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

First Draft: 7 March, 2006

Abstract. At Eurocrypt 2004, Chen, Kudla and Paterson introduced the concept of *concurrent signatures*, which allows two parties to produce two ambiguous signatures until an extra piece of information (called *keystone*) is released by the initial signer. Once the keystone is released publicly, both signatures are binding to their true signers *concurrently*. At ICICS 2004, Susilo, Mu and Zhang further proposed *perfect concurrent signatures* to strengthen the ambiguity of concurrent signatures. That is, even 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. However, this paper points out that Susilo et al.'s two perfect concurrent signatures 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, both of their two schemes are *unfair* for the matching signer. Moreover, we present a simple but effective way to avoid this attack such that the improved schemes are truly perfect concurrent signatures.

Keywords: Fair exchange, concurrent signature, security protocol.

1 Introduction

The concept of *concurrent signatures* was recently introduced bey Chen, Kudla and Paterson at 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* from the view point of any third 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 binding to their true signers *concurrently*, i.e., anybody can publicly verify 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 has a wide range of potential applications 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.,

[20, 14, 12]) 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 in PKIs. The point is that such an appropriate TTP may be costly or even unavailable to the parties involved.

In [11], Chen et al. ingeniously 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. Therefore, 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 a party needing the service of another, 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 binding 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 usual real life, however, if Alice does not want to buy a laptop (by releasing the keystone), why she wastes her time to book 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, 26] can be implemented very efficiently in both aspects of computation and communication, and do not need any help from a TTP. Therefore, the shortcomings in traditional solutions for fair exchange of signatures are overcome in a relatively simple and natural way.

At ICICS 2004, Susilo, Mu and Zhang [26] 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 parities indeed issued one of the two signatures, he/she still cannot deduce who signed which signature, different from Chen et al.'s scheme.

However, this paper shall point out that Susilo et al.'s perfect concurrent signatures are actually *not* concurrent signatures. Specifically, we successfully 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. To avoid this attack, we present a simple but effective way so that the improved schemes are truly perfect concurrent signatures. Moreover, our improvement from Schnorr signature obtains about 50% performance enhancement over their original scheme. In addition, we also address another weakness in their keystone generation algorithm.

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

2 Review of PSC1

We now review of PCS1 [26], which is a concurrent signature scheme derived from Schnorr signature [25]. Susilo et al. constructed PCS 1 by using some techniques from ring signatures [23, 1], as did by Chen et al.'s scheme in [11]. Basically, PCS1 consists of four algorithms: SETUP, ASIGN, AVERIFY and VERIFY, as described below.

- 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. It also selects a cryptographic hash function $H_1 : \{0,1\}^* \to \mathbb{Z}_q$. Then, the SETUP algorithm sets message space $\mathcal{M}$, keystone space $\mathcal{K}$, and keystone fix space $\mathcal{F}$ as follows: $\mathcal{M} = \mathcal{K} = \{0,1\}^*$, and $\mathcal{F} = \mathbb{Z}_q^*$. 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 outputs an ambiguous signature $\sigma = (c, s', s)$, given the input (y_i, y_j, x_i, s, m), where y_i and y_j are two public keys ($y_i \neq y_j$), x_i is the private key matching with y_i (i.e. $y_i = g^{x_i} \bmod p$), $s \in \mathcal{F}$, and $m \in \mathcal{M}$ is the message to be signed. The algorithm is carried out as follows:
  1. Select a random number $\alpha \in \mathbb{Z}_q$.
  2. Evaluate $c = H_1(m, g^{\alpha} y_j^s \bmod p)$.
  3. Compute $s' = (\alpha - c) \cdot x_i^{-1} \bmod q$.
  4. Output anonymous signature $\sigma = (c, s', s)$.
- AVERIFY. Given an anonymous signature-message pair (σ, y_i, y_j, m), where $\sigma = (c, s', s)$, y_i and y_j are valid public keys, the AVERIFY algorithm outputs accept, if

the following equality holds:

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

Otherwise, it outputs reject.

– VERIFY. The algorithm accepts input (k, S), where $k \in \mathcal{K}$ is the keystone and $S = (\sigma, y_i, y_j, m)$, and $\sigma = (c, s', s)$. The algorithm VERIFY outputs accept if 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 a 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. Symmetrically, one can get the protocol description for the case where the roles of Alice and Bob are changed.

1. The initial Alice performs as follows.
   – Select a message $m_A \in \mathcal{M}$.
   – Choose a random keystone $k \in \mathcal{K}$ and set $s_2 = H_1(k)$.
   – Run $\sigma_A \leftarrow$ ASIGN$(y_A, y_B, x_A, s_2, m_A)$. Let $\sigma_A = (c, s_1, s_2)$.
   – Pick a random $t \in \mathbb{Z}_q$ and compute $\hat{t} = y_A^t \bmod p$.
   – Send $(\sigma_A, \hat{t}, m_A)$ to the matching signer Bob.
2. Upon receiving Alice's ambiguous signature-message pair $(\sigma_A, \hat{t}, m_A)$, Bob validates it by checking whether the output of AVERIFY$(\sigma_A, y_A, y_B, m_A)$ is accept. If not, then Bob just aborts. Otherwise, Bob acts in the following way.
   – Select a message $m_B \in \mathcal{M}$.
   – Compute $r = \hat{t}^{x_B} \bmod p$, and $r' = r \bmod q$.
   – Set $s_1' = s_2 + r' \bmod q$.
   – Run $\sigma_B \leftarrow$ ASIGN$(y_B, y_A, x_B, s_1', m_B)$. Let $\sigma_B = (c', s_1', s_2')$.
   – Send (σ_B, m_B) to Alice.
3. After (σ_B, m_B) is received, Alice parses σ_B into (c', s_1', s_2'), and performs as follows.
   – Check whether AVERIFY$(\sigma_B, y_A, y_B, m_B)$ = accept. If not, Alice aborts. Otherwise, continue.
   – Compute $r' = s_1' - s_2 \bmod q$.
   – Compute $r = y_B^{x_A t} \bmod p$, and check whether $r' = r \bmod q$. If not, then Alice aborts. Otherwise, continue.
   – Issue the following signature proof Γ by using the private key x_A [26]:

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

   – 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, both signature-message pairs (σ_A, m_A) and (σ_B, m_B) are binding concurrently. Specifically, any verifier can conclude that σ_A and σ_B are respectively signed by Alice and Bob if all of the following verifications hold.
   - Parse σ_A into (c, s_1, s_2) and σ_B into (c', s_1', s_2').
   - Check whether $H_1(k) \equiv s_2$.
   - Compute $r' = s_1' - s_2 \bmod q$, and check whether $r' = r \bmod q$.
   - Check whether Γ is a valid signature proof.
   - Check the validity of σ_A and σ_B, i.e., if $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$ and $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$.
   - If all above verifications hold, output accept. Otherwise, output reject.

3 The Security of PCS1

Before analyzing the security of PCS1, we first briefly review the security definition of concurrent signatures and give some explanations on PCS1. The security of concurrent signatures is defined via three notions [11]: unforgeability, ambiguity, and fairness. *Unforgeability* requires any efficient adversary without the corresponding two secret keys cannot forge a valid concurrent signature with non-negligible probability under chosen message attacks [21]. *Ambiguity* means that given a concurrent signature without the keystone, any adversary cannot distinguish who of the two signers issued this signature. *Fairness* intuitively requires that (a) only the initial signer can reveal the keystone and (b) once the keystone is released, both signatures are binding to the two signers concurrently. Readers could refer to Section 3 in [11] for those standard formal definitions. Perfect concurrent signatures just strengthen the ambiguity by requiring any adversary is still unable to identify the true signers even he/she knows that each of the two signers indeed signed one of the two signatures. Formal definition is given by Definition 5 in [26].

Now, we compare PCS1 with Chen et al.'s original concurrent signature proposed in [11]. Actually, there are no essential differences in the underlying algorithms of those two schemes, because both of them are some variants of Schnorr-based ring signatures proposed by Abe et al. [1]. However, the authors of [11] and [26] have different ideas on the problem how to exchange and fix concurrent signatures between two parties. Simply speaking, Chen et al. [11] just used one keystone, but Susilo et al. exploited two keystones. This is the exact reason why *perfect* concurrent signatures can be implemented in [26].

In more detail, if we use the same notations as in PCS1 reviewed in the previous section, Chen et al. realized their concurrent signature scheme by setting $s_1' = s_2 = H_1(k)$, but Susilo et al. implemented their perfect concurrent signature scheme by setting $s_1' = r' + s_2 = r' + H_1(k) \bmod q$, where value r' is fixed by another keystone r via $r' = r \bmod q$ and $r = (y_B^{x_A t} \bmod p) \bmod q$. Therefore, given two valid ambiguous signatures $\sigma_A = (c, s_1, s_2)$ and $\sigma_B = (c, s_1', s_2')$ without any knowledge of keystones, an outsider can get different conclusions on the authorship of those two signatures. Specifically, in Chen et al.'s scheme the following three cases occur with the same probability 1/3:

i) Alice generates both $\sigma_A = (c, s_1, s_2)$ and $\sigma_B = (c', s'_1, s'_2)$ by first running $\mathsf{ASIGN}(y_A, y_B, x_A, s'_2, m_B)$ and then $\mathsf{ASIGN}(y_A, y_B, x_A, s_2 = s'_1, m_A)$.
ii) Bob generates both $\sigma_A = (c, s_1, s_2)$ and $\sigma_B = (c', s'_1, s'_2)$ by first running $\mathsf{ASIGN}(y_B, y_A, x_B, s_1, m_A)$ and then $\mathsf{ASIGN}(y_B, y_A, x_B, s'_1 = s_2, m_B)$.
iii) Alice first generates $\sigma_A = (c, s_1, s_2)$ by $\mathsf{ASIGN}(y_A, y_B, x_A, s_2, m_A)$, and then Bob generates $\sigma_B = (c', s'_1, s'_2)$ by running $\mathsf{ASIGN}(y_B, y_A, x_B, s'_1 = s_2, m_B)$.

However, in the Susilo et al.'s scheme it is also possible that Bob generates σ_A and Alice generates σ_B, since before the keystone κ is released an outsider cannot distinguish whether the value $r' = s'_1 - s_2 \bmod q$ is predetermined or not. Hence, the following case iv) will also appear equally with the above three cases (without the restriction $s'_1 = s_2$):

iv) Bob generates $\sigma_A = (c, s_1, s_2)$ by $\mathsf{ASIGN}(y_B, y_A, x_B, s_1, m_A)$, and Alice generates $\sigma_B = (c', s'_1, s'_2)$ by running $\mathsf{ASIGN}(y_A, y_B, x_A, s'_2, m_B)$.

In summary, Susilo et al. strengthened the ambiguity of concurrent signatures so that all four possible cases of authorship appear with the same probability 1/4. Due to this reason, their schemes are called *perfect concurrent signatures*. As pointed in [26], 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. scheme, however, this is very easy for an outsider.

Due to the similarity of PCS1 and Chen et al.'s scheme, the authors of [26] stated that the unforgeability of PCS1 can be established in the random oracle model, under the discrete logarithm assumption in subgroup $\langle g \rangle$. This is really reasonable, since one can incorporate the forking lemma [22] to provide a proof as did by Chen et al. in [11].

For the fairness, however, it is a different story.

3.1 On the Fairness

The authors of [26] 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 [26]).
Claim 2. After $\kappa = \{k, r, t, \hat{t}, \Gamma\}$ is released, both signatures σ_A and σ_B are binding to the two signers concurrently (Theorem 2 in [26]).

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. In 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 Bob'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 also bound to Alice (See the last paragraph of page 296 in [11]). In the example of purchasing laptop given in Introduction, 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 any 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 details are given below.

Attack 1 against 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.
   1.1) Pick two random numbers $t, \tilde{t} \in \mathbb{Z}_q$, and then 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 random keystone $k \in \mathcal{K}$ and set $s_2 = H_1(k) + r' - \tilde{r}' \bmod q$. That is, we have
   $$s_2 + \tilde{r}' = H_1(k) + r' \bmod q. \tag{4}$$
   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. It is easy to know $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$. So, honest Bob acts as follows.
   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 $\sigma_B = (c', s_1', s_2') \leftarrow \mathsf{ASIGN}(y_B, y_A, x_B, s_1', m_B)$.
   2.4) Send (σ_B, m_B) to Alice.
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 binding to Alice and Bob, respectively. However, (σ_A, m_A) is not binding 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 $s_2 = H_1(k) + r' - \tilde{r}' \bmod q$. □

3.2 On the Keystone Generation

In PCS1 protocol, a variant of Diffie-Hellman key exchange technique [13] 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 $\hat{h}$ to Bob. Then, Bob computes $r = \hat{h}^{x_B} \bmod p$, $r' = r \bmod q$, 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 $y_{AB} \stackrel{\triangle}{=} g^{x_A x_B} \bmod p$ by calculating

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

This point is that the value of y_{AB} is the crux for some other cryptosystems, such as strong designated verifier signature (SDVS) of Saeednia et al. [24], and signcryption scheme of Huang and Cheng [18]. That is, if y_{AB} is available to an adversary those cryptosystems are broken (See Helger et al.'s discussion [17] on SDVS). This implies that one user *cannot* use the same key pair to run PCS1 protocol and those cryptosystems, even though all of those cryptographic primitives work in the discrete logarithm setting and have the same parameters. In other words, this is an example showing that the simultaneous use of related keys for two cryptosystems is insecure (See [16] 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: The SETUP algorithm selects an *admissible bilinear pairing* (Sec 2.1 of [26]) $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, 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 $H_0 : \{0,1\}^* \to \mathbb{G}_1$ and $H_1 : \{0,1\}^* \to \mathbb{Z}_q$. Alice and Bob have private/public key pairs $(s_A, P_{pubA} = s_A P)$ and $(s_B, P_{pubB} = s_B P)$, where $s_A, s_B \in \mathbb{Z}_q^*$, 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 accepts input $(P_{pub1}, P_{pub2}, s_1, \hat{s}, m)$, where s_1 is the secret key associated with the public key P_{pub1}, $\hat{s} \in \mathcal{F}$, and $m \in \mathcal{M}$ is the message to be signed. The algorithm outputs an ambiguous signature $\sigma = (c_0, c_1, c_2)$ as follows:
  - Select a random $\hat{\alpha} \in \mathbb{Z}_q$.
  - Compute $c_0 = H_1(P_{pub1}||P_{pub2}||e(\hat{\alpha}H_0(m), P)e(\hat{s}H_0(m), P_{pub2}))$.
  - Let $c_1 = (\hat{\alpha} - c_0)s_1^{-1} \bmod q$, and $c_2 = \hat{s}$.
- AVERIFY: Given $\sigma = (c_0, c_1, c_2)$, $\mathsf{AVERIFY}(\sigma, P_{pub1}, P_{pub2}, m) = \mathsf{accept}$ iff the following equality holds.

$$c_0 \equiv H_1(P_{pub1}||P_{pub2}||e(H_0(m), P)^{c_0} e(H_0(m), P_{pub1})^{c_1} e(H_0(m), P_{pub2})^{c_2}).$$

- VERIFY: Given a concurrent signature (k, S), where $k \in \mathcal{K}$ and $S = (\sigma = (c_0, c_1, c_2), P_{pub1}, P_{pub2}, m)$, $\mathsf{VERIFY}(k, S) = \mathsf{accept}$ 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 performs the following.
   - Select a random keystone $k \in \mathcal{K}$ and sets $c_2 = H_1(k)$.
   - Pick a random $\alpha \in \mathbb{Z}_q^*$ and compute $Z = \alpha P$.
   - Runs $\sigma_A = (c_0, c_1, c_2) \leftarrow \mathsf{ASIGN}(P_{pubA}, P_{pubB}, s_A, c_2, m_A)$.
   - Send (σ_A, Z) to Bob.
2. Upon receiving (σ_A, Z), Bob checks whether $\mathsf{AVERIFY}(\sigma_A, P_{pubA}, P_{pubB}, m_A) \equiv \mathsf{accept}$. If not, Bob aborts. Otherwise, he performs the following.
   - Compute $r = e(P_{pubA}, Z)^{s_B}$.
   - Set $c_1' = c_2 + r \bmod q$.
   - Run $\sigma_B = (c_0', c_1', c_2') \leftarrow \mathsf{ASIGN}(P_{pubB}, P_{pubA}, s_B, c_2', m_B)$.
   - Send σ_B to Alice.
3. Once $\sigma_B = (c_0', c_1', c_2')$ is received, Alice first computes $r = e(Z, P_{pubB})^{s_A}$, and then checks whether both $\mathsf{AVERIFY}(\sigma_B, P_{pubA}, P_{pubB}, m_B) \equiv \mathsf{accept}$ and $c_1' \equiv c_2 + r \bmod q$. If any of the two verifications fails, then Alice aborts the protocol. Otherwise, Alice 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 can be validated by any verifier if all the following verifications hold:

- $c_1' \equiv c_2 + r \bmod q$, where $r = e(P_{pubA}, P_{pubB})^{\alpha}$.
- $\mathsf{AVERIFY}(\sigma_A, P_{pubA}, P_{pubB}, m_A) \equiv \mathsf{accept}$.
- $\mathsf{AVERIFY}(\sigma_B, P_{pubA}, P_{pubB}, m_B) \equiv \mathsf{accept}$.

Attack 2 against on PCS2 Protocol. Compared with PCS1, PCS2 protocol is more efficient since the 2nd keystone fix r is exchanged between Alice and Bob more efficiently (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 first chooses three random numbers $k, \alpha, \alpha' \in \mathbb{Z}_q^*$, then computes $Z = \alpha P$ and $Z' = \alpha' P$. Then, Alice computes $r = e(P_{pubA}, P_{pubB})^{\alpha}$, $r' = e(P_{pubA}, P_{pubB})^{\alpha'}$, and sets $c_2 = H_1(k) + r - r' \bmod q$. That is, we have the following equality:

$$c_2 + r' \equiv H_1(k) + r \bmod q. \tag{6}$$

After that, Alice generates σ_A on message m_A by using value c_2, and sends (σ_A, Z') to Bob. Once getting Bob's valid signature $\sigma_B = (c_0', c_1', c_2')$ on message m_B, where $c_1' = c_2 + r' \bmod q$ and $r' = e(P_{pubA}, Z')^{s_B}$, Alice releases (k, α) so that (σ_B, m_B) is binding 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 $\bar{c}_2 = H_1(k)$. Due to Eq. (6), it is easy to know that the keystone (k, α) shall bind $(\bar{\sigma}_A, \bar{m}_A)$ to Alice.

5 Improvements

We observe that the attack against the fairness of PCS1 and PCS2 results from the following fact: The initial singer Alice sets both two pieces of keystone alone. Therefore, Alice can 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 improve PCS1 and PCS2 as iPCS1 and iPCS2 by allowing Bob to 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 iPCS1, 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 our 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 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, an that the initial signer Alice and the matching signer Bob want to exchange their signatures on messages m_A and m_B, respectively.

1. The initial Alice performs as follows.
   - Choose a random keystone $k \in \mathcal{K}$ and set $s_2 = H_1(k)$.
   - Run $\sigma_A \leftarrow \mathsf{ASIGN}(y_A, y_B, x_A, s_2, m_A)$. Let $\sigma_A = (c, s_1, s_2)$.
   - Send (σ_A, m_A) to the matching signer Bob.
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, then Bob just aborts. Otherwise, Bob acts in the following way.
   - Pick a random $t \in \mathbb{Z}_q$ 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 $\sigma_B = (c', s_1', s_2') \leftarrow \mathsf{ASIGN}(y_B, y_A, x_B, s_1', m_B)$.
   - Send $(\sigma_B, m_B, \hat{t})$ to Alice.
3. After $(\sigma_B, m_B, \hat{t})$ is received, Alice parses σ_B into (c', s_1', s_2'), and 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. Once the keystone (k, k') is available, any verifier can verify that σ_A and σ_B are respectively signed by Alice and Bob if all of the following equalities hold.
   - $s_2 \equiv H_1(k)$ and $s_1' \equiv s_2 + H_1(k') \bmod q$.
   - $\mathsf{AVERIFY}(\sigma_A, y_A, y_B, m_A) \equiv \mathsf{accept}$.
   - $\mathsf{AVERIFY}(\sigma_B, y_A, y_B, m_B) \equiv \mathsf{accept}$.

iPCS2 Protocol: Again, we just 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 performs the following.
   - Select a random keystone $k \in \mathcal{K}$ and sets $c_2 = H_1(k)$.
   - Runs $\sigma_A = (c_0, c_1, c_2) \leftarrow \mathsf{ASIGN}(P_{pubA}, P_{pubB}, s_A, c_2, m_A)$.
   - Send (σ_A, m_A) to Bob.
2. Upon receiving (σ_A, m_A), Bob checks the validity of σ_A by testing whether $\mathsf{AVERIFY}$ $(\sigma_A, P_{pubA}, P_{pubB}, m_A) \equiv \mathsf{accept}$. If not, Bob aborts the protocol. Otherwise, he performs the following.
   - Pick a random $\alpha \in \mathbb{Z}_q^*$, compute $Z = \alpha P$ and $r = e(P_{pubA}, P_{pubA})^\alpha$.
   - Set the second keystone k' by $k' = r \bmod q$.
   - Compute $c_1' = c_2 + H_1(k') \bmod q$.
   - Run $\sigma_B = (c_0', c_1', c_2') \leftarrow \mathsf{ASIGN}(P_{pubB}, P_{pubA}, s_B, c_2', m_B)$.
   - Send (σ_B, Z) to Alice.
3. Once $\sigma_B = (c_0', c_1', c_2')$ is received, Alice performs as follows:
   - Compute $r = e(Z, P_{pubB})^{s_A}$, and $k' = r \bmod q$.
   - Test whether $c_1' \equiv c_2 + H_1(k') \bmod q$. If not, abort. Otherwise, continue.
   - Check whether $\mathsf{AVERIFY}(\sigma_B, P_{pubA}, P_{pubB}, m_B) \equiv \mathsf{accept}$.

- If σ_B is invalid, then Alice aborts the protocol. Otherwise, Alice releases the keystone (k, k') to bind both signatures σ_A and σ_B concurrently.

4. With the keystone (k, k'), the validity of σ_A and σ_B can be validated by any verifier if all the following verifications hold:
   - $c_2 \equiv H_1(k)$ and $c_1' \equiv c_2 + H_1(k') \bmod q$.
   - $\mathsf{AVERIFY}(\sigma_A, P_{pubA}, P_{pubB}, m_A) \equiv \mathsf{accept}$.
   - $\mathsf{AVERIFY}(\sigma_B, P_{pubA}, P_{pubB}, m_B) \equiv \mathsf{accept}$.

Based on the results in [11, 26] 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 proposition.

Proposition 2. *According to the formal definitions given in [11, 26], the above iPCS1 and iPCS2 are secure perfect concurrent signature protocols, under the discrete logarithm assumption and bilinear Diffie-Hellman assumption, respectively. That is, both iPCS1 and iPCS2 are ambiguous, fair, and existentially unforgeable under a chosen message attack in the multi-party setting.*

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. As a result, note that our iPCS1 is much more efficient since it improves the performance of original PCS1 by about 50%.

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 [26]	9.41E	3.41E	7.98E	$3\|q\|$	$4\|q\| + 2\|p\|$
iCPS1	3.41E	4.41E	2.5E	$3\|q\|$	$2\|q\|$
CPS2 [26]	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	$3\|q\|$	$2\|q\|$

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 any help from a trusted third party. To strengthen the ambiguity of concurrent signatures, two perfect concurrent signatures are proposed in [26]. This paper successfully identified an attack against those two perfect concurrent signatures showing that both of those two schemes 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 [26]. In addition, we also addressed another weakness in their keystone generation algorithm. As the future work, it is interesting to consider how to construct more efficient perfect concurrent signatures.

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. I. B. Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8(4): 201-222, 1995.
13. W. Diffie, and M.E. Hellman. New directions in cryptography. *IEEE Trans. on Inform. Theory*, 22: 644-654, 1976.
14. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6): 637-647, 1985.
15. S. Even and Y. Yacobi. Relations among public key signature schemes. *Technical Report 175*, Computer Science Dept., Technion, Israel, 1980.
16. 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.
17. 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.
18. 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.
19. J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In: *Crypto '99*, LNCS 1666, pp. 449-466. Sprnger-Verlage, 1999.
20. O. Goldreich. A simple protocol for signing contracts. In: *Crypto '83*, pp. 133-136. Plenum Press, 1984.
21. 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.
22. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3): 361-396, 2000.

23. R. L. Rivest, A. Shamir, and Y. Tauman. How to Leak a Secret. *Asiacrypt '01*, LNCS 2248, pp. 552-565. Spriger-Verlag, 2001.
24. 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.
25. C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3): 161-174, 1991.
26. 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.