystone fix $f_M = F_I(k_M) + U_M$
   - Selects a message $m_M \in \mathcal{M}$, and computes his ambiguous signature as $\sigma_M = (U'_M, U'_I, V') \leftarrow \mathsf{ASIGN}(ID_M, ID_I, S_{ID_M}, f_M, m_M)$.
   - Sends σ_M and K_M to Alice.
3. Alice verifies σ_M by testing whether
   - $U'_I = F_I(\hat{e}(K_M, S_{ID_I}) + U_M$

– $\mathsf{AVERIFY}(\sigma_M, ID_M, ID_I, m_M) = accept$
If not, then Alice aborts. Otherwise, Alice computes keystone $k_M = \hat{e}(K_M, S_{ID_I})$ and releases the keystone (k_I, k_M), then both signatures are binding concurrently.

Similar to the improved scheme 1, we have the following theorem. Due to space limitation, we omit its proof.

Theorem 2. *The improved concurrent signature scheme 2 is secure in the random oracle model, assuming the hardness of Co-CDH problem.*

6 Conclusion

Concurrent signatures were introduced as an alternative approach to solving the problem of fair exchange of signatures and several concrete concurrent signature schemes have been proposed. In this paper, we present attacks on the fairness of Chowet al.'s identity-based perfect concurrent signature schemes [6]. We also present a modified definition of ID-based concurrent signatures which redress the flaw of Chow et al.'s definition and propose two improved schemes to fix our attacks.

References

1. N. Asokan, V. Shoup, M. Waidner, Optimistic fair exchange of signatures, in: Advances in Cryptology - EUROCRYPT '98, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, Berlin, 1998, pp. 591 - 606.
2. D. Boneh, M. Naor, Timed commitments (extended abstract), in: Advances in Cryptology - CRYPTO 2000, Lecture Notes in Computer Science, vol. 1880, Springer-Verlag, Berlin, 2000, pp. 236 - 254.
3. D. Boneh, C. Gentry, B. Lynn and H. Shacham, Aggregrate and verifiably encrypted signatures from bilinear maps, in: Advances in Cryptology - EUROCRYPT 2003, Lecture Notes in Computer Science, vol. 2656, Springer-Verlag, Berlin, 2003, pp. 416 - 432.
4. J. Camenisch, V. Shoup, Practical verifiable encryption and decryption of discrete logarithms, in: Advances in Cryptology - CRYPTO 2003, Lecture Notes in Computer Science, vol. 2729, Springer-Verlag, Berlin, 2003, pp. 126 - 144.
5. L. Chen, C. Kudla, K. G. Paterson, Concurrent signatures, in: Advances in Cryptology - EUROCRYPT 2004, Lecture Notes in Computer Science, vol. 3027, Springer-Verlag, Berlin, 2004, pp. 287 - 305.
6. S. Chow, W. Susilo, Generic Construction of (Identity-Based) Perfect Concurrent Signatures, in: Information and Communications Security (ICICS 2005), Lecture Notes in Computer Science, vol. 3783, Springer-Verlag, Berlin, 2005, pp. 194-206.
7. S. Even, O. Goldreich, A. Lempel, A randomized protocol for signing contracts, Commun. ACM, 28(6): 637 - 647 (1985).
8. J. Garay, C. Pomerance, Timed fair exchange of standard signatures, in: Proc. Financial Cryptography 2003, Lecture Notes in Computer Science, vol. 2742, Springer-Verlag, Berlin, 2003, pp. 190 - 207.

9. J. Garay, M. Jakobsson, P. MacKenzie, Abuse-free optimistic contract signing, in: Advances in Cryptology - CRYPTO 1999, Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, Berlin, 1999, pp. 449 - 466.
10. O. Goldreich, A simple protocol for signing contracts, in: Advances in Cryptology - CRYPTO '83, Springer-Verlag, Berlin, 1983, pp. 133 - 136.
11. K. Nguyen, Asymmetric Concurrent Signatures Khanh Nguyen, in: Information and Communications Security (ICICS 2005), Lecture Notes in Computer Science, vol. 3783, Springer-Verlag, Berlin, 2005, pp. 181-193.
12. W. Susilo, Y. Mu, Tripartite Concurrent Signatures. in: The 20th IFIP International Information Security Conference (IFIP/SEC 2005), pp. 425-441, Springer, 2005.
13. W. Susilo, Y. Mu, F. Zhang, Perfect concurrent signature schemes, in: ICICS 2004, Lecture Notes in Computer Science, vol. 3269, Springer-Verlag, Berlin, 2004, pp. 14 - 26.
14. D. Tonien, W. Susilo, R. Safavi-Naini, Multi-party Concurrent Signatures, in: ISC 2006, Lecture Notes in Computer Science, vol. 4176, Springer-Verlag, Berlin, 2006, pp. 131 - 145.