SCIENCE CHINA
Information Sciences

• RESEARCH PAPERS •

May 2010 Vol. 53 No. 5: 945–953
doi: 10.1007/s11432-010-0065-1

Fair exchange signature schemes

LIU JingWei[1,2*], SUN Rong[1] & KWAK KyungSup[2]

[1]*Key Laboratory of Computer Network and Information Security, Ministry of Education, Xidian University, Xi'an* 710071*, China;*
[2]*UWB Wireless Communications Research Center, Inha University, Incheon* 402751*, Korea*

Received July 8, 2008; accepted July 29, 2009; published online March 22, 2010

Abstract In this paper a new class of fair exchange signature scheme (FESS) is proposed that allows two players to exchange digital signatures in a fair way. The new signature scheme is a general model and has various implementations based on most of the existing signature schemes; thus it may also be considered as an interesting extension of concurrent signature presented in EUROCRYPT 2004 that is constructed from ring signatures. In FESS, two unwakened signatures signed respectively by two participants can be verified easily by each player, but it would not go into effect until an extra piece of commitment *keystone* is released by one of the players. Once the *keystone* is revealed, two signatures are both aroused and become valid. A key feature of FESS is that two players can exchange digital signatures simultaneously through a secret commitment *keystone* without involvement of any trusted third party (TTP). Moreover, the efficiency of the new scheme is higher than that of concurrent signatures.

Keywords FESS, concurrent signature, Schnorr signature, fair exchange, electronic commerce

Citation Liu J W, Sun R, Kwak K S. Fair exchange signature schemes. Sci China Inf Sci, 2010, 53: 945–953, doi: 10.1007/s11432-010-0065-1

1 Introduction

Open networks such as the Internet lay a stable foundation for electronic commerce, which usually involves two distrusted parties exchanging their items from each other, such as e-commerce payment protocols, electronic contract signing, and certified e-mail delivery. Due to the rapid growth of electronic commerce nowadays, fair exchange turns out to be an increasingly important topic. A digital exchange problem is deemed to be fair if at the end of exchange, either party receives the expected item or neither party receives it. In general scenarios, digital items' exchanges have to be carried out over open networks and both participants may not trust each other. There could be subsequent disputes about what was exchanged during a transaction even if the exchange itself was completed fairly. In this case, evidence should be accumulated during the exchange to enable the settlement of any future disputes.

In the recent years various schemes on the fair exchange problem have been proposed and reported in the literature. These schemes often fall into three categories:

Solutions to fair exchange problem are often gradual exchange protocols [1–7] where two parties have to be too interactive and cumbersome for exchanging digital items by many steps. Nevertheless, these

*Corresponding author (email: j_w_liu@hotmail.com)

methods cannot provide fairness fully, because at the end of protocols, one player often has an advantage of one more bit than the other player does. In [3], the authors introduced timed commitments. A timed commitment is a commitment scheme in which there is an optional forced opening phase enabling the receiver to recover (with effort) the committed value without the help of the committer. But this method is only considered for Rabin and RSA signatures of a special kind. In [7], the authors show how to achieve timed fair exchange of digital signatures of standard type. Their construction follows the gradual release paradigm, and works on a new "time" structure, called a mirrored time-line. But the length of it leads to another apparent problem, which is making sure that the underlying sequence has a period large enough so that cycling is not observed.

Recently, lots of researches on fair exchange protocols mainly exploit an on-line or off-line trusted third party (TTP) [8–21], which is involved in protocols run (on-line) or account opening and disputes (off-line). However, either on-line or off-line TTP may cause the bottleneck problem and at least inefficiencies in the operation though its involvement is further reduced in [8, 12, 20]. In [8], the authors introduced a protocol that allows two players to exchange digital signatures over the Internet in a fair way. The protocol relies on a trusted third party, but is "optimistic," in which the third party is only required in cases that one player attempts to cheat or simply crashes. The key feature of the protocol is that a player can always force a timely and fair termination without the cooperation of the other player.

The latest direction of fair exchange is supposed to overleap TTP in the protocols. Two participants carry out digital items' exchange by using special signatures. In this way, some new fair exchange protocols are designed, which become more efficient than prior arts do. The concept of concurrent signatures was introduced by Chen et al. [22] in Eurocrypt 2004. Such signature schemes allow two parties to generate and exchange two signatures that are ambiguous until an extra piece of information (called keystone) is released by one of the parties, which exploits the ambiguity property enjoyed by the ring signatures [23, 24]. 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. Concurrent signature allows to build a fair exchange protocol in which two parties interact to exchange digital items without the involvement of the trusted third party. But it is at the sacrifice that the initial party controls the keystone and therefore he/she has an extra right to decide when the keystone is released. In ICICS 2004, Susilo et al. [25] further proposed a perfect concurrent signatures to strengthen the ambiguity of concurrent signatures. That is, even if both signers are known to have 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. But in [26], Wang et al. point out that Susilo et al.'s two perfect concurrent signature schemes are actually not concurrent signatures and present an effective way to avoid this attack.

However, most of the previous researches on fair exchange are not fully suitable for the applications on open networks, because most fair exchange applications have to be provided not only security but also efficiency. In our opinion, an ideal solution for fair exchange should be both secure and efficient. With the opinion, we propose a new class of fair exchange signature schemes (FESS) in this paper that allows two players to exchange digital signatures over open computer and communication networks (such as the Internet) in a fair way, so that either player obtains the other's signature at the same time, or neither player does. In FESS, each unwakened signature signed by two participants respectively can be verified easily by each player, but it does not go into effect until an extra piece of information *keystone* is released by one of the players. Once the *keystone* is revealed, two signatures are both aroused and become valid. The FESS can be applied to different application environments through various implementations. We will also introduce how to construct an implementation of FESS without a TTP. Thus it provides a valid primitive that is of interest in designing fair exchange schemes. Of course there might be other applications to consider. A key feature of FESS is that two players can exchange digital signatures simultaneously without involvement of any TTP. FESS does not overcome the weakness of concurrent signature, that the initial party controls the keystone, but it has higher efficiency than concurrent signature.

The main contributions in this paper are listed as follows:

1. A generic definition of FESS is proposed.
2. The way to construct FESS without TTP is demonstrated.
3. The security proof of FESS in the random oracle model is provided.

2 Basic definitions

In this section, we introduce the basic definitions of FESS. The parameters involved in the new schemes are depicted as follows.

- a plaintext message space $\mathcal{M}$: a set of strings over some alphabet;
- a keystone message space $\mathcal{K}_{ks}$: a set of strings over some alphabet;
- a keystone fix space $\mathcal{K}$: a set of possible keystone fix volume;
- a signature space $\mathcal{S}$: a set of possible signatures;
- a signing key space $\mathcal{X}$: a set of possible keys for signature creation;
- a verification key space $\mathcal{Y}$: a set of possible keys for signature verification;

Definition 1. A full FESS consists of five procedures (Parameter Setup, KGen, Sign, SVerify, KVerify):

- An efficient probabilistic algorithm Parameter Setup:

$$k \to \langle \{x_i\}, \{y_i\}, \text{ description of } \{\mathcal{M}, \mathcal{K}_{ks}, \mathcal{K}, \mathcal{S}\}\rangle,$$

where k is a security parameter, $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

- An efficient one-way function KGen: $\mathcal{K}_{ks} \to \mathcal{K}$, which generates a keystone fix $k \in \mathcal{K}$ with an input $keystone \in \mathcal{K}_{ks}$. Secure hash functions can be used as KGen.

- An efficient probabilistic signing algorithm Sign: $\mathcal{M} \times \mathcal{K} \times \mathcal{X} \to \mathcal{S}$, for any message $m \in \mathcal{M}$, keystone fix $k \in \mathcal{K}$ and private key $x \in \mathcal{X}$, we denote $s \leftarrow \text{Sign}_x(m, k)$ where $s \in \mathcal{S}$.

- An efficient signature verification algorithm SVerify: $\mathcal{M} \times \mathcal{K} \times \mathcal{S} \times \mathcal{Y} \to \{\text{True, False}\}$, for any $m \in \mathcal{M}$, $k \in \mathcal{K}$, and $y \in \mathcal{Y}$, it is necessary that

$$\text{SVerify}_y(m, k, s) = \text{True or False.} \tag{1}$$

- An efficient keystone verification algorithm KVerify:

$$\mathcal{M} \times \mathcal{K} \times \mathcal{S} \times \mathcal{Y} \times \mathcal{K}_{ks} \to \{\text{True, False}\},$$

for any $m \in \mathcal{M}$, $k \in \mathcal{K}$, $y \in \mathcal{Y}$ and $keystone \in \mathcal{K}_{ks}$, it is necessary that

$$\text{KVerify}_y(m, k, s, keystone) = \begin{cases} \text{True, if SVerify}_y(m, k, s) = \text{True and } k = \text{KGen}(keystone). \\ \text{False, elsewise.} \end{cases} \tag{2}$$

3 Basic models

3.1 Fair exchange signature protocols

In the normal case, most of fair exchange schemes often involve an on-line or off-line third party, but, by using an embedded commitment, FESS is performed only between two participants, without loss of generality, Alice (initial signer) and Bob (respond signer). Alice who initiates the protocol generates a piece of secret information–*keystone* randomly, signs a message with her private key and a keystone fix that is generated by a one-way function with the input *keystone*, and sends the signature message to Bob. Bob responds to this message by signing another message with his private key and the same keystone fix.

Following Definition 1, the detailed implementation of FESS is depicted as follows.

Alice and Bob first choose an efficient signature scheme and the relevant parameters. Let $x_A, x_B \in \mathcal{X}$ denote Alice's and Bob's private key respectively and $y_A, y_B \in \mathcal{Y}$ denote the public key corresponding to the private key of two participants.

1. Alice chooses a $keystone \in \mathcal{K}_{ks}$ randomly and computes $k = \mathrm{KGen}(keystone)$, where $k \in \mathcal{K}$. And she uses k and her private key x_A to sign a given message m_A agreed with Bob. The verifiable signature message is $\sigma_A = \langle m_A, k, s_A \rangle$, where $s_A = \mathrm{Sign}_{x_A}(m_A, k)$, and is sent to Bob.

2. After receiving Alice's signature message σ_A, Bob verifies the message σ_A by running algorithm SVerify described in section 2. If $\mathrm{SVerify}_{y_A}(\sigma_A) = \mathrm{True}$, Bob chooses a message m_B agreed with Alice and uses k and his private key x_B to generate a signature $s_B = \mathrm{Sign}_{x_B}(m_B, k)$. Bob sends the verifiable signature message $\sigma_B = \langle m_B, k, s_B \rangle$ back to Alice. Otherwise, if $\mathrm{SVerify}_{y_A}(\sigma_A) = \mathrm{False}$, Bob aborts. Note that Bob uses the same value k as Alice does.

3. After receiving Bob's verifiable signature message σ_B, Alice also verifies the message σ_B by running algorithm SVerify. If $\mathrm{SVerify}_{y_B}(\sigma_B) = \mathrm{True}$, Alice releases $keystone$ to arouse not only σ_B but also σ_A; thus two verifiable signatures go into effect at the same moment. If $\mathrm{SVerify}_{y_B}(\sigma_B) = \mathrm{False}$, Alice aborts.

4. Everyone can verify whether $\mathrm{KVerify}_{y_A}(\sigma_A, keystone) = \mathrm{True}$ or $\mathrm{KVerify}_{y_B}(\sigma_B, keystone) = \mathrm{True}$.

Here we need to point out that FESS provides fairness through the dormancy property, which is different from the ambiguous property of concurrent signature [22]. As a useful cryptographic tool, our scheme provides a primitive to build efficient fair exchange protocols. In the next section, we will give a specific implementation of FESS.

3.2 Attack model for FESS

For a secure signature scheme, the property of secure against existential forgery on adaptively chosen message attack is necessary. In formal security model in [27, 28], an adversary wins the game if he outputs a valid pair of a message and a signature, where he is allowed to require the signer to sign any message except for the message that the adversary has to sign. Here we will introduce an attack model for FESS, similarly to that in [28]. We say that a FESS, which consists of five algorithms: Parameter Setup, KGen, Sign, SVerify, KVerify, is secure against existential forgery on adaptively chosen message if no polynomial time algorithm $\mathcal{A}$ has a non-negligible advantage against a challenger $\mathcal{C}$ in the following game:

1. $\mathcal{C}$ runs Parameter Setup algorithm firstly and gives the public system parameters to $\mathcal{A}$.

2. $\mathcal{A}$ can require the following queries:

(a) Hash function query. Given a requested input, $\mathcal{C}$ outputs the value of hash function to respond to $\mathcal{A}$.

(b) KGen query. $\mathcal{A}$ can request that $\mathcal{C}$ selects a $keystone \in \mathcal{K}_{ks}$ and returns the fix $k = \mathrm{KGen}(keystone)$.

(c) KReveal query. $\mathcal{A}$ can request the keystone which corresponds to an existing keystone fix $k \in \mathcal{K}$ generated by a previous KGen query.

(d) Sign query. Given a message $m \in \mathcal{M}$ and a $k \in \mathcal{K}$, $\mathcal{C}$ returns a signature s which is generated by running Sign algorithm.

3. $\mathcal{A}$ outputs $\langle m, k, s \rangle$, where m is a message, k is a keystone fix and s is a signature such that $\langle m, k \rangle$ is not equal to the any previous input of Sign query and k is not only an existing output of KGen query but also an input of KReveal query. $\mathcal{A}$ wins the game if s is a valid signature of $\langle m, k \rangle$.

By using this attack model, we can reduce the security of FESS to the hardness of discrete logarithm problem in section 5.

4 An implementation of FESS

In this section, we give an implementation of FESS actually based on Schnorr signature. We introduce the system parameters firstly.

Parameter settings:

• System parameters: Let p and q be two large primes and $q|p-1$. The notation g denotes an element of order q of Z_p^*.

• Alice: Alice has a pair of keys (x_A, y_A) for Schnorr signature, where x_A is Alice's private key, y_A is her public key and $y_A = g^{-x_A} \bmod p$.

• Bob: Bob has a pair of keys (x_B, y_B) for Schnorr signature where x_B is Bob's private key, y_B is his public key and $y_B = g^{-x_B} \bmod p$.

1. Alice chooses $keystone = \langle ID_{AB} \rangle$ and computes $k = \mathrm{G}(ID_{AB})$, where ID_{AB} is a piece of random information about Alice's and Bob's identity and G is a hash function. Alice generates her signature $s_A = \mathrm{Sign}_{x_A}(m_A, k) = \langle r_A, e_A, c_A \rangle$, where $r_A = g^{k_A} \bmod p$, $e_A = \mathrm{H}(m_A, k, r_A)$, $c_A = k_A + e_A x_A \bmod q$, and H is a hash function. The verifiable signature message is $\sigma_A = \langle m_A, k, s_A \rangle$ and is sent to Bob.

2. Bob verifies σ_A by running SVerify algorithm. If $e_A = \mathrm{H}(m_A, k, g^{c_A} y_A^{e_A} \bmod p)$, Bob generates $s_B = \mathrm{Sign}_{x_B}(m_B, k) = \langle r_B, e_B, c_B \rangle$ and sends $\sigma_B = \langle m_B, k, s_B \rangle$ to Alice, otherwise aborts.

3. Alice verifies σ_B by running SVerify algorithm. If $e_B = \mathrm{H}(m_B, k, g^{c_B} y_B^{e_B} \bmod p)$, Alice releases *keystone*, otherwise aborts.

4. Each participant can prove σ_A (or σ_B) valid by verifying $k = \mathrm{G}(ID_{AB})$, $\mathrm{SVerify}_{y_A}(\sigma_A) = \mathrm{True}$ (or $\mathrm{SVerify}_{y_B}(\sigma_B) = \mathrm{True}$).

From the above implementation, we conclude that FESS is a general model and can be implemented based on most of the existing signature schemes, therefore, FESS has higher efficiency than concurrent signature does. We will show efficiency comparison between FESS and concurrent signature in the next section.

5 Security and efficiency analysis of FESS

5.1 Security

In this subsection, we will discuss the security of FESS in the random oracle model [29].

Lemma 5.1 (Completeness)**.** The proposed scheme achieves the property of completeness.

Proof. If $s = \mathrm{Sign}_x(m, k) = \langle r, e, c \rangle$, $r = g^{k'} \bmod p$, $e = \mathrm{H}(m, k, r)$ and $c = k' + ex \bmod q$ then $e = \mathrm{H}(m, k, g^c y^e \bmod p) \Leftrightarrow \mathrm{SVerify}_y(m, k, s) = \mathrm{True}$. Moreover, if $\mathrm{SVerify}_y(m, k, s) = \mathrm{True}$ and $k = \mathrm{G}(keystone)$ then $\mathrm{KVerify}_y(m, k, s, keystone) = \mathrm{True}$.

Thus, the proposed scheme achieves the property of completeness.

To prove the property of Unforgeability of FESS, we have to introduce an important conclusion—Forking Lemma [28] firstly. It gives a reductionist security proof for triplet ElGamal-family signature schemes which produce a signature (Gen, Sign, Verify) on an input message m.

Lemma 5.2 (Forking Lemma)**.** Let $\mathcal{A}$ be a probabilistic polynomial time Turing machine whose input only consists of public data. We denote respectively by Q and R the number of queries that $\mathcal{A}$ can ask to the random oracle and the number of queries that $\mathcal{A}$ can ask to the signer. Assume that, within time bound K, $\mathcal{A}$ produces, with probability $\varepsilon \geqslant 10(R+1)(R+Q)/2^k$ (where k is a security parameter), a valid signature $(m, \sigma_1, h, \sigma_2)$. If the triples (σ_1, h, σ_2) can be simulated without knowing the secret key, with an indistinguishable distribution probability, then there is another machine which has control over the machine obtained from $\mathcal{A}$ replacing interaction with the signer by simulation and produces two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1, h', \sigma_2')$ such that $h \neq h'$ in expected time $T' \leqslant 120686QT/\varepsilon$.

Lemma 5.3 (Unforgeability)**.** The FESS is unforgeable under a chosen message attack in the random oracle model.

Proof. The proof refers to the proof of unforgeability of the signature scheme [27, 28] by Pointcheval and Stern. We suppose that G and H are random oracles, and there exists a probabilistic polynomial time Turing machine $\mathcal{A}$ whose input only consists of public data. We assume that $\mathcal{A}$ can make Q_{G} queries to the random oracle G, Q_{H} queries to the random oracle H and Q_{S} queries to the signing oracle Sign. Within time bound T, $\mathcal{A}$ generates a valid signature $\langle m, k, \langle r, e, c \rangle \rangle$ with probability $\varepsilon \geqslant 10Q_{\mathrm{G}}(Q_{\mathrm{S}}+1)(Q_{\mathrm{S}}+Q_{\mathrm{H}})/2^l$ (where l is a security parameter).

Simulation. $\mathcal{C}$ gives the parameters $\langle g, p, q \rangle$ and $y = g^{-x} \bmod p$ to $\mathcal{A}$. $\mathcal{C}$ tries to simulate the challenger by simulating all the oracles to gain the secret key x. $\mathcal{A}$ can make queries as follows:

G-queries. $\mathcal{A}$ can query the random oracle G at any time. $\mathcal{C}$ simulates this random oracle by maintaining a list of tuple $\langle m_i, k_i\rangle$ which is called G-list. When the oracle is queried with an input $m \in \{0,1\}^*$, $\mathcal{C}$ responds as follows:

1. If the query m is already in an entity $\langle m, k_i\rangle$ of G-list, $\mathcal{C}$ outputs k_i.
2. Otherwise $\mathcal{C}$ selects $k \in \mathcal{K}$ randomly, outputs k and adds $\langle m, k\rangle$ to G-list.

H-queries. $\mathcal{A}$ can query the random oracle H at any time. $\mathcal{C}$ simulates this random oracle by maintaining a list of tuple $\langle \sum_i, e_i\rangle$ which is called H-list, where $\sum_i$ is a triple of $\langle m_i, k_i, r_i\rangle$. When the oracle is queried with an input $\sum$, $\mathcal{C}$ responds as follows:

1. If the query $\sum$ is already in an entity $\langle \sum, e_i\rangle$ of H-list, $\mathcal{C}$ outputs e_i.
2. Otherwise $\mathcal{C}$ selects $e \in \mathbb{Z}_q$ randomly, outputs e and adds $\langle \sum, e\rangle$ to H-list.

KGen-queries. $\mathcal{C}$ maintains a K-list of tuples $\langle keystone, k\rangle$. $\mathcal{A}$ can request that $\mathcal{C}$ selects a $keystone \in \mathcal{K}_{ks}$ and returns a keystone fix $k = \mathrm{G}(keystone)$. $\mathcal{C}$ chooses a random $keystone \in \mathcal{K}_{ks}$ and computes $k = \mathrm{G}(keystone)$. $\mathcal{C}$ outputs k and adds $\langle keystone, k\rangle$ to K-list. In fact, K-list is a sublist of G-list and is only required to answer KReveal queries.

KReveal-queries. $\mathcal{A}$ can request the keystone which corresponds to an existing keystone fix $k \in \mathcal{K}$ generated by previous KGen query. If there exists a tuple $\langle keystone, k\rangle$ in K-list, then $\mathcal{C}$ returns $keystone$, otherwise it outputs an invalid status.

Sign-queries. $\mathcal{C}$ simulates the signature oracle by responding queries of the form $\langle m, k\rangle$ where $m \in \mathcal{M}$ is the message to be signed and $k \in \mathcal{K}$ is a keystone fix. $\mathcal{C}$ answers the query as follows:

1. $\mathcal{C}$ picks two random numbers c, $e \in \mathbb{Z}_q$ where e is not equal to any previous output of H oracle.
2. $\mathcal{C}$ computes $r = g^c y^e \bmod p$. If $\sum = \langle m, k, r\rangle$ is equal to any previous input of H oracle, then return to step 1.
3. $\mathcal{C}$ adds $\langle \sum, e\rangle$ to H-list.
4. $\mathcal{C}$ outputs $s = \langle r, e, c\rangle$ as the signature for message m.

Note. Here we have to verify if the distribution of real signature δ is consistent with the forged signature δ'.

$$\begin{cases} \delta = \{(r,e,c) | k \in \mathbb{Z}_q, k \neq 0, e \in \mathbb{Z}_q,\ r = g^k \bmod p, c = k + xe \bmod q\}, \\ \delta' = \{(r,e,c) | e \in \mathbb{Z}_q, c \in \mathbb{Z}_q, r = g^c y^e \neq 1 \bmod p\}. \end{cases} \tag{3}$$

Firstly, we compute the probability of the real signature signed using secret key

$$\Pr_{\delta}[(r,e,c) = (\varepsilon,\beta,\gamma)] = \Pr_{k\neq 0,e}[r = g^k = \varepsilon, e = \beta, c = k + xe = \gamma] = \frac{1}{q(q-1)}. \tag{4}$$

The probability of the forged signature is

$$\Pr_{\delta'}[(r,e,c) = (\varepsilon,\beta,\gamma)] = \Pr_{e,c}[e = \beta, c = \gamma, r = g^c y^e = \varepsilon \neq 1 \bmod p] = \frac{1}{q(q-1)}. \tag{5}$$

So the triple $\langle r, e, c\rangle$ can be simulated without knowing the secret key, with an indistinguishable distribution probability. The signing oracle simulated by $\mathcal{C}$ has high quality, therefore, $\mathcal{A}$ is satisfied with the Sign-queries' answer and can fully exert his forgery ability.

Output. Finally, with a non-negligible probability, $\mathcal{A}$ outputs a valid signature $s = \langle r, e, c\rangle$ of a message $m \in \mathcal{M}$ and $k \in \mathcal{K}$, where $\mathcal{A}$ obtains $k = \mathrm{G}(keystone)$ through KGen queries and makes a KReveal query with the input k and no Sign query with the input $\langle m, k\rangle$.

Now $\mathcal{C}$ plays the simulation twice so that $\mathcal{A}$ can obtain two valid signatures $s = \langle r, e, c\rangle$ and $s' = \langle r, e', c'\rangle$ with $e \neq e'$. Then we have

$$r = g^c y^e = g^{c-xe} = g^{c'-xe'} = g^{c'} y^{e'} \bmod p. \tag{6}$$

By (6), $\mathcal{C}$ can solve the hard discrete logarithm:

$$\log_g y = -x = \frac{c - c'}{e' - e} \bmod q, \tag{7}$$

within expected time less than $\frac{120686\times 2^l\times Q_{\rm H}T}{10\times(Q_{\rm S}+1)\times(Q_{\rm S}+Q_{\rm H})}$. This contradicts the hardness of the discrete logarithm problem.

In [22], to prove the concurrence of two participants' signatures, Chen et al. make use of the ambiguous property of ring signatures. No one can confirm who is the signer within two participants unless initial signer releases the keystone. But in FESS, to ensure simultaneity, we introduce the property of Dormancy. Two signatures that have been exchanged are not valid until the secret information *keystone* is released to arouse them.

Lemma 5.4 (Dormancy)**.** The proposed scheme achieves the property of dormancy before the secret information *keystone* is released.

Proof. The random oracle assumption is the same as before. We suppose there exists a probabilistic polynomial time Turing machine $\mathcal{A}$ whose input only consists of public data. We assume that $\mathcal{A}$ can make $Q_{\rm G}$ queries to the random oracle G, $Q_{\rm K}$ queries to the random oracle KGen and $Q_{\rm S}$ queries to the signing oracle Sign.

Simulation. $\mathcal{C}$ gives the parameters $\langle g,p,q\rangle$ and $y=g^{-x} \bmod p$ to $\mathcal{A}$. $\mathcal{C}$ tries to simulate the challenger by simulating all the oracles to reveal a *keystone* with $k={\rm G}(keystone)$. $\mathcal{A}$ can query as in Lemma 5.3.

Output. Finally, with a non-negligible probability, $\mathcal{A}$ outputs a *keystone* and a valid signature $s=\langle r,e,c\rangle$ with a message $m\in\mathcal{M}$ and $k\in\mathcal{K}$, where $\mathcal{A}$ obtains $k={\rm G}(keystone)$ through KGen queries and does not make KReveal query with the input k.

In this case, it is easy for $\mathcal{A}$ to obtain a valid signature $s=\langle r,e,c\rangle$ through Sign-queries. But $\mathcal{A}$ cannot make any query to KReveal-queries with the input k, so he can only reveal *keystone* with a negligible probability $\frac{Q_{\rm G}Q_{\rm K}}{2^l}$. This contradicts the assumption that, with a non-negligible probability, $\mathcal{A}$ outputs a *keystone* and a signature $s=\langle r,e,c\rangle$.

Lemma 5.5 (Fairness)**.** The proposed scheme achieves the property of fairness.

Proof. The random oracle assumption is the same as in Lemma 5.3. We suppose there exists a probabilistic polynomial time Turing machine $\mathcal{A}$ whose input only consists of public data. We assume that $\mathcal{A}$ can make $Q_{\rm G}$ queries to the random oracle G, $Q_{\rm H}$ queries to the random oracle H, $Q_{\rm K}$ queries to the random oracle KGen and $Q_{\rm S}$ queries to the signing oracle Sign.

Simulation. $\mathcal{C}$ gives the parameters $\langle g,p,q\rangle$ and $y=g^{-x} \bmod p$ to $\mathcal{A}$. $\mathcal{C}$ tries to simulate the challenger by simulating all the oracles to gain the secret key x or reveal a *keystone* with $k={\rm G}(keystone)$. $\mathcal{A}$ can query as before.

Output. Finally, with a non-negligible probability, $\mathcal{A}$ outputs a *keystone* and a valid signature $s=\langle r,e,c\rangle$ with a message $m\in\mathcal{M}$ and $k\in\mathcal{K}$. One of the following two cases holds:

1. $\mathcal{A}$ obtains $k={\rm G}(keystone)$ through KGen queries and makes KReveal query with the input k and no Sign query with the input $\langle m,k\rangle$.

2. $\mathcal{A}$ produces $k={\rm G}(keystone)$ through KGen queries and does not make KReveal query with the input k.

In case 1, it is easy to educe a contradiction from Lemma 5.3. In case 2, the output conditions only occur with a negligible probability from Lemma 5.4.

Theorem 5.6. The FESS is secure in the random oracle model, assuming the hardness of the discrete logarithm problem.

Proof. The proof follows directly from completeness, unforgeability, dormancy and fairness.

5.2 Efficiency

Because FESS can be implemented based on more efficient signature schemes, it has higher efficiency than concurrent signature does. Executive efficiency comparison between FESS and concurrent signature is given in Table 1. In Table 1, "E" denotes the number of exponentiation in $\mathbb{Z}_p$, "M_p" denotes the

Table 1 Efficiency comparison

Algorithm	FESS	Concurrent signature
Initial Sign	$1E + 1M_q + 1A + 2H$	$2E + 1M_q + 1M_p + 2A + 2H$
Respond Sign	$1E + 1M_q + 1A + 1H$	$2E + 1M_q + 1M_p + 2A + 1H$
SVerify	$2E + 1M_p + 1H$	$3E + 2M_p + 1A + 1H$
KVerify	$2E + 1M_p + 2H$	$3E + 2M_p + 1A + 2H$

number of multiplication in $\mathbb{Z}_p$, "M_q" denotes the number of multiplication in $\mathbb{Z}_q$, "A" denotes the number of addition in $\mathbb{Z}_q$, "H" denotes the number of hash operation.

6 Conclusions

In this paper we propose a secure and efficient signature scheme—FESS that allows two players to exchange digital signatures in a fair way. It is a general model and can be implemented based on most of the existing signature schemes. In FESS, two unwakened signatures that are exchanged can be verified easily by each player, but they do not go into effect until an extra piece of information *keystone* is released by one of the players. Once the *keystone* is released, two signatures are both aroused and become effective. A key feature of the proposed scheme is that two players can exchange digital signatures simultaneously through a secret commitment without involvement of any trusted third party. Although FESS does not overcome the weakness of concurrent signature, that the initial party controls the keystone, we can point out that the executive efficiency of FESS is higher than that of concurrent signatures from the above comparison. For various implementations based on most of the existing signature schemes, FESS can be applied to different environments. As a useful cryptographic tool, FESS provides a primitive to build efficient fair exchange protocols.

The new scheme can also be extended to the multi-party case easily, in which the security assumption can be proved in the same way. In future research we will try to reduce the initial signer's advantage of revelation of *keystone*.

Acknowledgements

This work was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (Grant No. NIPA-2010-C1090-1011-0007), the Korea Foundation for Advanced Studies' International Scholar Exchange Fellowship for the academic year of 2009-2010, Korea and the National High-Tech Research & Development Program of China (Grant No. 2007AA01Z472), the 111 Project (B08038), China.

References

1 Brickell E F, Chaum D, Damgard I B, et al. Gradual and verifiable release of a secret. In: Proc. of Crypto'87, Lecture Notes in Computer Science, Vol. 293. Berlin: Springer-Verlag, 1987. 156–166
2 Ben-Or M, Goldreich O, Micali S, et al. A fair protocol for signing contracts. IEEE Trans Inf Theory, 1990, 36: 40–46
3 Boneh D, Naor M. Timed commitments (extended abstract). In: Proc. of Crypto'00, Lect Notes in Comput Sci, Vol. 1880. Berlin: Springer-Verlag, 2000. 236–254
4 Cleve R. Controlled gradual disclosure schemes for random bits and their applications. In: Proc. of Crypto'89, Lect Notes in Comput Sci, Vol. 435. Berlin: Springer-Verlag, 1989. 573–588
5 Damgard I B. Practical and provably secure release of a secret and exchange of signatures. In: Proc. of Eurocrypt'93, Lect Notes in Comput Sci, Vol. 765. Berlin: Springer-Verlag, 1993. 200–217
6 Goldreich O. A simple protocol for signing contracts. In: Proc. of Crypto'83. New York: Plenum Press, 1984. 133–136
7 Garay J, Pomerance C. Timed fair exchange of standard signatures. In: Proc. of Financial Cryptography 2003, Lect Notes in Comput Sci, Vol. 2742. Berlin: Springer-Verlag, 2003. 190–207
8 Asokan N, Shoup V, Waidner M. Optimistic fair exchange of digital signatures. In: Proc. of Eurocrypt'98. Lect Notes in Comput Sci, Vol. 1403. Berlin: Springer-Verlag, 1998. 591–606

9 Asokan N, Shoup V, Waidner M. Optimistic fair exchange of signatures. IEEE J Select Areas Commun, 2000, 18: 593–610

10 Boyd C, Foo E. Off-line fair payment protocols using convertible signature. In: Proc. of Asiacrypt'98, Lect Notes in Comput Sci, Vol. 1514. Berlin: Springer-Verlag, 1998. 271–285

11 Bao F. Colluding attacks to a payment protocol and two signature exchange schemes. In: Proc. of Asiacrypt'04, Lect Notes in Comput Sci, Vol. 3329. Berlin: Springer-Verlag, 2004. 417–429

12 Bao F, Deng R H, Mao W. Efficient and practical fair exchange protocols with off-line TTP. In: Proc. of IEEE Symposium on Security and Privacy. Los Alamitos: IEEE Computer Society, 1998. 77–85

13 Boneh D, Gentry C, Lynn B, et al. Aggregrate and verifiably encrypted signatures from bilinear maps. In: Proc. of Eurocrypt'03, Lect Notes in Comput Sci, Vol. 2656. Berlin: Springer-Verlag, 2003. 416–432

14 Deng R H, Gong L, Lazar A A, et al. Practical protocols for certified electronic mail. J Netw Syst Manag, 1996, 4: 279–297

15 Dodis Y, Reyzin L. Breaking and repairing optimistic fair exchange from PODC 2003. In: Proc. of ACM Workshop on Digital Rights Management (DRM). New York: ACM Press, 2003. 47–54

16 Franklin M, Reiter M. Fair exchange with a semi-trusted third party. In: Proc. of 4th ACM Conference on Computer and Communications Security. New York: ACM Press, 1997. 1–6

17 Garay J, Jakobsson M, MacKenzie P. Abuse-free optimistic contract signing. In: Proc. of Crypto'99, Lect Notes in Comput Sci, Vol. 1666. Berlin: Springer-Verlag, 1999. 449–466

18 Park J M, Chong E, Siegel H, et al. Constructing fair-exchange protocols for e-commerce via distributed computation of RSA signatures. In: Proc. of the Twenty-Second ACM Symposium on Principles of Distributed Computing (PODC 2003). New York: ACM Press, 2003. 172–181

19 Zhou J, Gollmann D. A fair non-repudiation protocol. In: Proc. of IEEE Symposium on Security and Privacy. Los Alamitos: IEEE Computer Society, 1996. 55–61

20 Zhou J, Gollmann D. An efficient non-repudiation protocol. In: Proc. of 10th IEEE Computer Security Foundations Workshop. Los Alamitos: IEEE Computer Society, 1997. 126–132

21 Zhou J, Deng R, Bao F. Some remarks on a fair exchange protocol. In: Proc. of Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2000. Lect Notes in Comput Sci, Vol. 1751. London: Springer-Verlag, Australia, 2000. 46–57

22 Chen L, Kudla C, Paterson K G. Concurrent signature. In: Proc. of Eurocrypt'04. Lect Notes in Comput Sci, Vol. 3027. Berlin: Springer-Verlag , 2004. 287–305

23 Rivest R, Shamir A, Tauman Y. How to leak a secret. In: Proc. of Asiacrypt'01, Lect Notes in Comput Sci, Vol. 2248. Berlin: Springer-Verlag, 2001. 552–565

24 Abe M, Ohkubo M, Suzuki K. 1-out-of-n signatures from a variety of keys. In: Proc. of Asiacrypt'02, Lect Notes in Comput Sci, Vol. 2501. Berlin: Springer-Verlag, 2002. 415–432

25 Susilo W, Mu Y, Zhang F. Perfect concurrent signature schemes. In: Proc. of Information and Communications Security (ICICS'04), Lect Notes in Comput Sci, Vol. 3269. Berlin: Spriger-Verlag, 2004. 14–26

26 Wang G, Bao F, Zhou J. The fairness of perfect concurrent signatures. In: Proc. of Information and Communications Security (ICICS'06), Lect Notes in Comput Sci, Vol. 4307. Berlin: Spriger-Verlag, 2006. 435–451

27 Pointcheval D, Stern J. Security proofs for signature schemes. In: Proc. of Eurocrypt'96, Lect Notes in Comput Sci, Vol. 1070. Berlin: Springer-Verlag, 1996. 387–398

28 Pointcheval D, Stern J. Security arguments for digital signatures and blind signatures. J Cryp, 2000, 13: 361–396

29 Bellare M, Rogaway P. Random oracles are practical: a paradigm for designing efficient protocols. In: Proc. of 1st CCCS. New York: ACM Press, 1993. 62–73