SCIS 2007 The 2007 Symposium on Cryptography and Information Security
Sasebo, Japan, Jan. 23-26, 2007
The Institute of Electronics, Information and Communication Engineers

An Optimistic Fair Exchange Protocol and its Security in the Universal Composability Framework

Yusuke Okada * Yoshifumi Manabe † Tatsuaki Okamoto †

Abstract— Fair exchange protocols allow two parties to either each party gets the other's item or neither party does, and this property is essential in e-commerce. In this paper, we define the ideal functionality of the fair exchange protocol in the universal composability framework, and present an optimistic fair exchange protocol that is secure in this framework.

Keywords: optimistic fair exchange protocols, digital signatures, universal composition.

1 Introduction

Fair exchange is an essential property in e-commerce, and various protocols have been proposed to realize fair exchange such as gradual secret exchange [10, 12], non-repudiation [14, 15], and optimistic fair exchange. Optimistic fair exchange protocols allow two involved parties to either each party get the other's item or neither party does where a Trusted Third Party (TTP) is not invoked when two involved parties perform the protocol correctly. This kind of protocol is more practical than the protocols in which TTP mediates all transactions. Many approaches have been taken to realize this kind of protocol [1, 2, 3, 9, 11, 13].

In this paper, we attempt to prove the security of optimistic fair exchange protocols in the universal composability framework, which was proposed by Canetti [5]. 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 quite helpful. The optimistic fair exchange protocol proposed in [11] can use any secure digital signature, so it is easy to handle within the universal composability framework by using the hybrid protocol.

2 Preliminaries

2.1 The universal composability framework

The universal composability, proposed by Canetti [5], is a general framework for analyzing the security of cryptographic protocols. In this framework, the security of protocols is defined via comparing the process of the execution of two protocols, a real process and an ideal process.

In the real process, the multi-party protocol is executed in a given environment in the presence of the adversary that controls the communication among the parties and can corrupt the parties. In the ideal process, there exists the ideal functionality that captures the desired functionality for carrying out the task and performs as a subroutine of multiple parties. The parties in the ideal protocol, called dummy parties , forward input from the environment to the ideal functionality and back directly.

The environment, representing all the other protocols running in the system, passes input to and obtains output from the parties and the adversary, outputs a single bit finally, attempting to distinguish with which protocol it interacts. A protocol π is said to UC-realize an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists 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 $\text{IDEAL}_{\mathcal{F}}$, ideal protocol for $\mathcal{F}$ and $\mathcal{S}$.

We use the following notation defined in [5]. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ represent the $\mathcal{Z}$'s output after interacting with π and $\mathcal{A}$, given the security parameter k and input z. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ represent the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in N,z\in\{0,1\}^*}$.

2.2 Optimistic fair exchange protocols

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 proposed a protocol in [11] that realizes optimistic fair exchange in such cases. We describe here a brief overview of the optimistic fair exchange protocol. First, we define three types of signatures, *pre-signature*, *post-signature*, and *notarized signature*, by prescribing the form of the signatures. Pre-signature is Alice's sig-

* Department of Social Informatics, Graduate School of Informatics, Kyoto University, Yoshidahonmachi, Sakyo-ku, Kyoto-shi, Japan. E-mail: yokada@ai.soc.i.kyoto-u.ac.jp
† NTT Laboratories, NTT Corporation, 1-1 Hikarinooka, Yokosuka-shi, Japan. E-mail: {manabe.yoshifumi, okamoto.tatsuaki}@lab.ntt.co.jp

nature on the concatenation of the purchase contract, Alice's public key certificate, TTP's public key certificate and the parameter that represent the expiration date of the pre-signature. Post-signature is Alice's signature on the concatenation of the purchase contract and Alice's public key certificate. Notarized signature is TTP's signature on Alice's pre-signature.

We define both post-signature and notarized signature as legally valid signatures. On the other hand, the pre-signature is defined as a legally invalid signature. TTP has the power to transform Alice's pre-signature into a notarized signature that has the same legal value as a post-signature.

In the protocol, we assume the data transactions are executed over secure channels. The protocol is as described below.

The main protocol

1. Alice sends her pre-signature to Bob.
2. Bob verifies the pre-signature and its expiration date. If invalid, Bob aborts the protocol. Else, Bob sends his digital data to Alice.
3. Alice verifies it. If invalid, she aborts the protocol. Else, Alice sends her post-signature to Bob.
4. Bob verifies the post-signature. If invalid or Bob doesn't get it by the expiration date of her pre-signature, then Bob invokes the dispute resolution protocol. Else, the exchange protocol ends correctly.

The dispute resolution protocol

1. Bob initiates the protocol. He sends Alice's pre-signature to TTP along with his digital data.
2. TTP verifies them. If either one of them is invalid, TTP aborts the protocol. Else, TTP sends the notarized signature to Bob, and Bob's digital data to Alice.

In this paper, we construct a hybrid protocol based on this protocol, slightly modifying it to make it easier to handle within the universal composability framework.

3 The fair exchange functionality

To prove the security of the protocol in the universal composability framework, we must define the ideal functionality of the protocol first. Fair exchange is a task where two parties interact such that either gets the other's item or neither does. Here, we consider the case that parties A and B exchange a digital signature on M_A for digital data M_B.

The fair exchange functionality, $\mathcal{F}_{\mathrm{FE}}^{(prop_A,\ prop_B,\ verify)}$ is shown in Figure 1. Two functions $prop_A : \{0,1\}^* \to \{0,1\}$ and $prop_B : \{0,1\}^* \to \{0,1\}$ capture the verification of M_A and M_B, respectively. The function $verify(\cdot)$ captures the signature verification function. Since this function may depend on the composition of the protocol, we will describe the definition of $verify(\cdot)$ in Section 4.

The functionality shown in Figure 1 captures the *fair exchange* task, not just the *optimistic* one. 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 and subroutine output from $\mathcal{Z}$ to T and back in either protocol.

Functionality $\mathcal{F}_{\mathrm{FE}}^{(prop_A,\ prop_B,\ verify)}$

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

Figure 1: The fair exchange functionality, $\mathcal{F}_{\mathrm{FE}}^{(prop_A,\ prop_B,\ verify)}$.

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

In this section, we present a hybrid protocol for realizing $\mathcal{F}_{\mathrm{FE}}^{(prop_A,\ prop_B,\ verify)}$, 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. Party T represents a trusted third party, which is guaranteed not to be corrupted by the adversary.

In this protocol, we use three types of signatures: σ_{pre}, σ_{post} and σ_{TTP}. We define 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. σ_{pre} and σ_{post} represent A's signature on m_{pre} and m_{post}, respectively. σ_{TTP} represents T's signature on σ_{pre}. Here, we define both σ_{post} and σ_{TTP} as legally valid signatures, so Bob expects to receive either σ_{post} or σ_{TTP}. In protocol π_{OFE}, $verify(\cdot)$ in

$\mathcal{F}_{\text{FE}}^{(prop_A,\ prop_B,\ verify)}$ is defined as the function that returns 1 iff $v_A(m_{post}, \sigma_{post}) = 1 \vee (v_A(m_{pre}, \sigma_{pre}) = 1 \wedge v_{TTP}(\sigma_{pre}, \sigma_{TTP}) = 1)$.

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 $\mathcal{F}_{\text{SCS}}$. There are two cases in which the resolution protocol is executed. One is the case where party A is corrupted by the adversary and instructed to send invalid σ'_{post}. The other is the case where party B is corrupted and instructed to send 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 resolve message to T.

The ideal functionalities $\mathcal{F}_{\text{SIG}}$, $\mathcal{F}_{\text{REG}}$, and $\mathcal{F}_{\text{SCS}}$

This hybrid protocol uses three ideal functionalities: $\mathcal{F}_{\text{SIG}}$, $\mathcal{F}_{\text{REG}}$, and $\mathcal{F}_{\text{SCS}}$. We describe here the ideal functionalities $\mathcal{F}_{\text{SIG}}$, $\mathcal{F}_{\text{REG}}$, and $\mathcal{F}_{\text{SCS}}$ defined by Canetti [5] in Figures 3, 4, and 5, respectively. We slightly modify $\mathcal{F}_{\text{REG}}$ from the original one in [5]. The modified registration functionality sends output (**Registered**, sid, v) to the party in order to clearly specify the activation of the key registering party.

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

Key Generation: Upon receiving a value (**KeyGen**, sid) from some party S, verify that $sid = (S, sid')$ 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 3: The signature functionality, $\mathcal{F}_{\text{SIG}}$.

5 The security of the protocol

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

Proof: Let $\mathcal{A}$ be an adversary in the real protocol and $\mathcal{S}_{\text{HYB}}$ be a hybrid protocol simulator that interacts with parties running π_{OFE} in the ($\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{REG}}, \mathcal{F}_{\text{SCS}}$)-hybrid model. $\mathcal{S}_{\text{HYB}}$ runs an internal copy of $\mathcal{A}$ as a black box, forwards any input from $\mathcal{Z}$ to $\mathcal{A}$ and vice versa.

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

1. Upon receiving input (**Register**, sid, v), verify that $sid = (P, sid')$. 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 4: The registration functionality, $\mathcal{F}_{\text{REG}}$.

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

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

1. Upon receiving input (**Establish-Session**, sid) from party I, verify that $sid = (I, R, sid')$ 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 $\{I, R\}$.

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

Here, we assume that for any $\mathcal{A}$ there exists $\mathcal{S}_{\text{HYB}}$ such that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\pi_{\text{OFE}},\mathcal{S}_{\text{HYB}},\mathcal{Z}}$ for any environment $\mathcal{Z}$.

We now construct a simulator $\mathcal{S}$ such that the view of the environment $\mathcal{Z}$ when interacting with $\mathcal{S}_{\text{HYB}}$ and π_{OFE} has the same distribution as $\mathcal{Z}$ when interacting with $\mathcal{S}$ and the ideal protocol for $\mathcal{F}_{\text{FE}}$. That is, for any $\mathcal{S}_{\text{HYB}}$ there exists $\mathcal{S}$ such that $\text{EXEC}_{\pi_{\text{OFE}},\mathcal{S}_{\text{HYB}},\mathcal{Z}} \approx \text{EXEC}_{\text{IDEAL}_{\mathcal{F}},\mathcal{S},\mathcal{Z}}$ for any environment $\mathcal{Z}$.

$\mathcal{S}$ runs an internal copy of $\mathcal{S}_{\text{HYB}}$ as a black box, forwards any input from $\mathcal{Z}$ to $\mathcal{S}_{\text{HYB}}$ and vice versa. $\mathcal{S}$ also runs an internal copy of each of the involved parties, and simulates $\mathcal{F}_{\text{SIG}}$, $\mathcal{F}_{\text{REG}}$, and $\mathcal{F}_{\text{SCS}}$.

We now describe the behavior of $\mathcal{S}$.

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

1. $\mathcal{S}$ simulates the processes of key generation and registration. It sends to $\mathcal{S}_{\text{HYB}}$ (in the name of

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

1. When activated with input $\underline{(\textbf{Initiate}, sid, M_A)}$, do:
   (a) A verifies that $sid = (A, B, sid')$ for some party B. If not, ignore the input. Else, it sends to $\mathcal{F}_{\text{SIG}}$ the message $(\textbf{KeyGen}, sid_A)$ where $sid_A = (A, sid)$, and obtains $(\textbf{Verification Algorithm}, sid_A, v_A)$. Next, A sends $(\textbf{Register}, sid_A, v_A)$ to $\mathcal{F}_{\text{REG}}$, and obtains $(\textbf{Registered}, sid_A, v_A)$.
   (b) A sends to $\mathcal{F}_{\text{SIG}}$ the message $(\textbf{Sign}, sid_A, m_{pre})$ where $m_{pre} = (M_A, A, T)$, and obtains $(\textbf{Signature}, sid_A, m_{pre}, \sigma_{pre})$. A then sends (m_{pre}, σ_{pre}) to B by using $\mathcal{F}_{\text{SCS}}$.
   (c) Upon receiving (m_{pre}, σ_{pre}), B verifies that $prop_A(M_A) = 1$. If not, B halts. Else, B sends $(\textbf{Retrieve}, sid_A)$ to $\mathcal{F}_{\text{REG}}$, and obtains $(\textbf{Retrieve}, sid_A, v_A)$ from $\mathcal{F}_{\text{REG}}$.
   (d) B sends $(\textbf{Verify}, sid_A, m_{pre}, \sigma_{pre}, v_A)$ to $\mathcal{F}_{\text{SIG}}$, and obtains $(\textbf{Verified}, sid_A, m_{pre}, v_A(m_{pre}, \sigma_{pre}))$. If $v_A(m_{pre}, \sigma_{pre}) = 1$, B outputs $\underline{(\textbf{Initiated}, sid)}$. Else, B halts.

2. When activated with input $\underline{(\textbf{Send}, sid, M_B)}$, do:
   (a) If $v_A(m_{pre}, \sigma_{pre}) = 1$, B sends M_B to A by using $\mathcal{F}_{\text{SCS}}$. Else, it halts.
   (b) Upon receiving M_B, A verifies that $prop_B(M_B) = 1$. If not, go to step(e). Else, A sends to $\mathcal{F}_{\text{SIG}}$ the message $(\textbf{Sign}, sid_A, m_{post})$ where $m_{post} = (M_A, A)$, and obtains $(\textbf{Signature}, sid_A, m_{post}, \sigma_{post})$ from $\mathcal{F}_{\text{SIG}}$.
   (c) A sends $(m_{post}, \sigma_{post})$ to B by using $\mathcal{F}_{\text{SCS}}$.
   (d) Upon receiving $(m_{post}, \sigma_{post})$, B sends $(\textbf{Verify}, sid_A, m_{post}, \sigma_{post}, v_A)$ to $\mathcal{F}_{\text{SIG}}$, and obtains $(\textbf{Verified}, sid_A, m_{post}, v_A(m_{post}, \sigma_{post}))$. If $v_A(m_{post}, \sigma_{post}) \neq 1$, go to step(e). Else, B sends $(\textbf{Verified}, sid)$ to A by using $\mathcal{F}_{\text{SCS}}$, and A outputs $\underline{(\textbf{Sent}, sid, M_B)}$.
   (e) B sends $(\textbf{Resolve}, sid, (m_{pre}, \sigma_{pre}), M_B)$ to T by using $\mathcal{F}_{\text{SCS}}$,
      i. Upon receiving $(\textbf{Resolve}, sid, (m_{pre}, \sigma_{pre}), M_B)$, T sends $(\textbf{Retrieve}, sid_A)$ to $\mathcal{F}_{\text{REG}}$, and obtains $(\textbf{Retrieve}, sid_A, v_A)$. It then sends $(\textbf{Verify}, sid_A, m_{pre}, \sigma_{pre}, v_A)$ to $\mathcal{F}_{\text{SIG}}$.
      ii. Upon receiving $(\textbf{Verified}, sid_A, m_{pre}, v_A(m_{pre}, \sigma_{pre}))$ from $\mathcal{F}_{\text{SIG}}$, T verifies that $v_A(m_{pre}, \sigma_{pre}) = 1$ and $prop_B(M_B) = 1$. If not, it halts. Else, it sends to $\mathcal{F}_{\text{SIG}}$ the message $(\textbf{KeyGen}, sid_T)$ where $sid_T = (T, sid)$, and obtains $(\textbf{Verification Algorithm}, sid_T, v_T)$. Next, it sends $(\textbf{Register}, sid_T, v_T)$ to $\mathcal{F}_{\text{REG}}$, and obtains $(\textbf{Registered}, sid_T, v_T)$.
      iii. T sends $(\textbf{Sign}, sid_T, \sigma_{pre})$ to $\mathcal{F}_{\text{SIG}}$, and obtains $(\textbf{Signature}, sid_T, \sigma_{pre}, \sigma_{TTP})$.
      iv. T sends M_B to A by using $\mathcal{F}_{\text{SCS}}$.
      v. Upon receiving M_B, A outputs $\underline{(\textbf{Sent}, sid, M_B)}$.

3. When activated with an input $\underline{(\textbf{Get}, sid)}$, do:
   (a) If B has obtained $(m_{post}, \sigma_{post})$ where $v_A(m_{post}, \sigma_{post}) = 1$, it outputs $\underline{(\textbf{Sent}, sid, M_A, \sigma_{post})}$.
   (b) Else, B sends $(\textbf{Get}, sid)$ to T by using $\mathcal{F}_{\text{SCS}}$. Upon receiving $(\textbf{Get}, sid)$, T sends σ_{TTP} to B by using $\mathcal{F}_{\text{SCS}}$. Upon receiving σ_{TTP}, B outputs $\underline{(\textbf{Sent}, sid, M_A, (\sigma_{pre}, \sigma_{TTP}))}$.

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

$\mathcal{F}_{\mathrm{SIG}}$) the message (**KeyGen**, sid_A), and obtains (**Verification Algorithm**, sid_A, s_A, v_A).

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

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

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

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

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

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

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

When $\mathcal{S}$ receives (**Get**, sid) from $\mathcal{F}_{\mathrm{FE}}$, $\mathcal{S}$ sends (**Send**, sid, $(m_{post}, \sigma_{post})$) to $\mathcal{F}_{\mathrm{FE}}$, 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 $\mathcal{S}_{\mathrm{HYB}}$ and π_{OFE} has the same distribution as of $\mathcal{Z}$ when interacting with $\mathcal{S}$ and the ideal protocol for $\mathcal{F}_{\mathrm{FE}}$.

Next, we construct $\mathcal{S}$ assuming party corruption. Since all messages are sent by using $\mathcal{F}_{\mathrm{SCS}}$ in π_{OFE}, it is only necessary to consider the case that $\mathcal{A}$ instructs a corrupted party to send modified data to $\mathcal{F}_{\mathrm{SCS}}$. The cases in which $\mathcal{A}$ instructs a corrupted party to register 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 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 $\mathcal{A}$ with the local state of the corrupted party except M_B. When black box $\mathcal{A}$ sends corruption message to party A, $\mathcal{S}$ (simulating for corrupted A) must send M_B to $\mathcal{A}$ after simulating B's sending of M_B.

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

1. $\mathcal{S}$ sends (**Sent**, sid, $|(m'_{pre}, \sigma'_{pre})|$) 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, $(m'_{pre}, \sigma'_{pre})$) 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**, $sid_A, m'_{pre}, v_A(m'_{pre}$, $\sigma'_{pre})$) to simulated B in the name of $\mathcal{F}_{\mathrm{SIG}}$. If $v_A(m'_{pre}, \sigma'_{pre}) = 1$, $\mathcal{S}$ sends **ok** to $\mathcal{F}_{\mathrm{FE}}$.

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

1. $\mathcal{S}$ sends (**Sent**, sid, $|(m'_{post}, \sigma'_{post})|$) 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, $(m'_{post}, \sigma'_{post})$) 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**, $sid_A, m'_{post}, v_A(m'_{post}$, $\sigma'_{post})$) to simulated B in the name of $\mathcal{F}_{\mathrm{SIG}}$. If $v_A(m'_{post}, \sigma'_{post}) = 1$, $\mathcal{S}$ simulates in the same way as the case where the parties are not corrupted.

3. Else, $\mathcal{S}$ simulates the process of resolution phase. $\mathcal{S}$ sends (**Sent**, sid, $|((m_{pre}, \sigma_{pre}), M_B)|$) 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, $((m_{pre}, \sigma_{pre}), M_B)$) 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**, $sid, M_A, (\sigma_{pre}, \sigma_{TTP})$) to $\mathcal{F}_{\mathrm{FE}}$.

4. Upon receiving (**Get**, sid) from $\mathcal{F}_{\mathrm{FE}}$, $\mathcal{S}$ simulates the process of B getting 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**, $sid_T, \sigma_{pre}, \sigma_{TTP}$)$|$ to $\mathcal{S}_{\mathrm{HYB}}$. Upon receiving **ok** from $\mathcal{S}_{\mathrm{HYB}}$, $\mathcal{S}$ sends **ok** to $\mathcal{F}_{\mathrm{FE}}$.

The case where party B is corrupted. When $\mathcal{A}$ 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 $prop_B(M'_B) = 1$, $\mathcal{S}$ simulates in the same way as the case where parties are not corrupted.

2. Else, if $\mathcal{A}$ instructs corrupted B to send (**Resolve**, sid, $(m_{pre}, \sigma_{pre}), M_B$) 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**, $sid, M_A, (\sigma_{pre}, \sigma_{TTP})$) to $\mathcal{F}_{\mathrm{FE}}$.

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

6 Conclusion

In this paper, we defined the fair exchange functionality $\mathcal{F}_{\mathrm{FE}}^{(prop_A,\ prop_B,\ verify)}$ in the universal composability framework and presented an optimistic fair exchange protocol that UC-realizes this functionality in the $(\mathcal{F}_{\mathrm{SIG}}, \mathcal{F}_{\mathrm{REG}}, \mathcal{F}_{\mathrm{SCS}})$-hybrid model.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communication*, volume 18, No. 4, pages 593-610, 2000.

[2] G. Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 138-146, 1999.

[3] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line TTP. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 77-85, 1998.

[4] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416-432. Springer-Verlag, 2003.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Foundations of Computer Science conference*, 2001. Full version at http://eprint.iacr.org/2000/067/.

[6] R Canetti. Universally composable signature, certification, and authentication. *17th Computer Security Foundations Workshop*, 2004. http://eprint.iacr.org/2001.

[7] R Canetti, H Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337-351. Springer-Verlag, 2002.

[8] R. Canetti and T. Rabin. Universal composition with joint state. *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265-281. Springer-Verlag, 2003.

[9] Y. Dodis and L. Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In *Proceedings of the 2003 ACM workshop on Digital rights management*. pages 47-54, 2003.

[10] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, volume 28, No. 6, pages 637-647, 1985.

[11] Y. Okada, Y. Manabe and T. Okamoto. Optimistic fair exchange protocol for E-Commerce. *The Symposium on Cryptography and Information Security*, 2006.

[12] T. Okamoto and K. Ohta. How to simultaneously exchange secrets by general assumptions. In *Proceedings of 2nd ACM Conference on Computer and Communications Security*, pages 184-192, 1994.

[13] J. M. Park, E. Chong, and H. J. Siegel. Constructing fair-exchange protocols for E-commerce via distributed computation of RSA signatures. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 172-181, 2003.

[14] J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 55-61, 1996.

[15] J. Zhou, D. Gollmann, An efficient non-repudiation protocol. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 126-132, 1997.