Accountability of Perfect Concurrent Signature

Li Yunfeng, He Dake, Lu Xianhui

Lab. of Info. Security & National Computing Grid
Southwest Jiaotong University
Chengdu 610031, Sichuan, China
forwardlyf@gmail.com

Abstract. Concurrent signature provided a novel idea for fair exchange protocol without trusted third party. Perfect Concurrent Signature is proposed to strengthen the ambiguity of the concurrent signature. Wang et al, pointed out there exist an attack against the fairness of Perfect Concurrent Signature and proposed the improved perfect concurrent signature. This paper find that in proposed (perfect) concurrent signature protocol, no matter two party or multi-party, the signer could bind multiple messages with one keystone set but let the other signers know only one of the messages. This is a new unfair case in the application of concurrent signature. Based on this observation, we propose that accountability should be one of the security properties of (perfect) concurrent signature and we give the definition of accountability of concurrent signature. To illustrate this idea, we give an attack scene against the accountability of improved perfect concurrent signature proposed by Wang et al, and propose an update version of perfect concurrent signature to avoid such attack.

Keywords: fair exchange, concurrent signature, accountability

1. Introduction

Fair exchange is a well studied problem. Especially after the optimized fair exchange concept was firstly proposed by Asokan, etc [1], many schemes [2] are proposed to implement fair exchange with off-line Trusted Third Party. In [3], Chen et al. ingeniously observed that the full power of fair exchange is not necessary in many applications, since there are some mechanisms that provide a more natural dispute resolution than the reliance on a TTP. Chen et al, proposed the concept of concurrent signature, CS for short, at Eurocrypt2004. Such signature scheme allows two parties, without any third party, to produce and exchange two ambiguous signatures which are ambiguous for any third party until an extra piece of information (called keystone) is released by one of the parties.

In order to strength the ambiguity of the signature before keystone is released, Susilo, Mu and Zhang [4] further proposed a strong notion called perfect concurrent signatures, PCS for short. Unfortunately, In [5] Wang et al, observed that, in PCS, the initiator generate the two keystones independently which enable the initiator could bind different ambiguous signature, but not the one send to the matching singer, with the one generated by the matching signer. To resolve this problem, Wang et al,

proposed improved perfect concurrent signature, called iPCS for short. The difference is that in iPCS the initial signer and matching signer generate one piece of keystone respectively. But in PCS only the initiator generates the two piece of keystone alone.

Later, in [6], Chow et al, improved their generic construction of (Identity-based) perfect concurrent signature with the idea from [5]. Asymmetric concurrent signature [7], tripartite concurrent signature [8] and mutli-party concurrent signature[11,12] are proposed. Recently, Feng et al, apply the concept of concurrent signature to establish the new concept of fair authenticated key exchange protocol in [11].

However, we found proposals given in [3,4,5,6,7,8,9,10] are vulnerable to follow attack scene: Any party of one (perfect) concurrent signature could generate multiple ambiguous signatures, for different messages, which could be bound with the same keystone at the same time. These messages maybe different from the one included in the ambiguous signature sent to the matching (initial) singer. Based on this observation, we propose that accountability [12] should be one of the security properties of (perfect) concurrent signature. Accountability means any third party could be convinced that the ambiguous signature, which can pass the verification of the verify algorithm, is the only one generated by the signer based on the released keystone.

In Section 2, we briefly overview the protocol structure and algorithms used in PCS and iPCS, and discuss the security properties of concurrent signature. Then we descript the attack against accountability of iPCS1 in section 3. Proposal to avoid this kind of attack is given in Section 4. Section 5 is the conclusion of this paper.

2. Overview of PCS/iPCS and security properties of CS

As mentioned by [5], the PCS and iPCS use the same protocol structure and algorithms, so, in this section, we firstly review the concept of PCS[4] and secondly review the iPCS1 protocol [5]. At last, we descript the new security property, accountability of (Perfect) Concurrent Signature in section 2.3.

2.1 Overview of Perfect Concurrent Signature

Basically, PCS 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 $g \in Z_p$ of order q. It also selects a cryptographic hash function $H_1 : (0,1)^* \to Z_q$. Then, the SETUP algorithm sets message space M, keystone space K, and keystone fix space F as follows: $M=K=\{0,1\}^*$ and $F=Z_q^*$. 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_A, y_B, x_A, s, m), where y_A and y_B are two public keys $y_A \neq y_B$, x_A, is the

private key matching with y_A (i.e. $y_A = g^{x_A} \bmod p$), $s = H_1(k), k \in_R Z_q, s \in F$, k is the keystone and $m \in M$ is the message to be signed. The algorithm is carried out as follows:

1. Select a random number $\alpha \in Z_q$.
2. Evaluate $c = H_1(m, g^{\alpha} y_B^s \bmod p)$.
3. Compute $s' = (\alpha - c) \cdot x_A^{-1} \bmod q$.
4. Output anonymous signature $\sigma = (c, s', s)$.

– AVERIFY. Given an ambiguous signature-message pair (σ, y_A, y_B, m), the AVERIFY algorithm outputs accept, if $c = H_1(m, g^c y_A^{s'} y_B^s \bmod p)$ holds, Otherwise, it outputs reject.

– VERIFY. The algorithm accepts input (k, S), where $k \in K$ is the keystone and $S = (\sigma, y_A, y_B, m)$. 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 perfect concurrent signatures. In the following concrete iPCS1 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.

In PCS, the initiator chooses the two pieces of keystone for herself and Bob independently, which result in the attack scene described in section3.1 of [5]. Then Wang, et al, argued to let the responder choose the second piece of keystone and proposed the iPCS1 and iPCS2 protocols. Follow section 2.2 will give the specified iPCS1 protocol.

2.2 Overview of iPCS1

As in [3,4,5], iPCS1 assumes 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, m_B respectively.

1. The initial Alice performs as follow:
   – Choose a random keystone $k \in K$ and set $s_1 = H_1(k)$.
   – Run $\sigma_A \leftarrow ASIGN(y_A, y_B, x_A, s_1, m)$. Let $\sigma_A = (c, s_2, s_1)$.
   – Send (σ_A, m_A) to the matching signer Bob.
2. Upon receiving (σ_A, m_A), Bob checks $AVERIFY(\sigma_A, y_A, y_B, m_A) = accept$ .If not, then Bob just aborts. Otherwise, Bob acts in the following way.
   – Pick a random $t \in Z_q$ and compute $\hat{t} = y_B^t \bmod p$
   – Compute $r = (y_A^{x_B})^t \bmod p$, and $k' = r \bmod q$.
   – Set $s_1' = s_1 + H_1(k') \bmod q$.
   – Run $\sigma_B = (c', s_2', s_1') \leftarrow ASIGN(y_B, y_A, x_B, s_1', m_B)$.
   – Send $(\sigma_B, m_B, \hat{t})$ to Alice.
3. After $(\sigma_B, m_B, \hat{t})$ is received, Alice performs as follows.

- Compute $r = \hat{t}^{x_A} \bmod p$, and $k'=r \bmod q$..
- Test whether $s_1' = s_1 + H_1(k') \bmod q$.
- Check whether $AVERIFY(\sigma_B, y_A, y_B, m_B) = accept$
- 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.
   - $s_1 = H_1(k)$ and $s_1' = s_1 + H_1(k') \bmod q$.
   - $AVERIFY(\sigma_A, y_A, y_B, m_A) = accept$
   - $AVERIFY(\sigma_B, y_A, y_B, m_B) = accept$

2.3 The accountability of CS/PCS

The original security model of concurrent signature in [3], as well as those defined in [4,5,8,10], requires a (perfect) concurrent signature to satisfy *correctness, ambiguity, unforgeability* and *fairness*. The correctness makes sure the signature will work out the desired result. Ambiguity stress that any third party can not identify who is the exactly signer of one ambiguous signature before the keystone is released. The unforgeability stress only the one, who has the private key, could generate the ambiguous signature bound with the keystone. The fairness request the initial signer and matching signer are bound with the ambiguous signatures at the same time when the keystone is released.

As defined in [12], Accountability of E_commerce protocol is the property whereby the association of a unique originator with an object or action can be proved to a third party. However, accountability is not been covered in the model of [3,4,5,8,10]. Almost all concurrent signature schemes proposed in [3,4,5,6,7,8,9,10] are facing attack against accountability. Hereunder, we will define the accountability for concurrent signature.

Different from the definition of accountability for E-Commerce protocol defined in [12], the accountability of concurrent signature stress the uniqueness of the message bound with the keystone and signer. Hereunder is our definition for accountability of concurrent signature.

Definition 1: *Accountability of concurrent signature is the property whereby the unique association of a signer, a keystone set with a message can be proved to a third party based on the valid protocol messages.*

Informally speaking, accountability requires the signers could convince themselves and any third party that the messages signed in the ambiguous signature are the unique set generated in one protocol run. Any signer could not generate ambiguous signature for any messages other then the one send to other signers in his ambiguous signature, which could satisfy the VERIFY and AVERIFY algorithm.

3. Attack against the accountability of iPCS1

Before we give the attack scene against accountability based on iPCS1[5], we study a case in real world. Alice is an inexpert music maker, Bob is a manager of shopping mall, Carl is a music reseller. Bob loves one song named as "happy shopping" created by Alice and decides to buy it from Alice. Then they run a concurrent signature protocol and exchange the ambiguous signature and release the keystone. After that Alice could get money from the bank and Bob get the song. However, based on the keystone used for this concurrent signature protocol run, Alice could generate another valid ambiguous signature for another song "elegant shopping" and show it to Carl. Carl is convinced that this "elegant shopping" is welcome to shopping mall so he may decide to buy the copyright of this song. It is clear enough that this flaw of concurrent signature gives Alice the chance to get advantage over Carl and Bob.

We found this results from the fact that in proposals [3,4,5,6,7,8,9,10], only the keystones are used to bind the two ambiguity signature together. The messages which are signed and exchanged were not bind with the protocol run. Here we give the attack scene in iPCS1:

1. Assume Alice and Bob want to exchange message m_A, m_B
2. The dishonest initial signer Alice performs in the following.
   - Choose a random keystone $k \in K$ and set $s_1 = H_1(k)$
   - Run $\sigma_A \leftarrow ASIGN(y_A, y_B, x_A, s_1, m_A)$, $\sigma_A = (c, s_2, s_1)$, $s_2 = (\alpha - c) \cdot x_A^{-1} \bmod q$ $c = H_1(m_A, g^{\alpha} y_B^{s_1} \bmod p)$,
   - Run $\hat{\sigma}_A \leftarrow ASIGN(y_A, y_B, x_A, s_1, \hat{m}_A)$, $\hat{\sigma}_A = (\hat{c}, \hat{s}_2, \hat{s}_1)$, $\hat{s}_1 = s_1$, $\hat{c} = H_1(\hat{m}_A, g^{\alpha} y_B^{s_1} \bmod p)$, $\hat{s}_2 = (\alpha - \hat{c}) \cdot x_A^{-1} \bmod q$
   - Send (σ_A, m_A) to Bob and keep the $(\hat{\sigma}_A, \hat{m}_A)$ private.
3. Upon receiving (σ_A, m_A), Bob check $AVERIFY(\sigma_A, y_A, y_B, m_A) \equiv accept$. If not, then Bob just aborts. Otherwise, Bob acts as what the protocol step2 specified in section 2.2 and send $(\sigma_B, m_B, \hat{t}), \sigma_B = (c', s_2', s_1')$ to Alice.
4. After $(\sigma_B, m_B, \hat{t})$ is received, Alice acts as the protocol step3 specified as following:
   - Compute $r = \hat{t}^{x_A} \bmod p$, and $k' = r \bmod q$.
   - Test whether $s_1' = s_1 + H_1(k') \bmod q$.
   - Check whether $AVERIFY(\sigma_B, y_A, y_B, m_B) = accept$
   - If σ_B is invalid, abort. Otherwise, release the keystone *(k,k')* publicly to bind signatures $\hat{\sigma}_A$ *or* σ_A and σ_B concurrently.
5. Once the keystone *(k,k')* is available any one can verify that $\hat{\sigma}_A$ *or* σ_A and σ_B are respectively signed by Alice and Bob because following equalities hold:
   - $s_1 = H_1(k)$ and $s_1' = s_1 + H_1(k') \bmod q$.
   - $AVERIFY(\hat{\sigma}_A, y_A, y_B, \hat{m}_A) = accept$
   - $AVERIFY(\sigma_A, y_A, y_B, m_A) = accept$
   - $AVERIFY(\sigma_B, y_A, y_B, m_B) = accept$

Bob think (σ_A, m_A), $(\sigma_B, m_B, \hat{t})$ is the only valid signature after the keystones are released. But in fact, Alice may public $(\hat{\sigma}_A, \hat{m}_A)$, $(\sigma_B, m_B, \hat{t})$ as the result of this session of concurrent signature protocol.

Proposition 1. After the keystone information *(k,k')* is released, the two signature-message pairs $(\hat{\sigma}_A, \hat{m}_A)$ and (σ_B, m_B) are binding to Alice and Bob, respectively.

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

- $\hat{s}_1 = s_1 = H_1(k)$, recall step 2 ;
- $s_1' = s_1 + H_1(k')$ recall step 3;
- $AVERIFY(\hat{\sigma}_A, y_A, y_B, \hat{m}_A) = accept$ and $AVERIFY(\sigma_B, y_A, y_B, m_B) = accept$, since $(\hat{\sigma}_A, \hat{m}_A)$ and (σ_B, m_B) both are properly generated by running algorithm ASIGN in step 2 and 3 respectively.

Therefore, according to the specification of algorithm VERIFY reviewed in section 2.2 $(\hat{\sigma}_A, \hat{m}_A)$ and (σ_B, m_B) are truly binding to Alice and Bob. At the same time, (σ_A, m_A) and (σ_B, m_B) are also binding to Alice and Bob based on the same keystone. Symmetrically, dishonest Bob could also generate some $(\hat{\sigma}_B, \hat{m}_B)$ to bind with (σ_A, m_A). So the iPCS1 protocol could not fulfill the fairness of concurrent signature.

4. An update version of the iPCS1

The attack, discussed above, against the accountability of PCS and iPCS results from the fact that Alice or Bob could generate multiple ambiguous signatures for different messages based on the same keystone, for example, (σ_A, m_A) and $(\hat{\sigma}_A, \hat{m}_A)$. The fundamental reason of this attack is that the keystone is transferred to the parameter used to create the signature for messages but the messages are not used in the transformation from keystone to signature parameter. This gives chance to both Alice and Bob to generate multiple signatures for different messages based on the same keystone. It is intuitive to include the messages in the transformation process from keystone to signature parameter to avoid such attack. In order to keep the properties of the perfect concurrent signature, we propose the update version of iPCS1, called uPCS1 for short, with the following updated calculations:

- $s_1 = H_1(k, m_A)$;
- $k' = H_1(r, m_B) \bmod q$
- $s_1' = s_1 + H_1(k') \bmod q$

The assumption, algorithms: SETUP. ASIGN, AVERIFY, VERIFY, and protocol structure keeps the same as iPCS1. Hereunder is uPCS1:

1. The initial Alice performs as follow:
   - Choose a random keystone $k \in K$ and set $s_1 = H_1(k, m_A)$.
   - Run $\sigma_A \leftarrow ASIGN(y_A, y_B, x_A, s_1, m)$. Let $\sigma_A = (c, s_2, s_1)$.
   - Send (σ_A, m_A) to the matching signer Bob.

2. Upon receiving (σ_A, m_A), Bob checks $AVERIFY(\sigma_A, y_A, y_B, m_A) = accept$. If not, then Bob just aborts. Otherwise, Bob acts in the following way.
   - Pick a random $t \in Z_q$ and compute $\hat{t} = y_B^t \bmod p$
   - Compute $r = (y_A^{x_B})^t \bmod p$, and $k' = H_1(r, m_B) \bmod q$.
   - Set $s_1' = s_1 + H_1(k') \bmod q$.
   - Run $\sigma_B = (c', s_2', s_1') \leftarrow ASIGN(y_B, y_A, x_B, s_1', m_B)$.
   - Send $(\sigma_B, m_B, \hat{t})$ to Alice.
3. After $(\sigma_B, m_B, \hat{t})$ is received, Alice performs as follows.
   - Compute $r = \hat{t}^{x_A} \bmod p$, and $k' = H_1(r, m_B) \bmod q$.
   - Test whether $s_1' = s_1 + H_1(k') \bmod q$.
   - Check whether $AVERIFY(\sigma_B, y_A, y_B, m_B) = accept$
   - If σ_B is invalid, abort. Otherwise, release the keystone *(k,k')* and Alice's signature on the keystone: $Sig_{x_A}(A, B, (k, k'))$. 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.
   - Check the vality of $Sig_{x_A}(A, B, (k, k'))$ based on *(k,k')*
   - $s_1 = H_1(k, m_A)$ and $s_1' = s_1 + H_1(k') \bmod q$
   - $AVERIFY(\sigma_A, y_A, y_B, m_A) = accept$
   - $AVERIFY(\sigma_B, y_A, y_B, m_B) = accept$

Alice releases the keystone and his signature on it at the same time. This makes sure that Bob will not forge valid keystones. Based on the results in [3,4,5] and the discussions previously provided, it is easy to see that the uPCS1 is truly perfect concurrent signature protocol because the uPCS1 dose not change the protocol structure and algorithm used in iPCS1. Compare to iPCS1, the uPCS1 has the same efficiency because uPCS1 only add one parameter for hash function.

Proposition 2: uPCS1 satisfy the accountability. (σ_A, m_A) and (σ_B, m_B) are the unique ambiguous signature bound with keystone *(k, k')* which satisfy the VERIFY algorithm.

Proof: Assume Alice generate another ambiguous signature $(\tilde{\sigma_A}, \tilde{m_A})$ which satisfy VERIFY algorithm. Then formulae $s_1^{\sim} = H_1(k, \tilde{m_A})$ (1) and $s_1^{'} = s_1^{\sim} + H_1(k^{'})$ (2) have to be hold. However, it is obvious that formula (2) could not hold in protocol uPCS1. So Alice could not generate a valid $(\tilde{\sigma_A}, \tilde{m_A})$. Bob could generate the only valid ambiguous signature. Even Bob generates another ambiguous signature $(\tilde{\sigma_B}, \tilde{m_B})$, but Bob could not generate or forge any valid keystone and Alice's signature on the keystone, so $(\tilde{\sigma_B}, \tilde{m_B})$ can not be verified by other third party. Based on above discussion, uPCS1 satisfy the accountability of concurrent signature.

5. Conclusion and future work

Concurrent signature [3] provides a novel idea for fair exchange protocol which enables two distrusted party to exchange digital items without any third party. Reference [4] proposed the perfect concurrent signature to strengthen the ambiguity of the concurrent signature. And then [5] points out there has an attack against the fairness of PCS and propose the iPCS. In this paper, we found that dishonest signer could generate multiple signature pairs for different messages based on the same keystone in iPCS, which is absolutely unfair. To avoid such attack, we propose the uPCS1 to bind the keystone and messages with the signer as discussed in section 3. Most important, we observed that concurrent signature proposals in [3,4,5,6,8,9,10] also have such flaw. This spurs us to define a new security property, accountability, for concurrent signature. As the future work, we will try to implement some concrete fair exchange applications, especially for fair exchanges in P2P network, based on the uPCS1.

References

[1] Asokan N, Shoup V, Waidner M. Optimistic fair exchange of digital signatures. IEEE Journal on Selected Areas in Communication, 18(4), pp.593--610,(2000)
[2] S.Kremer, O.Markowitch, and J.Zhou. An intensive survey of fair non-repudiation protocols. Computer Communications, 25(17), pp.1606-1621, Nov.2002.
[3] L. Chen, C. Kudla, and K. G. Paterson. Concurrent signatures. In: Eurocrypt ’04, LNCS 3027, pp. 287-305. Spriger, Berlin (2004)
[4] W. Susilo, Y. Mu, and F. Zhang. Perfect concurrent signature schemes. In: ICICS ’04, LNCS, Vol. 3269, pp. 14-26. Springer, Berlin. (2004)
[5] Wang Gui - lin，Bao Feng，Zhou Jian - ying.The fairness of perfect concurrent signatures, In: ICICS’06, LNCS, Vol. 4307, pp. 435-451, Springer, Berlin (2006)
[6] S.S.M. Chow and W.Susilo, Generic construction of (identity-based) perfect concurrent signatures, ICICS’05, LNCS, Vol. 3783, pp. 194–206, Springer, Berlin (2005)
[7] K. Nguyen, Asymmetric Concurrent Signatures, ICICS2005, LNCS, Vol. 3783, Springer, Berlin (2005)
[8] W. Susilo and Y. Mu, Tripartite concurrent signatures, IFIP/SEC’05, pp.425–441, (2005)
[9] Xinwen Zhang, Shangping Wang, Xiaofeng Wang, Identity-based Multi-party Concurrent Signatures. ChinaCrypt’2007: 325~327, Chengdu China(2007)
[10] Dongyvu Tonien, Willy Susilo and ReihanehSafavi-Naini, Multi-party Concurrent Signatures, ISC06, LNCS, Vol. 4176, pp.131-145, Springer-Verlag Berlin (2006)
[11] Dengguo Feng, Weidong Chen, Security Model and Modular Design of Fair Authenticated Key Exchange Protocols, 3rd SKLOIS Workshop, Security protocols, pp.1-18, Beijing, (2007)
[12] Kailar R. Accountability in electronic commerce protocols. IEEE Trans. on Software Engineering, 22(5), pp. 313−328. (1996)