The Fairness of Perfect Concurrent Signatures

Guilin Wang, Feng Bao, and Jianying Zhou

Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore 119613
{glwang, baofeng, jyzhou}@i2r.a-star.edu.sg

Abstract. In Eurocrypt 2004, Chen, Kudla and Paterson introduced the concept of *concurrent signatures*, which allow two parties to produce two ambiguous signatures until the initial signer releases an extra piece of information (called *keystone*). Once the keystone is publicly known, both signatures are bound to their true signers *concurrently*. In ICICS 2004, Susilo, Mu and Zhang further proposed *perfect concurrent signatures* to strengthen the ambiguity of concurrent signatures. That is, even if the both signers are known having issued one of the two ambiguous signatures, any third party is still unable to deduce who signed which signature, different from Chen et al.'s scheme. In this paper, we point out that Susilo et al.'s two perfect concurrent signature schemes are actually *not* concurrent signatures. Specifically, we identify an attack that enables the initial signer to release a carefully prepared keystone that binds the matching signer's signature, but not the initial signer's. Therefore, their schemes are *unfair* for the matching signer. Moreover, we present an effective way to avoid this attack so that the improved schemes are truly perfect concurrent signatures.

Keywords: Concurrent signature, fair exchange, security protocol.

1 Introduction

The concept of *concurrent signatures* was introduced by Chen, Kudla and Paterson in Eurocrypt 2004 [11]. Such signature schemes allow two parties to produce and exchange two ambiguous signatures until an extra piece of information (called *keystone*) is released by one of the parties. More specifically, before the keystone is released, those two signatures are *ambiguous* with respect to the identity of the signing party, i.e., they may be issued either by two parties together or just by one party alone; after the keystone is publicly known, however, both signatures are bound to their true signers *concurrently*, i.e., any third party can validate who signed which signature.

As explained below, concurrent signatures contribute a novel approach for the traditional problem of fair exchange of signatures: Two mutually mistrustful parties want to exchange their signatures in a *fair* way, i.e., after the completion of exchange, either each party gets the other's signature or neither party does. Fair exchange of signatures is widely useful in electronic commerce, like contract signing and e-payment.

According to whether a trusted third party (TTP) is needed in the exchange procedure, there are two essentially different approaches in the literature for the problem of fair exchanging signatures: (a) Gradual exchange without TTP; and (b) Optimal exchange with TTP. Though without the help of a TTP, the first type solutions (e.g., [21,15,13]) impractically assume that both parties have equivalent computation resources, and inefficiently exchange signatures "bit-by-bit" for many interactive rounds. There are many efficient implementations belonging to the second approach, such as verifiably encrypted signatures [6,5,9], escrowed signatures [3,4], convertible signatures [8], and verifiable confirmation of signatures [10] etc. However, all those schemes require a dispute-resolving TTP whose functions are beyond that of a CA (certification authority) in PKI (public key infrastructure). The point is that such an appropriate TTP may be costly or even unavailable to the parties involved.

In [11], Chen et al. remarkably observed that the *full* power of fair exchange is *not necessary* in many applications, since there exist some mechanisms that provide a more natural dispute resolution than the reliance on a TTP. In particular, concurrent signatures can be used as a weak tool to realize practical exchanges, if one of the two parties would like to complete such an exchange. Chen et al. presented several such applications, including one party needing the service of the other, credit card payment transactions, secret information releasing, and fair tendering of contracts. In the following, we only review a concrete example of the first kind application.

Consider a situation where a customer Alice would like to purchase a laptop from a computer shop owned by Bob. For this purpose, Alice and Bob can first exchange their ambiguous signatures via the Internet as follows. As the initial signer, Alice first chooses a keystone, and signs her payment instruction ambiguously to pay Bob the price of a laptop. Upon receiving Alice's signature, Bob as the matching signer agrees this order by signing a receipt ambiguously that authorizes Alice to pick one up from Bob's shop. However, to get the laptop from the shop physically, Alice has to show both Bob's signature and the keystone, because Bob's ambiguous signature alone can be forged by Alice easily. But the point is that once the keystone is released, both of the two ambiguous signatures become bound concurrently to Alice and Bob respectively. Therefore, Bob can present Alice's signature together with the corresponding keystone to get money from bank.

In the above example, Alice indeed has a degree of extra power over Bob, since she controls whether to release the keystone. Actually, this is the exact reason why concurrent signatures can only provide a somewhat weak solution for fair exchange of signatures. In the common real life, however, if Alice does not want to buy a laptop (by releasing the keystone), why she wastes her time to order it. At the same time, by adding a time limit in the receipt, Bob could cancel Alice's order conveniently like the practice in booking air-tickets nowadays. The advantage is that those solutions using concurrent signatures [11,27] can be implemented very efficiently in both aspects of computation and communication, and do not rely on any TTP. Therefore, the shortcomings in traditional solutions for fair exchange of signatures are overcome in a relatively simple and natural way.

In ICICS 2004, Susilo, Mu and Zhang [27] pointed out that in Chen et al.'s concurrent signatures, if the two parties are known to be trustworthy any third party can identify who is the true signer of both ambiguous signatures *before* the keystone is released. To strengthen the ambiguity of concurrent signatures, Susilo et al. further proposed a strong notion called *perfect concurrent signatures*, and presented two concrete constructions from Schnorr signature and bilinear pairing. That is, in their schemes even a third party knows or believes both parties indeed issued one of the two signatures, he/she still cannot deduce who signed which signature, different from Chen et al.'s scheme.

In this paper, we shall point out that Susilo et al.'s perfect concurrent signatures are actually *not* concurrent signatures. Specifically, we identify an attack against their two schemes that enables the initial signer Alice to release a carefully prepared keystone such that the matching signer Bob's signature is binding, but not her. Therefore, both of their two perfect concurrent signature schemes are *unfair* for the matching signer Bob. At the same time, we also address another weakness in their keystone generation algorithm. To avoid those flaws, we present an effective way to improve Susilo et al.'s schemes [27] so that the results are truly perfect concurrent signatures. Moreover, our improvement from Schnorr signature obtains about 50% performance enhancement over their original scheme. In addition, we notice that a similar attack applies to a generic construction of identity-based perfect concurrent signatures, which is proposed by Chow and Susilo in ICICS 2005 [12].

For simplicity, we call PCS1 and PCS2 for Susilo et al.'s two perfect concurrent signatures from Schnorr and bilinear pairing, respectively. Sections 2 presents the security model for perfect concurrent signatures. In Section 3, we review PCS1 and analyze its security. In Section 4, we discuss PCS2 and its security. In Section 5, we present and analyze the improved schemes. Finally, Section 6 concludes the paper.

2 Security Model and Definitions

This section presents a formal model for perfect concurrent signatures, which is adapted from [11,27]. Specifically, a perfect concurrent signature (PCS) scheme works just as a usual concurrent signature scheme but achieves stronger security. That is, besides the requirements of unforgeability and fairness for standard concurrent signatures, a PCS scheme is supposed to satisfy perfect ambiguity (see below) rather than (usual) ambiguity specified in [11].

Definition 1 (Syntax of Concurrent Signatures). *A* `concurrent signature scheme` *consists of the following five algorithms.*

SETUP: *On input a security parameter ℓ, this probabilistic algorithm outputs the descriptions of a tuple $(\mathcal{U}, \mathcal{M}, \mathcal{S}, \mathcal{K}, \mathcal{F})$, where $\mathcal{U}$ is the set of users, $\mathcal{M}$ the* *message space, $\mathcal{S}$ the signature space, $\mathcal{K}$ the keystone space, $\mathcal{F}$ the keystone fix space. The algorithm also outputs the public keys $\{X_i\}$ of all users, while each user keeps the corresponding private key x_i.*

KGEN: *This is a mapping from $\mathcal{K}$ to $\mathcal{F}$, which is called the keystone generation algorithm. Note that* KGEN *should be a one-way function, i.e., it is difficult to derive k from* $\mathsf{KGEN}(k)$.

ASIGN: *On inputs (X_i, X_j, x, f, m), where X_i and X_j are two distinct public keys, x is the private key corresponding to X_i or X_j (i.e. $x = x_i$ or $x = x_j$), $f \in \mathcal{F}$, and $m \in \mathcal{M}$, this probabilistic algorithm outputs an ambiguous signature $\sigma = (c, s_1, s_2) \in \mathcal{S} \times \mathcal{F} \times \mathcal{F}$ on message m, where $s_1 = f$ or $s_2 = f$.*

AVERIFY: *On input $S = (\sigma, X_i, X_j, m)$, where $\sigma = (c, s_1, s_2) \in \mathcal{S} \times \mathcal{F} \times \mathcal{F}$, X_i and X_j are two different public keys, and $m \in \mathcal{M}$, this deterministic algorithm outputs* accept *or* reject*. We also require* AVERIFY *should have the symmetry property, i.e.,* $\mathsf{AVERIFY}(\sigma, X_i, X_j, m) \equiv \mathsf{AVERIFY}(\sigma', X_j, X_i, m)$, *where $\sigma' = (c, s_2, s_1)$ is derived from $\sigma = (c, s_1, s_2)$.*

VERIFY: *On inputs (k, S), where $k \in \mathcal{K}$ and $S = (\sigma, X_i, X_j, m)$, this deterministic algorithm outputs* accept *if* $\mathsf{AVERIFY}(S)$=accept *and the keystone k is valid. Otherwise, it outputs* reject*.*

Please refer to Section 2.2 of [11] for a framework how the concurrent signatures can be generated and exchanged between two mutually mistrustful parties, i.e., their general concurrent signature protocol.

Now, we describe the security requirements for perfect concurrent signatures, i.e., correctness, unforgeability, perfect ambiguity, and fairness. *Correctness* requires that (a) every anonymous signature σ properly generated by ASIGN will be accepted by AVERIFY, and every pair (k, σ) properly generated by KGEN and ASIGN will be accepted by VERIFY. Since this is just a simple and basic requirement, we do not mention it any more. The other three security requirements should be considered under a chosen message attack in the multi-party setting, extended from the existential unforgeability given in [22]. The purpose is to capture an adversary who can simulate and/or observe concurrent signature protocol runs between any pair of users, as noticed in [11]. Informally, *unforgeability* requires any efficient adversary with neither of the corresponding two secret keys cannot forge a valid concurrent signature with non-negligible probability under chosen message attacks. Since this paper focuses on the fairness and perfect ambiguity of concurrent signatures, we just mention the following formal definition of unforgeability without specifying the details of the game between a challenger and an adversary.

Definition 2 (Unforgeability). *A concurrent signature scheme is* ***existentially unforgeable*** *under a chosen message attack in the multi-party model, if the success probability of any polynomially bounded adversary in the game specified in Section 3.2 of [11] is a negligible function of the security parameter ℓ.*

In [11], *ambiguity* means that given a concurrent signature without the keystone, any adversary cannot distinguish who of the two signers issued this signature. This notion is strengthened by Susilo et al. (See Definition 5 in [27]) to capture that there are four cases (not only three cases) for the issuers of two signatures. We call this strengthened notion *perfect ambiguity*, and refine its definition here by providing a full formal specification. That is, the perfect ambiguity of a concurrent signature scheme is formally defined via the following game between an adversary E and a challenger C.

Setup: For a given security parameter ℓ, C runs SETUP to obtain all descriptions of the user set $\mathcal{U}$, the message space $\mathcal{M}$, the signature space $\mathcal{S}$, the keystone space $\mathcal{K}$, the keystone fix space $\mathcal{F}$, and the keystone generation algorithm $\mathsf{KGEN} : \mathcal{K} \to \mathcal{F}$. SETUP also outputs the public and private key pairs $\{(X_i, x_i)\}$ for all users. Then, E is given all the public parameters and the public keys $\{X_i\}$, while C retains the private keys $\{x_i\}$.

Phase 1: E makes a sequence of KGen, $\mathsf{KReveal}$, ASign and $\mathsf{Private\ Key\ Extract}$ queries. These are answered by C as in the unforgeability game (Refer to Section 3.2 of [11]).

Challenge: E selects a challenge tuple (X_i, X_j, m_1, m_2) where X_i and X_j are public keys, and $m_1, m_2 \in \mathcal{M}$ are two messages to be signed. In response, C first selects keystones $k, k' \in_R \mathcal{K}$, computes $f_1 = \mathsf{KGEN}(k)$ and $f_2 = \mathsf{KGEN}(k) + \mathsf{KGEN}(k') \bmod q$. Then, by randomly selecting $b \in \{1, 2, 3, 4\}$, C outputs ambiguous signatures $\sigma_1 = (c, s_1, s_2)$ and $\sigma_2 = (c', s_1', s_2')$ as follows:
- If $b = 1$, $\sigma_1 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_i, f_1, m_1)$, $\sigma_2 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_i, f_2, m_2)$;
- If $b = 2$, $\sigma_1 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_j, f_1, m_1)$, $\sigma_2 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_j, f_2, m_2)$;
- If $b = 3$, $\sigma_1 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_i, f_1, m_1)$, $\sigma_2 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_j, f_2, m_2)$;
- If $b = 4$, $\sigma_1 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_j, f_1, m_1)$, $\sigma_2 \leftarrow \mathsf{ASIGN}(X_i, X_j, x_i, f_2, m_2)$.

Phase 2: E may continue to make another sequence of queries as in Phase 1; these are handled by C as before.

Output: E finally outputs a value $b' \in \{1, 2, 3, 4\}$ as its guess for b. We say E *wins* the game if $b' = b$ and E has not made a $\mathsf{KReveal}$ query on any of the following values: s_1, s_2, $s_1' - f_1$, and $s_2' - f_1$.

Definition 3 (Perfect Ambiguity). *A concurrent signature scheme is called* `perfectly ambiguous` *if no polynomially bounded adversary can win the above game with a probability that is non-negligibly greater than* $1/4$.

Fairness intuitively requires that (1) a concurrent signature scheme should be fair for the initial signer Alice, i.e., only Alice can reveal the keystone; and (2) a concurrent signature scheme should be fair for the matching signer Bob, i.e., once the keystone is released, both signatures are bound to their signers concurrently. This concept is formally defined via the following game (adapted from [11]) between an adversary E and a challenger C:

Setup: This is the same as in the game for perfect ambiguity.

Queries: KGen, $\mathsf{KReveal}$, ASign and $\mathsf{Private\ Key\ Extract}$ queries are answered by C as in the unforgeability game (Section 3.2 of [11]).

Output: E finally chooses the challenge public keys X_c and X_d, outputs a keystone $k \in \mathcal{K}$ and $S = (\sigma, X_c, X_d, m)$ such that $\mathsf{AVERIFY}(S) = \mathsf{accept}$, where $m \in \mathcal{M}$ and $\sigma = (c, s_1, s_2) \in \mathcal{S} \times \mathcal{F} \times \mathcal{F}$. The adversary *wins* the game if either of the following two cases holds:

1. (k, S) is accepted by VERIFY so that $s_2 = \mathsf{KGEN}(k)$ is a previous output from a KGen query but no KReveal query on input s_2 was made.
2. E additionally produces $S' = (\sigma', X_c, X_d, m')$ along with another keystone $k' \in \mathcal{K}$, where $m' \in \mathcal{M}$ and $\sigma' = (c', s_1', s_2') \in \mathcal{S} \times \mathcal{F} \times \mathcal{F}$, such that $\mathsf{AVERIFY}(S') = \mathsf{accept}$ and $s_1' = s_2 + H_1(k')$. Furthermore, (k, k', S') is accepted by VERIFY, but (k, S) is not accepted by VERIFY.

Definition 4 (Fairness). *A concurrent signature scheme is* ***fair*** *if the success probability of any polynomially bounded adversary in the above game is negligible.*

Definition 5. *We say that a correct concurrent signature scheme is* ***secure****, if it is existentially unforgeable, perfectly ambiguous, and fair under a chosen message attack in the multi-party setting.*

3 PSC1 and Its Security

3.1 Review of PSC1

We now review PCS1 [27], which is a concurrent signature scheme derived from Schnorr signature [26]. Susilo et al. constructed PCS1 by using some techniques from ring signatures [24,1], as did by Chen et al. in [11].

- SETUP. On input a security parameter ℓ, the SETUP algorithm first randomly generates two large prime numbers p and q such that $q|(p-1)$, and a generator $g \in \mathbb{Z}_p$ of order q, where q is exponential in ℓ. Then, the SETUP algorithm sets the message space $\mathcal{M}$, the keystone space $\mathcal{K}$, the signature space $\mathcal{S}$, and the keystone fix space $\mathcal{F}$ as follows: $\mathcal{M} = \mathcal{K} = \{0,1\}^*$, and $\mathcal{S} = \mathcal{F} = \mathbb{Z}_q$. It also selects a cryptographic hash function $H_1 : \{0,1\}^* \to \mathbb{Z}_q$, which is used as the message digest function and the keystone generation algorithm. In addition, we assume that $(x_A, y_A = g^{x_A} \bmod p)$ and $(x_B, y_B = g^{x_B} \bmod p)$ are the private/public key pairs of Alice and Bob, respectively.
- ASIGN. The algorithm ASIGN takes input (y_i, y_j, x, s, m), where y_i and y_j are two public keys $(y_i \neq y_j)$, x is the private key corresponding to y_i or y_j (i.e. $x = x_i$ or $x = x_j$), $s \in \mathcal{F}$ is a keystone fix, and $m \in \mathcal{M}$ is the message to be signed. By picking a random number $\alpha \in_R \mathbb{Z}_q$, the algorithm outputs an ambiguous signature $\sigma = (c, s_1, s_2)$ as follows:
  - If $x = x_i$: $c = H_1(m, g^\alpha y_j^s \bmod p)$, $s_1 = (\alpha - c) \cdot x_i^{-1} \bmod q$, $s_2 = s$;
  - If $x = x_j$, $c = H_1(m, g^\alpha y_i^s \bmod p)$, $s_1 = s$, $s_2 = (\alpha - c) \cdot x_j^{-1} \bmod q$.
- AVERIFY. Given an ambiguous signature-message pair (σ, y_i, y_j, m), where $\sigma = (c, s_1, s_2)$, y_i and y_j are two public keys, the AVERIFY algorithm outputs accept or reject according to whether the following equality holds:

$$c \equiv H_1(m, g^c y_i^{s_1} y_j^{s_2} \bmod p). \tag{1}$$

- VERIFY. The algorithm takes input (k, S), where $k \in \mathcal{K}$ is the keystone and $S = (\sigma, y_i, y_j, m)$, and $\sigma = (c, s_1, s_2)$. The algorithm VERIFY outputs accept if $\mathsf{AVERIFY}(S)$=accept and the keystone k is *valid* by running a *keystone verification algorithm.* Otherwise, VERIFY outputs reject.

Note that the above are just the basic algorithms for generating and verifying concurrent signatures. In the following concrete concurrent signature protocol, it explicitly describes how to generate and verify keystones, and how to exchange concurrent signatures between two parties without the help of any TTP.

PCS1 Protocol: Before running the protocol, we assume that the SETUP algorithm is executed and the public keys y_A and y_B are published. Here we also assume that Alice is the initial signer and Bob is the matching signer and they want to exchange their signatures on messages m_A and m_B. Symmetrically, one can get the protocol description for the case where the roles of Alice and Bob are changed.

1. Alice sends Bob $(\sigma_A, \hat{t}, m_A)$, which are computed as follows.
   – Choose a random keystone $k \in_R \mathcal{K}$ and set $s_2 = H_1(k)$.
   – Run $\sigma_A = (c, s_1, s_2) \leftarrow \mathsf{ASIGN}(y_A, y_B, x_A, s_2, m_A)$.
   – Pick a random $t \in_R \mathbb{Z}_q$ and compute $\hat{t} = y_A^t \bmod p$.
2. Upon receiving $(\sigma_A, \hat{t}, m_A)$, where $\sigma_A = (c, s_1, s_2)$, Bob checks whether $\mathsf{AVERIFY}(\sigma_A,\ y_A, y_B, m_A) \equiv \mathsf{accept}$. If not, then Bob aborts. Otherwise, he sends (σ_B, m_B) to Alice by performing as follows.
   – Compute $r = \hat{t}^{x_B} \bmod p$, $r' = r \bmod q$, and set $s_1' = s_2 + r' \bmod q$.
   – Run $\sigma_B = (c', s_1', s_2') \leftarrow \mathsf{ASIGN}(y_A, y_B, x_B, s_1', m_B)$.
3. After (σ_B, m_B) is received, where $\sigma_B = (c', s_1', s_2')$, Alice performs as follows.
   – Check whether $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) = \mathsf{accept}$. If not, Alice aborts. Otherwise, continue.
   – Compute $r' = s_1' - s_2 \bmod q$, $r = y_B^{x_A t} \bmod p$, and check whether $r' \equiv r \bmod q$. If not, then Alice aborts. Otherwise, continue.
   – Issue the following signature proof Γ to show that r and $\hat{t}$ are properly generated by using the knowledge of his private key x_A (Refer to Section 2.2 in [27] for details):

$$\Gamma \leftarrow SPKEQ(\gamma : r = y_B^{t\gamma} \wedge \hat{t} = g^{t\gamma} \wedge y_A = g^{\gamma})(k). \tag{2}$$

   – Release the keystone $\kappa = \{k, r, t, \hat{t}, \Gamma\}$ publicly to bind both signatures σ_A and σ_B concurrently.
4. VERIFY Algorithm. After the keystone $\kappa = \{k, r, t, \hat{t}, \Gamma\}$ is released publicly, $\sigma_A = (c, s_1, s_2)$ and $\sigma_B = (c', s_1', s_2')$ are validated as Alice's and Bob's signature w.r.t. messages m_A and m_B respectively, iff all the following verifications hold.
   – Check whether $H_1(k) \equiv s_2$;
   – Check whether $r' = r \bmod q$, where $r' = s_1' - s_2 \bmod q$;
   – Check whether Γ is a valid signature proof;
   – Check $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$ and $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$.

In a summary, by using two pieces of keystone Susilo et al. strengthened the ambiguity of concurrent signatures so that there are four cases of authorship for two ambiguous signatures σ_A and σ_B. Due to this reason, their schemes achieves *perfect ambiguity.* As pointed in [27], in such schemes even an outsider knows (or believes) that Alice and Bob signed exactly one of two signatures σ_A and σ_B, he/she still cannot deduce whether Alice signed σ_A or σ_B. In Chen et al.'s scheme, however, this is very easy for an outsider. Based on the similarity of PCS1 and Chen et al.'s scheme, the unforgeability of PCS1 can be established in the random oracle model under the discrete logarithm assumption in subgroup $\langle g \rangle$, as stated in [27]. This reason is that one can incorporate the forking lemma [23] to provide a proof as did by Chen et al. in [11]. For the fairness, however, it is a different story.

3.2 On the Fairness

The authors of [27] argued the fairness of PCS1 protocol by the following two claims:

Claim 1. Before $\kappa = \{k, r, t, \hat{t}, \Gamma\}$ is released, both signatures σ_A and σ_B are ambiguous (Theorem 1 in [27]).

Claim 2. After $\kappa = \{k, r, t, \hat{t}, \Gamma\}$ is released, both signatures σ_A and σ_B are bound to the two signers concurrently (Theorem 2 in [27]).

Claim 1 is correct, but Claim 2 may be false if the initial signer Alice is dishonest. To illustrate this point, we now present a concrete attack against PCS1 protocol such that once κ is released, (σ_A, m_A) is not binding to Alice, but (σ_B, m_B) is indeed binding to Bob. Moreover, if necessary Alice can issue another signature-message $(\bar{\sigma}_A, \bar{m}_A)$ to binding herself, where message $\bar{m}_A$ is chosen at her will. From the view point of Bob, he is cheated by Alice, because what he expected is to exchange his signature on message m_B with Alice's signature on message m_A. But the result is that Alice indeed obtained his signature on message m_B, while Bob did not get Alice's signature on message m_A (though he may get Alice's signature on a different message $\bar{m}_A$). Naturally, this is *unfair* for the matching signer Bob. Because fairness implies that the matching signer Bob cannot be left in a position where a keystone binds his signature to him while the initial signer Alice's signature is not bound to Alice (See the last paragraph of page 296 in [11]). In the example of purchasing laptop given in Section 1, due to this attack Bob may be unable to get money from Alice, but Alice can pick up one laptop from Bob's shop.

The following is the basic idea of this attack. It is truly a natural and interesting method to construct perfect concurrent signatures by exploiting two keystones instead of one. However, we notice that in step 2 of PCS1 protocol no mechanism is provided for Bob to check the validity of the keystone fix s_2. Based on this observation, dishonest initial signer Alice can set $s_2 = H_1(k) + r' - \tilde{r}' \bmod q$, i.e., $s_2 + \tilde{r}' = H_1(k) + r' \bmod q$, where r' and $\tilde{r}'$ are some properly generated values. Then, Alice generates an ambiguous signature on m_A by using value s_2 (though she does not know the keystone for s_2). After receiving Bob's ambiguous signature on m_B, Alice can issue her signature on $\bar{m}_A$ by using the value $\bar{s}_2 = H_1(k)$ at her will. The detail follows.

Attack 1 on PCS1 Protocol. In this attack, we assume that the initial signer Alice is dishonest, but the matching signer Bob is honest, i.e., he follows each step of PCS1 protocol properly.

1. The dishonest initial signer Alice performs in the following way.
   1.1) Pick $t, \tilde{t} \in_R \mathbb{Z}_q$, and compute values $\hat{t}$, r, r', $\hat{t}'$, $\tilde{r}$, and $\tilde{r}'$ by

$$\begin{array}{lll} \hat{t} = y_A^t \bmod p, & r = y_B^{x_A t} \bmod p \ (= \hat{t}^{x_B} \bmod p), & r' = r \bmod q, \\ \hat{t}' = y_A^{\tilde{t}} \bmod p, & \tilde{r} = y_B^{x_A \tilde{t}} \bmod p \ (= \hat{t}'^{x_B} \bmod p), & \tilde{r}' = \tilde{r} \bmod q. \end{array} \tag{3}$$

   1.2) Choose a keystone $k \in_R \mathcal{K}$ and set $s_2 = H_1(k) + r' - \tilde{r}' \bmod q$. So Alice has the following equality

$$s_2 + \tilde{r}' = H_1(k) + r' \bmod q. \tag{4}$$

   1.3) Run $\sigma_A = (c, s_1, s_2) \leftarrow \mathsf{ASIGN}(y_A, y_B, x_A, s_2, m_A)$.
   1.4) Send $(\sigma_A, \hat{t}', m_A)$ to the matching signer Bob.
2. Since $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$, honest Bob sends Alice (σ_B, m_B) by
   2.1) Compute $\tilde{r} = \hat{t}'^{x_B} \bmod p$, and $\tilde{r}' = \tilde{r} \bmod q$;
   2.2) Set $s_1' = s_2 + \tilde{r}' \bmod q$;
   2.3) Run $\sigma_B = (c', s_1', s_2') \leftarrow \mathsf{ASIGN}(y_A, y_B, x_B, s_1', m_B)$.
3. Since (σ_B, m_B) is properly generated by honest Bob, it is easy to know that $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$ and that $\tilde{r}' \equiv s_1' - s_2 \bmod q$. That is, (σ_B, m_B) is Bob's valid signature. Now, Alice selects a message $\bar{m}_A$ at her choice and performs as follows.
   3.1) Set $\bar{s}_2 = H_1(k)$.
   3.2) Run $\bar{\sigma}_A = (\bar{c}, \bar{s}_1, \bar{s}_2) \leftarrow \mathsf{ASIGN}(y_A, y_B, x_A, \bar{s}_2, \bar{m}_A)$.
   3.3) Retrieve $(t, \hat{t}, r, r')$ from Step 1.1 (recall Eq. (3)).
   3.4) Issue a proof $\Gamma \leftarrow SPKEQ(\gamma : r = y_B^{t\gamma} \wedge \hat{t} = g^{t\gamma} \wedge y_A = g^{\gamma})(k)$.
   3.5) Output $(\bar{\sigma}_A, \bar{m}_A)$, (σ_B, m_B), and the keystone $\kappa = \{k, r, t, \hat{t}, \Gamma\}$.

On the validity of attack 1, we have the following proposition:

Proposition 1. *After the keystone information $\kappa = \{k, r, t, \hat{t}, \Gamma\}$ is released, the two signature-message pairs $(\bar{\sigma}_A, \bar{m}_A)$ and (σ_B, m_B) are bound to Alice and Bob, respectively. However, (σ_A, m_A) is not bound to Alice.*

Proof: This proof is almost self-evident, so we just mention the following main facts:

– $H_1(k) \equiv \bar{s}_2$ (recall Step 3.1).
– $r' \equiv s_1' - \bar{s}_2 \bmod q \equiv r \bmod q$ (recall Eqs. (3), (4)).
– Γ is a valid signature proof for $SPKEQ(\gamma : r = y_B^{t\gamma} \wedge \hat{t} = g^{t\gamma} \wedge y_A = g^{\gamma})(k)$, since it is properly generated by Alice in Step 3.4.
– $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$ and $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$, since both $\sigma_B = (c', s_1', s_2')$ and $\bar{\sigma}_A = (\bar{c}, \bar{s}_1, \bar{s}_2)$ are properly generated by running algorithm ASIGN in Steps 3.2 and 2.3, respectively.

Therefore, according to the specification of algorithm VERIFY reviewed in Section 2, $(\bar{\sigma}_A, \bar{m}_A)$ and (σ_B, m_B) are truly binding to Alice and Bob. However, the same keystone information $\kappa = \{k, r, t, \hat{t}, \Gamma\}$ cannot be used to bind (σ_A, m_A) to Alice. In fact, even Alice is unable to reveal a keystone k' such that $s_2 = H_1(k')$. Otherwise, this implies Alice can find a pre-image of hash value $s_2 = H_1(k) + r' - \tilde{r}' \bmod q$. □

3.3 On the Keystone Generation

In PCS1 protocol, a variant of Diffie-Hellman key exchange technique [14] is used to derive keystone fix r'. In summary, r' is generated as follows. By selecting a random number $t \in \mathbb{Z}_q$, Alice first sets $\hat{t} = y_A^t \bmod p$ and sends $\hat{t}$ to Bob. Then, Bob computes $r = \hat{t}^{x_B} \bmod p$, $r' = r \bmod q$, and sets $s_1' = s_2 + r' \bmod q$. Finally, Alice issues a signature proof $\Gamma \leftarrow SPKEQ(\gamma : r = y_B^{t\gamma} \wedge \hat{t} = g^{t\gamma} \wedge y_A = g^{\gamma})(k)$, and releases keystone information $\kappa = \{k, r, t, \hat{t}, \Gamma\}$. Hence, from the public information κ any third party can derive the value $y_{AB} \overset{\triangle}{=} g^{x_A x_B} \bmod p$ by calculating

$$y_{AB} = r^{t^*} \bmod p, \quad \text{where } t^* = t^{-1} \bmod q. \tag{5}$$

A problem is that the value of y_{AB} is the crux for some other cryptosystems, such as the strong designated verifier signature (SDVS) of Saeednia et al. [25], and the signcryption scheme of Huang and Cheng [19]. That is, if y_{AB} is available to an adversary those cryptosystems are broken (Check [18] for more discussions on SDVS). This implies that one user *cannot* use the same key pair to run PCS1 protocol and those cryptosystems, even though all of them work in the discrete logarithm setting with the same parameters. In other words, this is an example showing that the simultaneous use of related keys for two cryptosystems is insecure (See [17] for some positive results).

4 PCS2 and Its Fairness

This section briefly reviews and analyzes PCS2, which is a perfect concurrent signature constructed from bilinear pairing.

- SETUP: This algorithm first selects an *admissible bilinear pairing* (Sec 2.1 of [27]) $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic (additive and multiplicative, respectively) groups with the same prime order q. It also selects two cryptographic hash functions $H_0 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q$. Alice and Bob have private/public key pairs $(x_A, P_A = x_A P)$ and $(x_B, P_B = x_B P)$, where $x_A, x_B \in \mathbb{Z}_q^*$, and P is a generator of group $\mathbb{G}_1$. System parameters $\{\mathbb{G}_1, \mathbb{G}_2, e, q, P, H_0, H_1\}$ are publicly known.
- ASIGN: The ASIGN algorithm takes inputs (P_i, P_j, x, f, m), where x is the secret key associated with public keys P_i or P_j (i.e., $x = x_i$ or $x = x_j$), $f \in \mathcal{F}$ is a keystone fix, and $m \in \mathcal{M}$ is the message to be signed. By selecting a random number $a \in_R \mathbb{Z}_q$, the algorithm outputs an ambiguous signature $\sigma = (c, s_1, s_2)$ as follows:

  - If $x = x_i$: $c = H_1(P_i||P_j||e(aH_0(m), P) \cdot e(fH_0(m), P_j))$, $s_1 = (a - c)x_i^{-1} \bmod q$, $s_2 = f$;
  - If $x = x_j$: $c = H_1(P_i||P_j||e(aH_0(m), P) \cdot e(fH_0(m), P_i))$, $s_1 = f$, $s_2 = (a - c)x_j^{-1} \bmod q$.
- $\mathsf{AVERIFY}$: Given $\sigma = (c, s_1, s_2)$, $\mathsf{AVERIFY}(\sigma, P_i, P_j, m) = \mathsf{accept}$ iff the following equality holds.

$$c \equiv H_1(P_i||P_j||e(H_0(m), P)^c \cdot e(H_0(m), P_i)^{s_1} \cdot e(H_0(m), P_j)^{s_2}).$$

- VERIFY: Given a concurrent signature (k, S), where $k \in \mathcal{K}$ and $S = (\sigma = (c, s_1, s_2), P_i, P_j, m)$, $\mathsf{VERIFY}(k, S) = \mathsf{accept}$ iff k is a valid keystone by executing the keystone verification algorithm, and $\mathsf{AVERIFY}(S) = \mathsf{accept}$.

PCS2 Protocol. Without losing generality, we assume that the initial signer Alice and the matching signer Bob want to exchange their signatures on messages m_A and m_B, respectively.

1. Alice first sends (σ_A, Z) to Bob by performing as follows
   - Select a random keystone $k \in_R \mathcal{K}$ and set $s_2 = H_1(k)$;
   - Pick a randomness $\alpha \in \mathbb{Z}_q^*$ and compute $Z = \alpha P$;
   - Run $\sigma_A = (c, s_1, s_2) \leftarrow \mathsf{ASIGN}(P_A, P_B, x_A, s_2, m_A)$.
2. Upon receiving (σ_A, Z), Bob checks whether $\mathsf{AVERIFY}(\sigma_A, P_A, P_B, m_A) \equiv \mathsf{accept}$. If not, Bob aborts. Otherwise, he returns the following value σ_B to Alice.
   - Compute $r = e(P_A, Z)^{x_B}$, and set $s_1' = s_2 + r \bmod q$;
   - Run $\sigma_B = (c', s_1', s_2') \leftarrow \mathsf{ASIGN}(P_A, P_B, x_B, s_1', m_B)$.
3. Once $\sigma_B = (c', s_1', s_2')$ is received, Alice first computes $r = e(P_B, Z)^{x_A}$, then checks whether both $\mathsf{AVERIFY}(\sigma_B, P_A, P_B, m_B) \equiv \mathsf{accept}$ and $s_1' \equiv s_2 + r \bmod q$. If any of those two verifications fails, Alice aborts. Otherwise, she releases the keystone (k, α) so that both signatures σ_A and σ_B are binding concurrently. With (k, α), the validity of σ_A and σ_B is validated if all the following verifications hold:
   - $s_2 \equiv H_1(k)$ and $s_1' \equiv s_2 + r \bmod q$, where $r = e(P_A, P_B)^{\alpha}$;
   - $\mathsf{AVERIFY}(\sigma_A, P_A, P_B, m_A) \equiv \mathsf{accept}$;
   - $\mathsf{AVERIFY}(\sigma_B, P_A, P_B, m_B) \equiv \mathsf{accept}$.

Attack 2 on PCS2 Protocol. Compared with PCS1, PCS2 protocol is more efficient since the 2nd keystone fix r is exchanged between Alice and Bob in a more effective way (thanks to the bilinear pairing). However, PCS2 protocol is also *unfair* for the matching signer Bob, since a dishonest initial signer Alice can cheat Bob in an analogous way as in PCS1. More precisely, dishonest Alice can first select three random numbers $k, \alpha, \alpha' \in_R \mathbb{Z}_q^*$, and compute $Z = \alpha P$, $Z' = \alpha' P$, $r = e(P_A, P_B)^{\alpha}$, and $r' = e(P_A, P_B)^{\alpha'}$. Then, Alice further sets $s_2 = H_1(k) + r - r' \bmod q$, i.e., the following equality holds:

$$s_2 + r' \equiv H_1(k) + r \bmod q. \tag{6}$$

After that, Alice runs $\sigma_A \leftarrow \mathsf{ASIGN}(P_A, P_B, x_A, s_2, m_A)$, and sends (σ_A, Z') to Bob. Once getting Bob's valid signature $\sigma_B = (c', s_1', s_2')$ on message m_B, where

$s_1' = s_2 + r' \bmod q$ and $r' = e(P_A, Z')^{x_B}$, Alice releases (k, α) so that (σ_B, m_B) is bound to Bob. However, the same keystone information (k, α) does not bind (σ_A, m_A) to Alice. Moreover, if needed Alice can generate her signature $\bar{\sigma}_A$ on a different message $\bar{m}_A$ of her choice by using value $\bar{s}_2 = H_1(k)$. Due to Eq. (6), the keystone (k, α) shall bind $(\bar{\sigma}_A, \bar{m}_A)$ to Alice as well as (σ_B, m_B) to Bob.

5 The Improved Schemes

5.1 Description of the Improved Schemes

We observe that the attacks against the fairness of PCS1 and PCS2 result from the following fact: The initial signer Alice sets both two pieces of keystone alone. This privilege allows Alice to choose two pairs of keystone fixes so that the sums of them have the same value (recall Eqs. (4) and (6)). However, this sum determines the matching signer Bob's signature. Therefore, to avoid this attack we shall improve PCS1 and PCS2 as iPCS1 and iPCS2 by letting Bob choose the second keystone. At the same time, our improved protocols are designed to achieve a symmetry for both keystones. That is, both keystones can be values in the same domain and have the same verification algorithm. Moreover, the signature proof Γ is totaly removed in our iPCS1 to get a more efficient concurrent signature protocol (Check Table 1). The reason is that in the iPCS1 protocol (see below), the authenticity of $H_1(k')$ can be checked by Alice in Step 3 as follows: $s_1' \equiv s_2 + H_1(k') \bmod q$, where $k' = (\hat{t}^{x_A} \bmod p) \bmod q$ and $(\hat{t}, s_1')$ is received from Bob.

In the following description, we just specify the two improved concurrent signature protocols iPCS1 and iPCS2, while the corresponding algorithms are the same as in PCS1 and PCS2, respectively. In addition, note that iPCS1 protocol also works well for Chen et al.'s concurrent signature scheme [11].

iPCS1 Protocol: As in PCS1, we assume that the SETUP algorithm is already executed, and that the initial signer Alice and the matching signer Bob want to exchange their signatures on messages m_A and m_B, respectively.

1. Alice sends Bob (σ_A, m_A), where $\sigma_A = (c, s_1, s_2)$ is calculated as follows:
   – Choose a random keystone $k \in_R \mathcal{K}$ and set $s_2 = H_1(k)$;
   – Run $\sigma_A = (c, s_1, s_2) \leftarrow \mathsf{ASIGN}(y_A, y_B, x_A, s_2, m_A)$.
2. Upon receiving (σ_A, m_A), Bob checks whether $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$. If not, Bob aborts. Otherwise, Bob returns back $(\sigma_B, m_B, \hat{t})$ to Alice by
   – Pick a random $t \in_R \mathbb{Z}_q$ and compute $\hat{t} = y_B^t \bmod p$;
   – Compute $r = y_A^{x_B t} \bmod p$, and $k' = r \bmod q$;
   – Set $s_1' = s_2 + H_1(k') \bmod q$;
   – Run $\sigma_B = (c', s_1', s_2') \leftarrow \mathsf{ASIGN}(y_A, y_B, x_B, s_1', m_B)$.
3. Upon receiving $(\sigma_B, m_B, \hat{t})$, where $\sigma_B = (c', s_1', s_2')$, Alice performs as follows:
   – Compute $r = \hat{t}^{x_A} \bmod p$, and $k' = r \bmod q$;
   – Test whether $s_1' \equiv s_2 + H_1(k') \bmod q$;

- Check whether $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$;
- If σ_B is invalid, abort. Otherwise, release the keystone (k, k') publicly to bind both signatures σ_A and σ_B concurrently.

4. VERIFY Algorithm. With the keystone (k, k'), anybody can check the validity of $\sigma_A = (c, s_1, s_2)$ and $\sigma_B = (c', s'_1, s'_2)$ as follows.
   - Alice signs σ_A iff $s_2 \equiv H_1(k)$ and $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$.
   - Bob signs σ_B iff $s'_1 \equiv H_1(k) + H_1(k') \bmod q$ and $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$.

iPCS2 Protocol: Again, we assume that the initial signer Alice and the matching signer Bob want to exchange their signatures on messages m_A and m_B, respectively.

1. Alice first sends Bob (σ_A, m_A), where σ_A is computed as follows.
   - Select a random keystone $k \in_R \mathcal{K}$ and sets $s_2 = H_1(k)$;
   - Run $\sigma_A = (c, s_1, s_2) \leftarrow \mathsf{ASIGN}(P_A, P_B, x_A, s_2, m_A)$.
2. Upon receiving (σ_A, m_A), Bob checks that $\mathsf{AVERIFY}\ (\sigma_A, P_A, P_B, m_A) \equiv \mathsf{accept}$. If not, Bob aborts. Otherwise, he returns back (σ_B, Z) to Alice by performing below.
   - Pick a random $\alpha \in \mathbb{Z}_q^*$, compute $Z = \alpha P$ and $r = e(P_A, P_B)^{\alpha}$;
   - Set the second keystone $k' = r \bmod q$;
   - Compute $s'_1 = s_2 + H_1(k') \bmod q$;
   - Run $\sigma_B = (c', s'_1, s'_2) \leftarrow \mathsf{ASIGN}(P_A, P_B, x_B, s'_1, m_B)$.
3. Once $\sigma_B = (c', s'_1, s'_2)$ is received, Alice acts in the following way:
   - Compute $r = e(Z, P_B)^{x_A}$, and $k' = r \bmod q$.
   - Test whether $s'_1 \equiv s_2 + H_1(k') \bmod q$. If not, abort. Otherwise, continue.
   - Check whether $\mathsf{AVERIFY}(\sigma_B, P_A, P_B, m_B) \equiv \mathsf{accept}$.
   - If σ_B is invalid, Alice aborts. Otherwise, Alice releases the keystone (k, k') to bind both signatures σ_A and σ_B concurrently.
4. VERIFY Algorithm. With the keystone (k, k'), anybody can check the validity of $\sigma_A = (c, s_1, s_2)$ and $\sigma_B = (c', s'_1, s'_2)$ as follows.
   - Alice signs σ_A iff $s_2 \equiv H_1(k)$ and $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$.
   - Bob signs σ_B iff $s'_1 \equiv H_1(k) + H_1(k') \bmod q$ and $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$.

Table 1 gives the efficiency comparison for all concurrent signature protocols discussed in this paper. As the main computational overheads, we only consider multi-exponentiations (denote by E), scalar multiplications (denote by M), and bilinear mappings (denote by e). As in [5], we assume that simultaneous exponentiations are efficiently carried out by means of an exponent array. Namely, the costs for $a_1^{x_1} a_2^{x_2}$ and $a_1^{x_1} a_2^{x_2} a_3^{x_3}$ are only equivalent to 1.16 and 1.25 single exponentiation, respectively. Note that our iPCS1 outperforms the original PCS1 by about 50%, while iPCS2 is a little more efficient than PCS2.

Table 1. Efficiency Comparison

Protocol	Comp. Cost of Alice	Comp. Cost of Bob	Comp. Cost of Verifier	Signature Size	Keystone Size
CS [11]	2.41E	2.41E	2.5E	$3\|q\|$	$\|q\|$
CPS1 [27]	9.41E	3.41E	7.98E	$3\|q\|$	$4\|q\| + 2\|p\|$
iCPS1	3.41E	4.41E	2.5E	$3\|q\|$	$2\|q\|$
CPS2 [27]	6e+3.41E+1M	6e+3.41E	7e+3.5E	$3\|q\|$	$2\|q\|$
iCPS2	6e+3.41E	6e+3.41E+1M	6e+2.5E	$3\|q\|$	$2\|q\|$

5.2 Security Analysis of the Improved Schemes

Based on the results in [11,27] and the discussions previously provided, it is not difficult to see that both iPCS1 and iPCS2 are truly perfect concurrent signature protocols. Formally, we have the following theorem.

Theorem 1. *The above iPCS1 is a secure perfect concurrent signature protocol in the random oracle model under the discrete logarithm assumption. That is, iPCS1 is perfectly ambiguous, fair, and existentially unforgeable under a chosen message attack in the multi-party setting.*

Proof. First of all, *unforgeability* holds due to the facts that all basic algorithms in our iPCS are the same as in PCS1, and PCS1 is unforgeable (Theorem 4 in [27]). Second, *perfect ambiguity* is almost evident. The reason is that in the game of perfect ambiguity, to guess the signer of $\sigma_1 = (c, s_1, s_2)$ the adversary E has to distinguish whether either s_1 or s_2 is a random number from $\mathbb{Z}_q$ or an output of $H_1(\cdot)$ for some keystone k. However, this is impossible in the random oracle model, since the hash function $H_1(\cdot)$ is treated as a truly random function with the range of $\mathbb{Z}_q$. Similarly, E cannot determine the authorship of σ_2 better than guessing. Therefore, the adversary E cannot guess correctly the value $b \in \{1, 2, 3, 4\}$ selected by the challenger C at random with an advantage that is non-negligibly greater than 1/4. Now, we turn to prove *fairness* in detail.

We suppose that there exists an algorithm E that with non-negligible probability wins the fairness game, under the assumption that $H_1(\cdot)$ is a random oracle. Then, based on E's output we derive some contradictions.

To initialize the fairness game for a given security parameter ℓ, the challenger C runs the SETUP algorithm to generate the public parameters (p, q, g) as usual, where q is exponential in ℓ, choose all the private keys $x_i \in_R \mathbb{Z}_q$, set the public keys as $y_i = g^{x_i} \bmod p$, and give these public keys to the adversary E. Then, C responds to E's different kinds of queries as follows.

H_1-Queries: E can query the random oracle H_1 at any time. C simulates the random oracle by keeping a list of pairs (m_i, r_i), which is called the H_1-List. When an input $m \in \{0,1\}^*$ as H_1-query is received, C responds as follows:

1. If the query m is the first component for some pair (m_i, r_i) in the H_1-List, then C outputs r_i.

2. Otherwise, C selects a random number $r \in_R \mathbb{Z}_q$, outputs r as the value of $H_1(m)$, and adds the pair (m, r) to the H_1-List.

KGen Queries: E can request that the challenger C properly generate a keystone fix. To this end, C maintains a K-List of pairs (k_i, f_i), and answers such a query by choosing a random keystone $k \in_R \mathcal{K}$ and computing $f = H_1(k)$. C outputs f and adds the tuple (k, f) to the K-List. Note that K-List is a sublist of H_1-List, but is required to answer KReveal queries later.

KReveal Queries: E can request that the challenger C reveal the keystone k for any keystone fix f which is produced for answering a previous KGen Query. If there exists a pair (k, f) on the K-List, then C returns k, otherwise it outputs invalid.

ASign Queries: E can also make any signature query of the form (y_i, y_j, s, m), where $s \in \mathbb{Z}_q$, y_i and y_j $(y_i \neq y_i)$ are two public keys, and $m \in \{0,1\}^*$ is the message to be signed. To answer E's query, C computes the signature as normal and outputs $\sigma = (c, s_1, s_2) = \mathsf{ASIGN}(y_i, y_j, x_i, s, m)$.

Private Key Extract Queries: E can request the private key for any public key y_i. Since it is C that sets up all the private keys, C just returns the appropriate private key x_i as its answer.

In the final stage, E finally outputs $S = (\sigma, y_A, y_B, m)$ and a keystone $k \in \mathcal{K}$ with non-negligible probability η, where y_A and y_B are two public keys, $m \in \mathcal{M}$, and $\sigma = (c, s_1, s_2) \in \mathcal{S} \times \mathcal{F} \times \mathcal{F}$, such that $\mathsf{AVERIFY}(S) = \mathsf{accept}$ and either of the following two cases holds:

1. (k, S) is accepted by VERIFY so that $s_2 = \mathsf{KGEN}(k)$ is a previous output from a KGen query but no KReveal query on input s_2 was made.
2. E additionally produces $S' = (\sigma', y_A, y_B, m')$ along with another keystone $k' \in \mathcal{K}$, where $m' \in \mathcal{M}$ and $\sigma' = (c', s_1', s_2') \in \mathcal{S} \times \mathcal{F} \times \mathcal{F}$, such that $\mathsf{AVERIFY}(S') = \mathsf{accept}$ and $s_1' = s_2 + H_1(k')$. Furthermore, (k, k', S') is accepted by VERIFY, but (k, S) is not accepted by VERIFY.

Since the adversary E produces the above output with non-negligible probability η, either case 1 or case 2 must occur with non-negligible probability. We now analyze those two cases separately and then derive contradictions.

Case 1. Suppose E's outputs satisfy the conditions in case 1 with non-negligible probability. Namely, E has found a keystone k and an output of a KGen query s_2 such that $s_2 = H_1(k)$, but without making a KReveal query on input s_2. Since $H_1(\cdot)$ is a random oracle, E's probability of producing such a k is at most $\mu_1\mu_2/q$, where μ_1 is the number of H_1 queries made by E and μ_2 is the number of KGen queries made by E. Both μ_1 and μ_2 are polynomially bounded in the security parameter ℓ and q is exponential in ℓ, so this probability is negligible. This contradicts our assumption that case 1 occurs with non-negligible probability.

Case 2. Suppose case 2 occurs with non-negligible probability, i.e., the adversary E's outputs satisfy all conditions in case 2. Since $S' = (\sigma', y_A, y_B, m')$ is accepted by AVERIFY, $s_1' = s_2 + H_1(k')$, and (k, k', S') is accepted by VERIFY, we must have $s_2 = \mathsf{KGEN}(k) = H_1(k)$. At the same time, since $S = (\sigma, y_A, y_B, m)$ is also accepted by AVERIFY and we already have $s_2 = \mathsf{KGEN}(k) = H_1(k)$, this implies that (k, S) is also accepted by VERIFY. This contradicts to the condition in case 2 that (k, S) is not accepted by VERIFY. $\square$

Theorem 2. *The above iPCS2 is a secure perfect concurrent signature protocol in the random oracle model under bilinear Diffie-Hellman assumption. That is, iPCS2 is perfectly ambiguous, fair, and existentially unforgeable under a chosen message attack in the multi-party setting.*

Theorem 2 can be proved analogously as Theorem 1. That is, unforgeability and perfect ambiguity essentially follow from the results in [27] since all basic algorithms in iPCS2 are the same as in PCS2, while fairness can be obtained in a similar way as we do in Theorem 1.

6 Conclusion

For the applications with somewhat weak requirement of fairness, concurrent signatures [11] provide very simple and natural solutions for the traditional problem of fair exchange signatures without involving any trusted third party. To strengthen the ambiguity of concurrent signatures, two perfect concurrent signatures are proposed in [27]. This paper successfully identified an attack against those two perfect concurrent signatures by showing that they are actually *not* concurrent signatures. Consequently, those two schemes are *unfair* in fact. To avoid this attack, we presented effective improvements to achieve truly perfect concurrent signatures. Moreover, our improvement from Schnorr signature obtains about 50% performance enhancement over the original scheme in [27]. We also addressed another weakness in their keystone generation algorithm. In addition, we remarked that a similar attack can apply to an identity-based perfect concurrent signature scheme proposed in [12]. As the future work, it is interesting to consider how to improve the efficiency of perfect concurrent signatures, and how to construct concurrent signature schemes in multi-party setting.

References

1. M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n signatures from a variety of keys. In: *Asiacrypt '02*, LNCS 2501, pap. 415-432. Spriger-Verlag, 2002.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In: *Proc. 4th ACM Conf. on Comp. and Comm. Security*, pp. 8-17. ACM Press, 1997.
3. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In: *Eurocrypt '98*, LNCS 1403, pp. 591-606. Springer-Verlag, 1998.
4. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4): 591-606, 2000.
5. G. Ateniese. Efficient verifiable encryption (and fair exchange) of digital signature. In: *Proc. of AMC Conference on Computer and Communications Security* (*CCS'99*), pp. 138-146. ACM Press, 1999.
6. F. Bao, R.H. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line TTP. In: *Proc. of IEEE Symposium on Security and Privacy*, pp. 77-85, 1998.

7. M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Trans. on Inform. Theory*, 36(1): 40-46, 1990.
8. C. Boyd and E. Foo. Off-line fair payment protocols using convertible signatures. In: *Asiacrypt '98*, LNCS 1514, pp. 271-285. Springer-Verlag, 1998.
9. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In: *Crypto '03*, LNCS 2729, pp. 126-144. Springer-Verlag, 2003.
10. L. Chen. Efficient fair exchange with verifiable confirmation of signatures. In: *Asiacrypt '98*, LNCS 1514, pp. 286-299. Springer-Verlag, 1998.
11. L. Chen, C. Kudla, and K. G. Paterson. Concurrent signatures. In: *Eurocrypt '04*, LNCS 3027, pp. 287-305. Spriger-Verlag, 2004.
12. S.S.M. Chow and W. Susilo. Generic construction of (identity-based) perfect concurrent signatures. In: *Information and Communications Security ICICS '05*, LNCS 3783, pp. 194-206. Spriger-Verlag, 2005.
13. I. B. Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8(4): 201-222, 1995.
14. W. Diffie, and M.E. Hellman. New directions in cryptography. *IEEE Trans. on Inform. Theory*, 22: 644-654, 1976.
15. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6): 637-647, 1985.
16. S. Even and Y. Yacobi. Relations among public key signature schemes. *Technical Report 175*, Computer Science Dept., Technion, Israel, 1980.
17. S. Haber and B. Pinkas. Securely combining public-key cryptosystems. In: *Proc. of the 8th ACM Conf. on Computer and Communications Security*, pp. 215-224. ACM Press, 2001.
18. H. Lipmaa, G. Wang, and F. Bao. Designated verifier signature schemes: Attacks, new security notions and a new construction. In: *Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, LNCS 3580, pp. 459-471. Springer-Verlag, 2005.
19. H.-F. Huang and C.-C. Chang. An efficient convertible authenticated encryption scheme and its variant. In: *Information and Communications Security (ICICS'03)*, LNCS 2836, pages 382-392. Springer-Verlag, 2003.
20. J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In: *Crypto '99*, LNCS 1666, pp. 449-466. Sprnger-Verlage, 1999.
21. O. Goldreich. A simple protocol for signing contracts. In: *Crypto '83*, pp. 133-136. Plenum Press, 1984.
22. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM Journal of Computing*, 17(2): 281-308, 1988.
23. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3): 361-396, 2000.
24. R. L. Rivest, A. Shamir, and Y. Tauman. How to Leak a Secret. *Asiacrypt '01*, LNCS 2248, pp. 552-565. Spriger-Verlag, 2001.
25. S. Saeednia, S. Kremer, and O. Markowitch. An efficient strong designated verifier signature scheme. In: *Information Security and Cryptology - ICISC 2003*, LNCS 2971, pp. 40-54. Springer-Verlag, 2004.
26. C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3): 161-174, 1991.
27. W. Susilo, Y. Mu, and F. Zhang. Perfect concurrent signature schemes. In: *Information and Communications Security (ICICS '04)*, LNCS 3269, pp. 14-26. Spriger-Verlag, 2004.