An optimistic fair exchange protocol and its security in the universal composability framework

Yusuke Okada*

Department of Social Informatics,
Graduate School of Informatics,
Kyoto University,
Kyoto, Japan
E-mail: yokada@ai.soc.i.kyoto-u.ac.jp
*Corresponding author

Yoshifumi Manabe

NTT Communication Science Laboratories,
Nippon Telegraph and Telephone Corporation,
Department of Social Informatics,
Graduate School of Informatics,
Kyoto University,
Kyto, Japan
E-mail: manabe.yoshifumi@lab.ntt.co.jp

Tatsuaki Okamoto

NTT Information Sharing Platform Laboratories,
Nippon Telegraph and Telephone Corporation,
Department of Social Informatics,
Graduate School of Informatics,
Kyoto University,
Kyto, Japan
E-mail: okamoto.tatsuaki@lab.ntt.co.jp

Abstract: Fair exchange protocols allow both or neither of two parties to obtain the other's items, and this property is essential in e-commerce. In this paper, we construct an optimistic fair exchange protocol that is applicable to any digital signature by prescribing three forms of signatures, namely presignature, post-signature and notarised signature. We set an expiration date for presignature, and thus realise the timely termination of the protocol. Next, we define an ideal functionality of fair exchange protocols in the universal composability framework. Then, we construct an optimistic fair exchange protocol based on the above protocol, and prove its security in the universal composability framework.

Keywords: optimistic fair exchange protocols; digital signature; trusted third party; TTP; universal composition.

Reference to this paper should be made as follows: Okada, Y., Manabe, Y. and Okamoto T. (2008) 'An optimistic fair exchange protocol and its security in the universal composability framework', *Int. J. of Applied Cryptography*, Vol. 1, No. 1, pp.70–77.

Biographical notes: Yusuke Okada received an ME from Kyoto University, Kyoto, Japan, in 2007. His research interests are cryptography and information security. His current affiliation is KDDI Corporation, Japan.

Yoshifumi Manabe received BE, ME and Dr.E. from Osaka University, Osaka, Japan, in 1983, 1985 and 1993, respectively. In 1985, he joined Nippon Telegraph and Telephone Corporation. Currently, he is a Senior Research Scientist, Supervisor of NTT Communication Science Laboratories. His research interests include distributed algorithms and cryptography. He has been a Guest Associate Professor at Kyoto University since 2001.

Tatsuaki Okamoto received BE, ME and Dr.E. from the University of Tokyo, Tokyo, Japan, in 1976, 1978 and 1988, respectively. He is a Fellow of NTT Information Sharing Platform Laboratories. He is presently engaged in research on cryptography and information security. He is the Director of the Japan Society for Industrial and Applied Mathematics and a Guest Professor at Kyoto University.

1 Introduction

Fair exchange is an essential property in e-commerce, and various protocols have been proposed for realising fair exchange including gradual secret exchange (Even et al., 1985; Okamoto and Ohta, 1994), non-repudiation (Zhou and Gollmann, 1996, 1997) and optimistic fair exchange. Optimistic fair exchange protocols allow both or neither of two involved parties to obtain the other's items, where a Trusted Third Party (TTP) is not invoked when the two involved parties perform the protocol correctly. This kind of protocol is more practical than those in which TTP mediates all transactions. Many approaches have been employed to realise this kind of protocol (Asokan et al., 2000; Ateniese, 1999; Bao et al., 1998; Dodis and Reyzin, 2003; Park et al., 2003). Optimistic fair exchange protocols can be categorised by the data to be exchanged such as the exchange of digital signatures on two different messages, the exchange of digital signatures on the same message, and the exchange of a digital signature and digital data. Here we consider protocols that exchange a digital signature and digital data.

In this paper, we construct an optimistic fair exchange protocol that is applicable to any digital signature scheme such as RSA or DSA by prescribing the form of signatures, and prove the security of optimistic fair exchange protocols in the universal composability framework, which was proposed by Canetti (2001). This framework provides a unified methodology for proving the security of various protocols. Furthermore, in the universal composability framework, it is guaranteed that a secure primitive maintains its security even if other primitives run concurrently. Since optimistic fair exchange protocols use many primitives such as digital signatures, secure channels and certificate authorities, this property is very helpful. Our optimistic fair exchange protocol can employ any secure digital signature, so it is easy to handle within the universal composability framework by using the hybrid protocol.

2 Preliminaries

2.1 Optimistic fair exchange protocols

Asokan et al. (2000) proposed an optimistic fair exchange protocol that uses *verifiable escrow*. To use TTP as an escrow service, a signer encrypts his/her signature under the public key of TTP. Verifiable escrow is an encryption scheme with an attached decryption policy that represents the conditions under which the encryption will be decrypted by TTP. First, the signer reduces his/her signature to a certain homomorphic preimage of the signature. The signer then verifiably escrows the homomorphic preimage using a cut-and-choose interactive zero-knowledge proof. This scheme is applicable to any signature as long as the signature scheme can be reduced to a certain homomorphic preimage of the signature. They introduced homomorphic presignatures for RSA, DSA, Schnorr, Fiat-Shamir signatures among others. The drawback of this protocol is that it is highly interactive and needs a large amount of computation.

Bao et al. (1998) proposed a fair exchange protocol with off-line TTP that uses Certificate of Encrypted Message Being a Signature (CEMBS). In this protocol, parties sign their messages (such as a contract) and encrypt their signatures. CEMBS is used to convince parties that an encrypted signature is a certain party's signature on a message without revealing the signature itself. To realise this property, CEMBS uses proof-of-knowledge techniques and has to utilise the combination of a particular public key cryptosystem and a digital signature scheme ((Bao et al., 1998) used ElGamal and DSA or ElGamal and Gullou-Quisquater). This *ad hoc* technique is not a desirable property. Boneh et al. (2003) recently proposed a new verifiably encrypted signature scheme based on the GDH signature of Boneh et al. (2001). This scheme is completely non-interactive.

Park et al. (2003) introduced an optimistic fair exchange protocol that uses the two-party multisignature scheme as a primitive element. We use the term *two-signature* to represent a two-party multisignature quoting from Dodis and Reyzin (2003). Park et al. (2003) composed a two-signature scheme based on RSA signature, but Dodis and Reyzin (2003) broke this scheme. Recently, Boldyreva (2003) proposed a non-interactive multisignature scheme based on the GDH signature of Boneh et al. (2001). Dodis and Reyzin (2003) introduced an optimistic fair exchange protocol by utilising the non-interactive multisignature of Boldyreva. The protocol of Dodis and Reyzin (2003) has two drawbacks. First, TTP has to safely store as many secret arbitration keys as the number of users. Next, it requires special elliptic curve groups with a bilinear map and a two-signature scheme.

2.2 Universal composability framework

The universal composability framework, proposed by Canetti (2001), is a general framework for analysing the security of cryptographic protocols. In this framework, the security of protocols is defined by comparing the executions of two protocols, a real process and an ideal process.

In the real process, a multiparty protocol is executed in a given environment in the presence of an adversary that controls the communication between the parties and can corrupt the parties. In the ideal process, there is an ideal functionality that captures the desired functionality for carrying out the task and performs as a subroutine of multiple parties. Parties in the ideal protocol, called dummy parties, forward input from the environment to the ideal functionality and send back reply directly.

The environment, which represents all the other protocols running in the system, passes input to and obtains output from the parties and the adversary, and finally outputs a single bit attempt to distinguish with which protocol is interacting. A protocol π is said to UC-realise an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ there is an ideal process adversary $\mathcal{S}$ (we often call the adversary $\mathcal{S}$ a simulator) such that no environment $\mathcal{Z}$ can tell whether it is interacting with π and $\mathcal{A}$ or with $\text{IDEAL}_{\mathcal{F}}$, which is the ideal protocol for $\mathcal{F}$ and $\mathcal{S}$.

We use the following notation defined in Canetti (2001). Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ represent $\mathcal{Z}$'s output after it has interacted with π and $\mathcal{A}$, given the security parameter

k and input z. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ represent the ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in N,z\in\{0,1\}^*}$.

3 Fair exchange functionality

First, we define the ideal functionality of fair exchange protocols. Fair exchange is a task where two parties interact such that either party obtains the other's item or neither does. In other words, no dishonest party can obtain the honest party's item without the honest party obtaining the dishonest party's item. The fair exchange functionality, $\mathcal{F}_{\mathrm{FE}}^{(\mathrm{prop}_A,\ \mathrm{prop}_B,\ \mathrm{verify})}$ is shown in Figure 1. Here, we consider the case where parties A and B exchange a digital signature on M_A for digital data M_B.

Two functions $\mathrm{prop}_A : \{0,1\}^* \to \{0,1\}$ and $\mathrm{prop}_B : \{0,1\}^* \to \{0,1\}$ capture the verification of M_A and M_B, respectively. The function $\mathrm{verify}(\cdot)$ captures the signature verification function. Since this function may depend on the composition of the protocol, we will describe the definition of $\mathrm{verify}(\cdot)$ in Section 4.3.

The functionality shown in Figure 1 captures the *fair exchange* task, not just the *optimistic* task. Party T (this party, representing TTP, appears in the real protocol) does not appear explicitly in the ideal protocol for this functionality. Instead, the functionality itself plays the role of TTP. This difference between the ideal and real protocol poses no problem because there is no input or subroutine output from $\mathcal{Z}$ to T and back in either protocol.

4 Optimistic fair exchange protocol

Here we describe an optimistic fair exchange protocol that is applicable to any digital signature scheme such as RSA or DSA by prescribing the form of signatures. The parties involved in the protocol are Alice (customer), Bob (merchant) and TTP. In this paper, we consider exchange protocols where two involved parties exchange a digital signature for digital data. For example, Alice purchases digital data (e.g. music files, license keys) from Bob in exchange for her digital signature on the purchase contract. We do not consider digital data that allows Alice to obtain a benefit if she obtains multiple copies of the same digital data. This assumption is required when the dispute resolution protocol is invoked.

4.1 Definition of presignature, post-signature, and notarised signature

First, we define three types of signatures, *presignature*, *post-signature* and *notarised signature*, by prescribing the form of the signatures. We assume that the signature scheme consists of the triple of algorithm (KeyGen, Sign, Verify). We also assume that Alice and TTP have already generated their secret and public key pairs by executing KeyGen in the setup phase, and used PKI to certify their public keys. Presignature, post-signature and notarised signature are defined as follows.

Presignature: Alice's presignature is of the form

$$\sigma_{\mathrm{pre}} = \mathrm{Sign}_A(M_A, \mathrm{cert}_A, \mathrm{cert}_{\mathrm{TTP}}, t)$$

Post-signature: Alice's post-signature is of the form

$$\sigma_{\mathrm{post}} = \mathrm{Sign}_A(M_A, \mathrm{cert}_A)$$

Notarised signature: the notarised signature by TTP is of the form

$$\sigma_{\mathrm{TTP}} = \mathrm{Sign}_{\mathrm{TTP}}(\sigma_{\mathrm{pre}})$$

The term M_A represents the purchase contract. cert_A and $\mathrm{cert}_{\mathrm{TTP}}$ indicate the certificates of Alice and TTP, respectively, and parameter t is the expiration date of the presignature. We introduce this parameter to realise the *timely termination* of the protocol. The presignature is Alice's signature on the concatenation of the purchase contract, Alice's public key certificate, TTP's public key certificate and the parameter that represents the expiration date of the presignature. The post-signature is Alice's signature on the concatenation of the purchase contract and Alice's public key certificate. The notarised signature is TTP's signature on Alice's presignature. We define both the post-signature and notarised signature as legally valid signatures. In addition, even if Bob shows both σ_{pre} and σ_{TTP}, these are regarded as one legally valid signature. On the other hand, the presignature is defined as a legally invalid signature. TTP has the power to transform Alice's presignature into a notarised signature that has the same legal value as a post-signature.

Figure 1 The fair exchange functionality, $\mathcal{F}_{\mathrm{FE}}^{(\mathrm{prop}_A,\ \mathrm{prop}_B,\ \mathrm{verify})}$

Functionality $\mathcal{F}_{\mathrm{FE}}^{(\mathrm{prop}_A,\ \mathrm{prop}_B,\ \mathrm{verify})}$

1. Upon receiving input (**Initiate**, sid, M_A) from party A, verify that $\mathrm{sid} = (A, B, \mathrm{sid}')$ for some party B and $\mathrm{prop}_A(M_A) = 1$. If not, ignore this input. Else, send (**Initiate**, sid, M_A) to the adversary. Upon receiving **ok** from the adversary, record the entry (A, B, M_A) and send output (**Initiated**, sid) to B.
2. Upon receiving input (**Send**, sid, M_B) from B, verify that there is an entry (A, B, M_A) and $\mathrm{prop}_B(M_B) = 1$. If not, ignore this input. Else, send (**Sent**, sid, $|M_B|$) to the adversary. Upon receiving (**Signature**, sid, M_A, σ_A) from the adversary, check whether $\mathrm{verify}(M_A, \sigma_A) = 1$. If not, ignore the input. Else, record the entries (A, B, M_A, σ_A) and (B, A, M_B) and send output (**Sent**, sid, M_B) to A.
3. Upon receiving (**Get**, sid) from B, verify that there exist two entries (A, B, M_A, σ_A) and (B, A, M_B). If not, ignore this input. Else, send (**Get**, sid) to the adversary. Upon receiving **ok** from the adversary, and send output (**Sent**, sid, M_A, σ_A) to B.

4.2 Description of optimistic fair exchange protocol

We now construct an optimistic fair exchange protocol by using these signatures. It consists of two protocols, the main protocol and a dispute resolution protocol. In the protocols, we assume that data transactions are executed over secure channels established using techniques such as SSL. Alice initiates the main protocol with Bob. The main protocol is as described below.

Main protocol:

1 Alice sends her presignature to Bob.

2 Bob verifies the presignature and its expiration date. If either one is invalid, Bob aborts the protocol. Else, Bob sends his digital data to Alice.

3 Alice verifies the digital data. If it is invalid, she aborts the protocol. Else, Alice sends her post-signature to Bob.

4 Bob verifies the post-signature. If it is invalid or Bob does not receive it by the expiration date of Alice's presignature, then Bob invokes the dispute resolution protocol. Else, the exchange protocol ends correctly.

Then, we describe the dispute resolution protocol. Bob initiates the protocol with TTP.

Dispute resolution protocol:

1 Bob sends Alice's presignature to TTP along with his digital data.

2 TTP verifies the presignature, its expiration date, and the digital data. If any one of them is invalid, TTP aborts the protocol. Else, TTP sends the notarized signature to Bob.

3 TTP also forwards Bob's digital data to Alice.

Bob has to send his digital data in Step 1 of the dispute resolution protocol and TTP forwards Bob's digital data to Alice in Step 3 is to prevent malicious Bob from obtaining the notarised signature without sending his digital data to Alice. In the dispute resolution protocol, the TTP's verification of the digital data may constitute a bottleneck, because it is difficult to associate the verification of digital data with a certain mathematical algorithm. To verify the digital data efficiently in practice, we propose the use of hash tables. We assume that there is a hash table of digital data such as music files and online software. TTP generates a message digest h_B of digital data M_B by using hash functions, and verifies M_B by checking whether or not h_B is in the hash table.

Expiration date of presignatures: parameter t defines the expiration date of Alice's presignature. This parameter may be set by Alice or set as a system parameter. Bob rejects expired presignatures in the main protocol, and TTP will not transform presignatures into notarised signatures after the expiration date. This parameter realises the *timely termination* of the exchange protocol because both Alice and Bob will be at a disadvantage if they do not terminate the protocol by the expiration date.

Disadvantage for Alice: if Alice sets an unfavourable t (e.g. too short or past date), Bob will not send M_B. Thus, Alice cannot obtain any advantage from this.

Disadvantage for Bob: if Bob sends his digital data after the expiration date of Alice's presignature and Alice does not send her postsignature, Bob cannot obtain Alice's valid signature by using the dispute resolution protocol because TTP does not transform presignatures after the expiration date. Thus, parameter t constrains Bob to send M_B by the expiration date. If Bob sends M_B and Alice does not send her post-signature until the expiration date, Bob invokes the dispute resolution protocol and has Alice's presignature transformed into a notarised signature by the expiration date.

For practical purposes, we should prearrange when Bob invokes the dispute resolution protocol in cases where Alice does not send her post-signature. That is, for example, Bob will be able to have presignatures transformed into notarised signatures between the expiration date and the following day. After waiting for Alice's post-signature until the expiration date, Bob invokes the dispute resolution protocol during this period and obtains the notarised signature.

4.3 Protocol π_{OFE} in the $(\mathcal{F}_{\mathrm{SIG}}, \mathcal{F}_{\mathrm{REG}}, \mathcal{F}_{\mathrm{SCS}})$-hybrid model

Next, we construct a hybrid protocol based on the protocol mentioned above, slightly modifying it to make it easier to handle within the universal composability framework. Here, we present a hybrid protocol for realising $\mathcal{F}_{\mathrm{FE}}^{(\mathrm{prop}_A,\ \mathrm{prop}_B,\ \mathrm{verify})}$, given the ideal functionalities $\mathcal{F}_{\mathrm{SIG}}$, $\mathcal{F}_{\mathrm{REG}}$, and $\mathcal{F}_{\mathrm{SCS}}$. The protocol π_{OFE} is shown in Figure 2. The underlined parts in the figure represent parties' outputs to the environment $\mathcal{Z}$. Party T represents a TTP, which is guaranteed not to be corrupted by the adversary.

We use the term m_{pre} as the concatenation of (M_A, A, T) and m_{post} as the concatenation of (M_A, A), where A and T represent the respective IDs. Presignature σ_{pre} and post-signature σ_{post} represent A's signature on m_{pre} and m_{post}, respectively. Notarised signature σ_{TTP} represents T's signature on σ_{pre}. σ_{post} and σ_{TTP} are defined as legally valid signatures, so Bob expects to receive either σ_{post} or σ_{TTP}. Thus, in protocol π_{OFE}, verify$(\cdot)$ in $\mathcal{F}_{\mathrm{FE}}^{(\mathrm{prop}_A,\ \mathrm{prop}_B,\ \mathrm{verify})}$ is defined as the function that returns 1 iff $v_A(m_{\mathrm{post}}, \sigma_{\mathrm{post}}) = 1 \vee (v_A(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}}) = 1 \wedge v_{\mathrm{TTP}}(\sigma_{\mathrm{pre}}, \sigma_{\mathrm{TTP}}) = 1)$.

In Figure 2, Step 2(e) corresponds to the resolution protocol. When neither A nor B is corrupted, party A correctly outputs (**Sent**, sid, M_B) in Step 2(d) and goes to Step 3, because all messages between A and B are sent and received by using $\mathcal{F}_{\mathrm{SCS}}$. There are two cases in which the resolution protocol is executed. One is where party A is corrupted by the adversary and instructed to send an invalid σ'_{post}. The other is where party B is corrupted and instructed to send an invalid M'_B. In this case, A enters the waiting state and goes to Step 2(e). The adversary can instruct corrupted B to send a resolve message to T.

This hybrid protocol uses three ideal functionalities: $\mathcal{F}_{\mathrm{SIG}}$, $\mathcal{F}_{\mathrm{REG}}$ and $\mathcal{F}_{\mathrm{SCS}}$. We show the ideal functionalities $\mathcal{F}_{\mathrm{SIG}}$, $\mathcal{F}_{\mathrm{REG}}$ and $\mathcal{F}_{\mathrm{SCS}}$ defined by Canetti (2001) in Figures 3–5, respectively. We slightly modify $\mathcal{F}_{\mathrm{REG}}$ from the original one in Canetti (2001). The modified registration functionality sends output (**Registered**, sid, v) to the party in order to specify clearly the activation of the key registering party.

Figure 2 Protocol π_{OFE} in the $(\mathcal{F}_{\mathrm{SIG}}, \mathcal{F}_{\mathrm{REG}}, \mathcal{F}_{\mathrm{SCS}})$-hybrid model

Protocol π_{OFE} in the $(\mathcal{F}_{\mathrm{SIG}}, \mathcal{F}_{\mathrm{REG}}, \mathcal{F}_{\mathrm{SCS}})$-hybrid model

1. When activated with input (**Initiate**, sid, M_A), do:
   (a) A verifies that sid $= (A, B, \mathrm{sid}')$ for some party B. If not, ignore the input. Else, it sends the message (**KeyGen**, sid_A) to $\mathcal{F}_{\mathrm{SIG}}$ where $\mathrm{sid}_A = (A, \mathrm{sid})$, and obtains (**Verification Algorithm**, sid_A, v_A). Next, A sends (**Register**, sid_A, v_A) to $\mathcal{F}_{\mathrm{REG}}$, and obtains (**Registered**, sid_A, v_A).
   (b) A sends the message (**Sign**, sid_A, m_{pre}) to $\mathcal{F}_{\mathrm{SIG}}$ where $m_{\mathrm{pre}} = (M_A, A, T)$ and obtains (**Signature**, sid_A, m_{pre}, σ_{pre}). A then sends $(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}})$ to B by using $\mathcal{F}_{\mathrm{SCS}}$.
   (c) Upon receiving $(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}})$, B verifies that $\mathrm{prop}_A(M_A) = 1$. If not, B halts. Else, B sends (**Retrieve**, sid_A) to $\mathcal{F}_{\mathrm{REG}}$, and obtains (**Retrieve**, sid_A, v_A) from $\mathcal{F}_{\mathrm{REG}}$.
   (d) B sends (**Verify**, sid_A, m_{pre}, σ_{pre}, v_A) to $\mathcal{F}_{\mathrm{SIG}}$, and obtains (**Verified**, sid_A, m_{pre}, $v_A(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}})$). If $v_A(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}}) = 1$, B outputs (**Initiated**, sid). Else, B halts.
2. When activated with input (**Send**, sid, M_B), do:
   (a) If $v_A(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}}) = 1$, B sends M_B to A by using $\mathcal{F}_{\mathrm{SCS}}$. Else, it halts.
   (b) Upon receiving M_B, A verifies that $\mathrm{prop}_B(M_B) = 1$. If not, go to step(e). Else, A sends the message (**Sign**, sid_A, m_{post}) to $\mathcal{F}_{\mathrm{SIG}}$ where $m_{\mathrm{post}} = (M_A, A)$, and obtains (**Signature**, sid_A, m_{post}, σ_{post}) from $\mathcal{F}_{\mathrm{SIG}}$.
   (c) A sends $(m_{\mathrm{post}}, \sigma_{\mathrm{post}})$ to B by using $\mathcal{F}_{\mathrm{SCS}}$.
   (d) Upon receiving $(m_{\mathrm{post}}, \sigma_{\mathrm{post}})$, B sends (**Verify**, sid_A, m_{post}, σ_{post}, v_A) to $\mathcal{F}_{\mathrm{SIG}}$, and obtains (**Verified**, sid_A, m_{post}, $v_A(m_{\mathrm{post}}, \sigma_{\mathrm{post}})$). If $v_A(m_{\mathrm{post}}, \sigma_{\mathrm{post}}) \neq 1$, go to step(e). Else, B sends (**Verified**, sid) to A by using $\mathcal{F}_{\mathrm{SCS}}$, and A outputs (**Sent**, sid, M_B).
   (e) B sends (**Resolve**, sid, $(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}})$, M_B) to T by using $\mathcal{F}_{\mathrm{SCS}}$,
      (i.) Upon receiving (**Resolve**, sid, $(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}})$, M_B), T sends (**Retrieve**, sid_A) to $\mathcal{F}_{\mathrm{REG}}$, and obtains (**Retrieve**, sid_A, v_A). It then sends (**Verify**, sid_A, m_{pre}, σ_{pre}, v_A) to $\mathcal{F}_{\mathrm{SIG}}$.
      (ii.) Upon receiving (**Verified**, sid_A, m_{pre}, $v_A(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}})$) from $\mathcal{F}_{\mathrm{SIG}}$, T verifies that $v_A(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}}) = 1$ and $\mathrm{prop}_B(M_B) = 1$. If not, it halts. Else, it sends the message (**KeyGen**sid_T) to $\mathcal{F}_{\mathrm{SIG}}$, where $\mathrm{sid}_T = (T, \mathrm{sid})$, and obtains (**Verification Algorithm**, sid_T, v_T). Next, it sends (**Register**, sid_T, v_T) to $\mathcal{F}_{\mathrm{REG}}$, and obtains (**Registered**, sid_T, v_T).
      (iii.) T sends (**Sign**, sid_T, σ_{pre}) to $\mathcal{F}_{\mathrm{SIG}}$, and obtains (**Signature**, sid_T, σ_{pre}, σ_{TTP}).
      (iv.) T sends M_B to A by using $\mathcal{F}_{\mathrm{SCS}}$.
      (v.) Upon receiving M_B, A outputs (**Sent**, sid, M_B).
3. When activated with an input (**Get**, sid), do:
   (a) If B has obtained $(m_{\mathrm{post}}, \sigma_{\mathrm{post}})$ where $v_A(m_{\mathrm{post}}, \sigma_{\mathrm{post}}) = 1$, it outputs (**Sent**, sid, M_A, σ_{post}).
   (b) Else, B sends (**Get**, sid) to T by using $\mathcal{F}_{\mathrm{SCS}}$. Upon receiving (**Get**, sid), T sends σ_{TTP} to B by using $\mathcal{F}_{\mathrm{SCS}}$. Upon receiving σ_{TTP}, B outputs (**Sent**, sid, M_A, $(\sigma_{\mathrm{pre}}, \sigma_{\mathrm{TTP}})$).

5 Security of protocol π_{OFE}

Theorem 1: *Protocol π_{OFE} UC-realises fair exchange functionality $\mathcal{F}_{\mathrm{FE}}^{(prop_A,\ prop_B,\ verify)}$ in the $(\mathcal{F}_{\mathrm{SIG}}, \mathcal{F}_{\mathrm{REG}}, \mathcal{F}_{\mathrm{SCS}})$-hybrid model.*

Proof: Let $\mathcal{S}_{\mathrm{HYB}}$ be a hybrid protocol simulator that interacts with parties running π_{OFE} in the $(\mathcal{F}_{\mathrm{SIG}}, \mathcal{F}_{\mathrm{REG}}, \mathcal{F}_{\mathrm{SCS}})$-hybrid model. We now construct a simulator $\mathcal{S}$ such that the view of the environment $\mathcal{Z}$ when interacting with $\mathcal{S}_{\mathrm{HYB}}$ and π_{OFE} has the same distribution as $\mathcal{Z}$ when interacting with $\mathcal{S}$ and the ideal protocol for $\mathcal{F}_{\mathrm{FE}}$. That is, for any $\mathcal{S}_{\mathrm{HYB}}$ there exists $\mathcal{S}$ such that $\mathrm{EXEC}_{\pi_{\mathrm{OFE}}, \mathcal{S}_{\mathrm{HYB}}, \mathcal{Z}} \approx \mathrm{EXEC}_{\mathrm{IDEAL}_{\mathcal{F}}, \mathcal{S}, \mathcal{Z}}$ for any environment $\mathcal{Z}$. $\mathcal{S}$ runs an internal copy of $\mathcal{S}_{\mathrm{HYB}}$ as a black box, forwards any input from $\mathcal{Z}$ to $\mathcal{S}_{\mathrm{HYB}}$ and vice versa. $\mathcal{S}$ also runs an internal copy of each of the involved parties, and simulates $\mathcal{F}_{\mathrm{SIG}}$, $\mathcal{F}_{\mathrm{REG}}$ and $\mathcal{F}_{\mathrm{SCS}}$. The behaviour of $\mathcal{S}$ is described as follows.

A case where no party is corrupted. When $\mathcal{S}$ receives (**Initiate**, sid, M_A) from $\mathcal{F}_{\mathrm{FE}}$, where sid $= (A, B, \mathrm{sid}')$, it proceeds as follows:

1 $\mathcal{S}$ simulates the processes of key generation and registration. It sends the message (**KeyGen**, sid_A) to $\mathcal{S}_{\mathrm{HYB}}$ (in the name of $\mathcal{F}_{\mathrm{SIG}}$), and obtains (**Verification Algorithm**, sid_A, s_A, v_A).

Figure 3 The signature functionality, $\mathcal{F}_{\text{SIG}}$

Functionality $\mathcal{F}_{\text{SIG}}$

Key Generation: Upon receiving a value (**KeyGen**, sid) from some party S, verify that sid $= (S, \text{sid}')$ for some sid$'$. If not, then ignore the request. Else, hand (**KeyGen**, sid) to the adversary. Upon receiving (**Algorithms**, sid, s, v) from the adversary, where s is a description of a PPT ITM, and v is a description of a deterministic polytime ITM, output (**Verification Algorithm**, sid, v) to S.

Signature generation: Upon receiving a value (**Sign**, sid, m) from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output (**Signature**, sid, m, σ) to S and record the entry (m, σ). Else, output an error message to S and halt.

Signature Verification: Upon receiving a value (**Verify**, sid, m, σ, v') from some party V, do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output (**Verified**, sid, m, $v'(m, \sigma)$) to V.

Figure 4 The registration functionality, $\mathcal{F}_{\text{REG}}$

Functionality $\mathcal{F}_{\text{REG}}$

1. Upon receiving input (**Register**, sid, v), verify that sid $= (P, \text{sid}')$. If sid$'$ is not of that form, or this is not the first input from P, then ignore this input. Else, send (**Registered**, sid, v) to the adversary and record the value v. Then, send (**Registered**, sid, v) to P.
2. Upon receiving input (**Retrieve**, sid) from party P', send a delayed output (**Retrieve**, sid, v) to P'. (If no value v is recorded, then set $v = \perp$.)

Figure 5 The secure communication session functionality, $\mathcal{F}_{\text{SCS}}$

Functionality $\mathcal{F}_{\text{SCS}}$

$\mathcal{F}_{\text{SCS}}$ proceeds as follows, when parameterised by the leakage function $l : \{0, 1\}^* \to \{0, 1\}^*$.

1. Upon receiving input (**Establish-Session**, sid) from party I, verify that sid $= (I, R, \text{sid}')$ for some R, record I as active, record R as the responder, and send a public delayed output (**Establish-Session**, sid) to R.
2. Upon receiving (**Establish-Session**, sid) from party R, verify that R is recorded as the responder, and record R as active.
3. Upon receiving input (**Send**, sid, m) from party $P \in \{I, R\}$, send (**Sent**, sid, P, $l(m)$) to the adversary. In addition, if P is active then send a private delayed output (**Sent**, sid, P, m) to the other party in $\{I, R\}$.

It then sends (**Verification Algorithm**, sid_A, v_A) to simulated A. Next, it sends (**Registered**, sid_A, v_A) to $\mathcal{S}_{\text{HYB}}$ and simulated A.

2 $\mathcal{S}$ simulates the processes of signature generation and the sending of $(m_{\text{pre}}, \sigma_{\text{pre}})$. It sends the message (**Establish-Session**, sid) to $\mathcal{S}_{\text{HYB}}$ (in the name of $\mathcal{F}_{\text{SCS}}$), obtains **ok** from $\mathcal{S}_{\text{HYB}}$ and sends (**Establish-Session**, sid) to simulated B. Next, it sends the message (**Sent**, sid, $|(m_{\text{pre}}, \sigma_{\text{pre}})|$) to $\mathcal{S}_{\text{HYB}}$. Upon receiving **ok** from $\mathcal{S}_{\text{HYB}}$, it sends (**Sent**, sid, $(m_{\text{pre}}, \sigma_{\text{pre}})$) to simulated B.

3 $\mathcal{S}$ simulates the processes of key retrieval and signature verification. It sends the message (**Retrieve**, sid_A, v_A) to $\mathcal{S}_{\text{HYB}}$ (in the name of $\mathcal{F}_{\text{REG}}$). Upon receiving **ok** from $\mathcal{S}_{\text{HYB}}$, it sends (**Retrieve**, sid_A, v_A) to simulated B. Then, $\mathcal{S}$ sends **ok** to $\mathcal{F}_{\text{FE}}$.

When $\mathcal{S}$ receives (**Send**, sid, $|M_B|$) from $\mathcal{F}_{\text{FE}}$, it proceeds as follows:

1 $\mathcal{S}$ simulates the process of sending M_B. It sends the message (**Sent**, sid, $|M_B|$) to $\mathcal{S}_{\text{HYB}}$ (in the name of $\mathcal{F}_{\text{SCS}}$), and receives **ok** from $\mathcal{S}_{\text{HYB}}$.

2 $\mathcal{S}$ simulates the processes of the signature generation and the sending of $(m_{\text{post}}, \sigma_{\text{post}})$. It sends the message (**Sent**, sid, $|(m_{\text{post}}, \sigma_{\text{post}})|$) to $\mathcal{S}_{\text{HYB}}$ (in the name of $\mathcal{F}_{\text{SCS}}$). Upon receiving **ok** from $\mathcal{S}_{\text{HYB}}$, it sends (**Sent**, sid, $(m_{\text{post}}, \sigma_{\text{post}})$) to simulated B.

3 $\mathcal{S}$ simulates the process of sending the verification message (**Verified**, sid). It sends the message |(**Verified**, sid)| to $\mathcal{S}_{\text{HYB}}$ (in the name of $\mathcal{F}_{\text{SCS}}$). Upon receiving **ok** from $\mathcal{S}_{\text{HYB}}$, it sends (**Signature**, sid, M_A, σ_{post}) to $\mathcal{F}_{\text{FE}}$.

When $\mathcal{S}$ receives (**Get**, sid) from $\mathcal{F}_{\text{FE}}$, $\mathcal{S}$ sends (**Send**, sid, $(m_{\text{post}}, \sigma_{\text{post}})$) to $\mathcal{F}_{\text{FE}}$, since there is no party corruption.

In this case, $\mathcal{S}$ can perform the simulation perfectly. That is, the view of the environment $\mathcal{Z}$ when interacting with $\mathcal{S}_{\text{HYB}}$ and π_{OFE} has the same distribution as that of $\mathcal{Z}$ when interacting with $\mathcal{S}$ and the ideal protocol for $\mathcal{F}_{\text{FE}}$.

Next, we construct $\mathcal{S}$ assuming party corruption. Since all messages are sent by using $\mathcal{F}_{\text{SCS}}$ in π_{OFE}, it is only necessary to consider the case where $\mathcal{S}_{\text{HYB}}$ instructs a corrupted party to send modified data to $\mathcal{F}_{\text{SCS}}$. Cases where $\mathcal{S}_{\text{HYB}}$ instructs a

corrupted party to register a modified key to $\mathcal{F}_{\mathrm{REG}}$ or instructs a corrupted party to sign a modified message by $\mathcal{F}_{\mathrm{SIG}}$ are similar to the case described above.

Simulating party corruption: to simulate party corruption, $\mathcal{S}$ has to simulate the current local state of the corrupted party. $\mathcal{S}$ knows the secret keys of the parties, so it can clearly provide simulated $\mathcal{S}_{\mathrm{HYB}}$ with the local state of the corrupted party except for M_B. When black box $\mathcal{S}_{\mathrm{HYB}}$ sends a corruption message to party A, $\mathcal{S}$ (simulating corrupted A) must send M_B to $\mathcal{S}_{\mathrm{HYB}}$ after simulating B's transmission of M_B.

A case where party A is corrupted: when $\mathcal{S}_{\mathrm{HYB}}$ instructs corrupted A to send (**Send**, sid, $(m'_{\mathrm{pre}}, \sigma'_{\mathrm{pre}})$) to $\mathcal{F}_{\mathrm{SCS}}$, $\mathcal{S}$ proceeds as follows:

1 $\mathcal{S}$ sends (**Sent**, sid, $|(m'_{\mathrm{pre}}, \sigma'_{\mathrm{pre}})|$) to $\mathcal{S}_{\mathrm{HYB}}$ in the name of $\mathcal{F}_{\mathrm{SCS}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends (**Sent**, sid, $(m'_{\mathrm{pre}}, \sigma'_{\mathrm{pre}})$) to simulated B in the name of $\mathcal{F}_{\mathrm{SCS}}$.

2 Next, $\mathcal{S}$ simulates the process of signature verification. $\mathcal{S}$ sends (**Verified**, sid_A, m'_{pre}, $v_A(m'_{\mathrm{pre}}, \sigma'_{\mathrm{pre}})$) to simulated B in the name of $\mathcal{F}_{\mathrm{SIG}}$. If $v_A(m'_{\mathrm{pre}}, \sigma'_{\mathrm{pre}}) = 1$, $\mathcal{S}$ sends **ok** to $\mathcal{F}_{\mathrm{FE}}$.

When $\mathcal{S}_{\mathrm{HYB}}$ instructs corrupted A to send (**Send**, sid, $(m'_{\mathrm{post}}, \sigma'_{\mathrm{post}})$) to $\mathcal{F}_{\mathrm{SCS}}$, simulated $\mathcal{S}$ proceeds as follows:

1 $\mathcal{S}$ sends (**Sent**, sid, $|(m'_{\mathrm{post}}, \sigma'_{\mathrm{post}})|$) to $\mathcal{S}_{\mathrm{HYB}}$ in the name of $\mathcal{F}_{\mathrm{SCS}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends (**Sent**, sid, $(m'_{\mathrm{post}}, \sigma'_{\mathrm{post}})$) to simulated B in the name of $\mathcal{F}_{\mathrm{SCS}}$.

2 Next, $\mathcal{S}$ simulates the process of signature verification. $\mathcal{S}$ sends (**Verified**, sid_A, m'_{post}, $v_A(m'_{\mathrm{post}}, \sigma'_{\mathrm{post}})$) to simulated B in the name of $\mathcal{F}_{\mathrm{SIG}}$. If $v_A(m'_{\mathrm{post}}, \sigma'_{\mathrm{post}}) = 1$, $\mathcal{S}$ simulates in the same way as when the parties are not corrupted.

3 Else, $\mathcal{S}$ simulates the process of the resolution phase. $\mathcal{S}$ sends (**Sent**, sid, $|((m_{\mathrm{pre}}, \sigma_{\mathrm{pre}}), M_B)|$) to $\mathcal{S}_{\mathrm{HYB}}$ in the name of $\mathcal{F}_{\mathrm{SCS}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends (**Sent**, sid, $((m_{\mathrm{pre}}, \sigma_{\mathrm{pre}}), M_B)$) to simulated T. Next, $\mathcal{S}$ simulates the processes of the signature generation of T. $\mathcal{S}$ then sends (**Sent**, sid, $|M_B|$) to $\mathcal{S}_{\mathrm{HYB}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends (**Sent**, sid, M_B) to corrupted A, and sends (**Signature**, sid, M_A, $(\sigma_{\mathrm{pre}}, \sigma_{\mathrm{TTP}})$) to $\mathcal{F}_{\mathrm{FE}}$.

4 Upon receiving (**Get**, sid) from $\mathcal{F}_{\mathrm{FE}}$, $\mathcal{S}$ simulates the process of B obtaining T's signature. $\mathcal{S}$ sends (**Sent**, sid, |(**Get**, sid)|) to $\mathcal{S}_{\mathrm{HYB}}$ in the name of $\mathcal{F}_{\mathrm{SCS}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends (**Sent**, sid, (**Get**, sid)) to T. Next, $\mathcal{S}$ sends |(**Signature**, sid_T, σ_{pre}, σ_{TTP})| to $\mathcal{S}_{\mathrm{HYB}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends **ok** to $\mathcal{F}_{\mathrm{FE}}$.

A case where party B is corrupted: when $\mathcal{S}_{\mathrm{HYB}}$ instructs corrupted B to send (**Send**, sid, M'_B) to $\mathcal{F}_{\mathrm{SCS}}$, $\mathcal{S}$ proceeds as follows:

1 $\mathcal{S}$ sends (**Send**, sid, $|M'_B|$) to $\mathcal{S}_{\mathrm{HYB}}$ in the name of $\mathcal{F}_{\mathrm{SCS}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends (**Send**, sid, M'_B) to simulated A. If $\mathrm{prop}_B(M'_B) = 1$, $\mathcal{S}$ simulates in the same way as when parties are not corrupted.

2 Else, if $\mathcal{S}_{\mathrm{HYB}}$ instructs corrupted B to send (**Resolve**, sid, $(m_{\mathrm{pre}}, \sigma_{\mathrm{pre}})$, M_B) to $\mathcal{F}_{\mathrm{SCS}}$, $\mathcal{S}$ simulates the process of resolution phase. Simulated A finally receives (**Sent**, sid, M_B), and $\mathcal{S}$ then sends (**Signature**, sid, M_A, $(\sigma_{\mathrm{pre}}, \sigma_{\mathrm{TTP}})$) to $\mathcal{F}_{\mathrm{FE}}$.

3 When corrupted B sends (**Get**, sid) to $\mathcal{F}_{\mathrm{SCS}}$, $\mathcal{S}$ simulates the process of B obtaining T's signature. $\mathcal{S}$ finally sends (**Signature**, sid_T, σ_{pre}, σ_{TTP}) to simulated B, and sends **ok** to $\mathcal{F}_{\mathrm{FE}}$.

6 Conclusion

In this paper, we constructed an optimistic fair exchange protocol that is applicable to any digital signature by prescribing three forms of signatures, namely presignature, post-signature and notarised signature. We set the expiration date for the presignature, and thus realised the timely termination of the protocol.

Next, we defined the fair exchange functionality $\mathcal{F}_{\mathrm{FE}}^{(\mathrm{prop}_A,\ \mathrm{prop}_B,\ \mathrm{verify})}$ in the universal composability framework, and constructed an optimistic fair exchange protocol that UC-realises the fair exchange functionality in the $(\mathcal{F}_{\mathrm{SIG}}, \mathcal{F}_{\mathrm{REG}}, \mathcal{F}_{\mathrm{SCS}})$-hybrid model by slightly modifying the protocol mentioned above.

References

Asokan, N., Shoup, V. and Waidner, M. (2000) 'Optimistic fair exchange of digital signatures', *IEEE Journal on Selected Areas in Communication*, Vol. 18, No. 4, pp.593–610.

Ateniese, G. (1999) 'Efficient verifiable encryption (and fair exchange) of digital signatures', *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pp.138–146.

Bao, F., Deng, R. and Mao, W. (1998) 'Efficient and practical fair exchange protocols with off-line TTP', *Proceedings of the IEEE Symposium on Security and Privacy*, pp.77–85.

Boldyreva, A. (2003) 'Efficient threshold signature, multisignature and blind signature schemes based on the Gap-Diffie-Hellman-group signature scheme', *Proceedings of Practice and Theory in Public Key Cryptsystems - PKC 2003*, Vol. 2567 of *Lecture Notes in Computer Science*, pp.31–46.

Boneh, D., Gentry, C., Lynn, B. and Shacham, H. (2003) 'Aggregate and verifiably encrypted signatures from bilinear maps', *Advances in Cryptology - EUROCRYPT 2003*, Vol. 2656 of *Lecture Notes in Computer Science*, pp.416–432.

Boneh, D., Lynn, B. and Shacham, H. (2001) 'Short signatures from weil pairing', *Advances in Cryptology - ASIACRYPT 2001*, Vol. 2248 of *Lecture Notes in Computer Science*, pp.514–532.

Canetti, R. (2001) 'Universally composable security: a new paradigm for cryptographic protocols', *Proceedings of the 42nd Foundations of Computer Science Conference*, pp.136–145, Full version at http://eprint.iacr.org/2000/067/.

Canetti, R. (2004) 'Universally composable signature, certification, and authentication', *17th Computer Security Foundations Workshop*, pp.219–235, Available at: http://eprint.iacr.org/2001.

Canetti, R. and Krawczyk, H. (2002) 'Universally composable notions of key exchange and secure channels', *Advances in Cryptology - EUROCRYPT 2002*, Vol. 2332 of *Lecture Notes in Computer Science*, pp.337–351.

Dodis, Y. and Reyzin, L. (2003) 'Breaking and repairing optimistic fair exchange from PODC 2003', *Proceedings of the 2003 ACM Workshop on Digital Rights Management*, pp.47–54.

Even, S., Goldreich, O. and Lempel, A. (1985) 'A randomized protocol for signing contracts', *Communications of the ACM*, Vol. 28, No. 6, pp.637–647.

Okamoto, T. and Ohta, K. (1994) 'How to simultaneously exchange secrets by general assumptions', *Proceedings of 2nd ACM Conference on Computer and Communications Security*, pp.184–192.

Park, J.M., Chong, E. and Siegel, H.J. (2003) 'Constructing fair-exchange protocols for E-commerce via distributed computation of RSA signatures', *Proceedings of the 22nd Symposium on Principles of Distributed Computing*, pp.172–181.

Zhou, J. and Gollmann, D. (1996) 'A fair non-repudiation protocol', *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pp.55–61.

Zhou, J. and Gollmann, D. (1997) 'An efficient non-repudiation protocol', *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pp.126–132.