Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 29 (2006) 2770–2780
computer communications
www.elsevier.com/locate/comcom

A new dependable exchange protocol ☆

Hao Wang [a,*], Heqing Guo [a], Manshan Lin [a], Jianfei Yin [c], Qi He [b], Jun Zhang [b]

[a] *School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China*
[b] *School of Computer Engineering, Nanyang Technological University, Singapore*
[c] *College of Information Engineering, Shenzhen University, Shenzhen 518060, China*

Available online 6 December 2005

Abstract

As electronic transaction becomes common practice in real-world business, its dependability develops into a major concern, especially in critical transactions, e.g., electronic payment and electronic contract signing. Many recent fair-exchange protocols can recover the transaction from network failures; however, few can survive local system failures. In this paper, we propose a new *Dependable Exchange Protocol*. With proper convertible signature scheme and message logging method, the exchange protocol provides a recovery method for network and local system failures. To the best of our knowledge, this protocol is the first fault-tolerant exchange protocol in the context of offline TTP and asynchronous channels.
© 2005 Elsevier B.V. All rights reserved.

Keywords: E-commerce; Dependability; Exchange protocol; Fairness; Fault tolerance

1. Introduction

As electronic transaction becomes common practice in real-world business, its dependability develops into a major concern. When the transaction between organizations is executed on network, they may face the risks of broken fairness in case of local system failures [2], network failures, cheating behavior of either involved organization, etc. Thus, it is very important for them to follow some kind of protocol assuring *dependability*. Dependability guarantees fairness for involved parties and recoverability from failures. Fairness is a vital requirement for electronic transactions, which means that when the electronic transaction terminates, either both/all parties get their expected items or neither does.

We first set up the application scenario for our transaction protocol: company B (the client, denoted as Bob) is going to purchase some electronic goods from company A (the merchant, denoted as Alice) and they have settled on the goods and the price. Now they need to finish the exchange of Bob's check with Alice's goods. Bob's check, issued by Bob's bank, is an electronic document composed of his bank-certified account information, and goods information, which can be validated only after signed by his signature. With the signed check, Alice can get her money paid from Bob's bank. Note that *anonymity* is not considered in this scenario and interested readers can find detailed discussion in [3]. With this scenario set, we state the assumptions of our protocol explicitly (refer to Section 3).

To realize fairness, a TTP (trusted third party who is assumed to be fully trusted) must be included into the protocol when network or system failures occur [4]. In [5], Asokan et al. introduce the idea of optimistic approach and present fair protocols with offline TTP, in which TTP intervenes only when an error occurs (network error or malicious party's cheating). Protocols with offline TTP can recover network failures between Alice and Bob. Yet the recovered messages are different from those produced by the sender or the recipient, which make the protocols

☆ A preliminary version of this paper was published at WICS'05 [1].
* Corresponding author. Tel.: +86 20 33178911.
E-mail addresses: iswanghao@acm.org (H. Wang), guozhou@scut.edu.cn (H. Guo), lmshill@hotmail.com (M. Lin), yjhhome@hotmail.com (J. Yin), qihe0001@ntu.edu.sg (Q. He), jzhang@ntu.edu.sg (J. Zhang).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.10.028

suffer from weak fairness, as the recovered messages may lose some functionality of the original ones. *Transparent TTP* is first introduced by Micali [6] to solve this problem. The TTP-recovered messages are exactly the same as the ones from the original sender. In this way, the outcome messages will not indicate whether the TTP has been involved, so the recovery is done in a transparent way. Of course, the transparent recovery may decrease observability of the exchange (for detailed discussion, see Section 5.1.1).

Until now, the best method to realize transparent TTP is to use *convertible signatures* (CS). It sends a partial committed signature that can be converted into a full signature (that is a normal signature) by both the TTP and the signer. Recently, Park et al. [7] present a very efficient protocol in which the output evidences are standard RSA signatures and the partial signature is non-interactively verifiable. But very soon, Dodis and Reyzin [8] break the scheme by proving that the TTP can obtain the signer's entire secret key with only her registration information. In the same paper, they propose a new CS scheme (*DR signature scheme*) to produce an equally efficient but more secure protocol.

Unfortunately, all these protocols do not consider local system failures and assume that local systems of Alice, Bob, and TTP are all reliable. To our knowledge, there are only very few efforts to overcome the local system failures problem. Liu et al. [2] propose the *Semantics-based Message Logging* (SbML method) to enable recovery of local system failures. The SbML is a logging method which optimizes the number of messages that need to be pessimistically logged (log all messages before sending out). Involved parties can define their critical points (called *point-of-no-return*) in the protocol run and message will be logged before they enter the defined points. This logging method works in protocols with online TTP [9]. Although they mention offline TTP, fairness after failures can still be potentially broken (as discussed in Section 5.2). As Ezhilchelvan and Shrivastava [10] argue that as the number of protocol messages is small, to minimize the overhead of logging may not worth the effort. In the same paper, Ezhilchelvan and Shrivastava have proposed a whole set of fault-tolerant protocols. They have considered different types of misbehaviors, communication channel, and online/offline TTP. But they have not proposed any protocol that is fault tolerant of local system failures in the context of offline TTP and asynchronous channels.

In this paper, we propose the first dependable exchange protocol, with transparent TTP (as we already know, it is an upgraded version of offline TTP) and asynchronous channels, to be fault tolerant of network and local system failures. We use the DR signature scheme and a proper non-interactive zero-knowledge proof method [11] as the basic cryptographic tools to realize transparent TTP. To enable recovery of network and local system failures, we use pessimistic message logging method and introduce a new inquiry sub-protocol. We prove that the transaction protocol is dependable. Also we argue that the transparent TTP will not affect the observability of our protocol. When implementing the protocols, we incorporate the label and message construction design principles proposed by Gurgens et al. [12]. When using offline TTP, the problem of encrypted item validation arises and we propose two methods considering different application scenarios.

In Section 2, we first lay out all the preliminaries: the dependability definition of electronic exchange, the cryptographic tools including DR signature scheme and the zero-knowledge proof method. Then, we present the exchange protocol in Section 3, with detailed analysis in Section 4. In Section 5, we compare our protocol with former ones and discuss several possible extensions: the observability and encrypted item validation problem with transparent TTP, unreliable network between Alice/Bob and the TTP, and protocol information storing issue for TTP.

2. Preliminaries

2.1. Dependability of electronic exchange

When we design an exchange protocol, what is meant to be achieved must be made clear at first. Five requirements for fair exchange have been formulated by Asokan et al. in [13] and further discussed in [14]. Their requirement definitions do not presume new advances in recent years. In [15], Kremer et al. study many former fairness definitions and present a well-knitted definition. Recently, we present a set of new requirements for fair protocols with transparent TTP [16]. Based on that, we extract four properties for dependable protocol and we define a protocol dependable if it satisfies all these properties.

Definition 1 (*Effectiveness*)**.** An exchange protocol is *effective* if when both parties are honest, there exists a successful exchange of both parties' expected items.

Definition 2 (*Fairness*)**.** An exchange protocol is *fair* if when the protocol run ends, either both parties get their expected items or neither of them gets anything useful.

Definition 3 (*Timeliness*)**.** An exchange protocol is *timely* if the protocol can be completed in a finite amount of time while preserving fairness for both exchangers.

Definition 4 (*Non-repudiability*)**.** An exchange protocol is *non-repudiable* if when the exchange succeeds, either payer or payee cannot deny (partially or totally) his/her participation.

With the above definitions of properties, we can further define the dependability as follows:

Definition 5 (*Dependability*)**.** An exchange protocol is *dependable* if it assures effectiveness, fairness, timeliness and non-repudiability even in case of unreliable local systems and communication channels.

2.2. Cryptographic tools

2.2.1. Dodis–Reyzin convertible signature scheme

The DR signature is based on a recent widely used DSA-like signature scheme called *Gap Diffie–Hellman* (*GDH*) *signature* and the corresponding *GDH groups* (see Section 4 in [8] for detailed description).

GDH signature. Assume G is a multiplicative group of prime order p. Key generation algorithm of the GDH signature scheme picks a GDH group of order p, and random $g \in G$, $x \in Z_p$. It computes $h = g^x$, and set the public key to be (g, h) (G, p is public accessible), and the secret key to be x. To sign a message m, one computes $\sigma = H(m)^x$, where $H(m)$ is a random oracle. To verify σ, one outputs $V_{\text{DDH}}(g, h, H(m), \sigma)$, that is, test if $\log_g h = \log_{H(m)}\sigma$ (outputting **TRUE** means equal). And we use the non-interactive zero-knowledge proof method proposed in [11].

DR signature. This CS signature scheme contains one register procedure and several signing/verifying algorithms. See Fig. 1, (1) in step 0 and 0′, the signer registers to the TTP who will make sure it can convert the signer's partial signature into a full one; (2) then the signer sends his/her partial signature in step 1; (3) if everything is OK (like the signer gets what s/he wants), the signer will send the full signature in step 2; if the signer quit prematurely (for cheating, or system breakdown), the recipient can turn to the TTP and get the full signature.

Register procedure. Signer (say Alice) chooses random $g \in G$, $x, x_1 \in Z_p$, computes $x_2 = x - x_1 \bmod p$, $h = g^x$, $h_1 = g^{x_1}$, and sets her public key $pk = (g, h)$, secret key $sk = (x, x_1)$, partial public key $ppk = h_1$, partial secret key $psk = x_2$, then she sends the pk, ppk, and psk to the TTP, the TTP will check whether $h = h_1 g^{x_2}$ so that it can finish the signature conversion, if so, TTP can send back a confirmation.

Signing/verifying algorithms of full signature. They are just the signing/verifying algorithms of normal GDH signature: $FS(m) = \sigma = H(m)^x$, $Ver(m, \sigma) = V_{\text{DDH}}(g, h, H(m), \sigma)$.

Signing/verifying algorithms of partial signature. Similar to former ones but using the public key h_1: $PS(m) = \sigma' = H(m)^{x_1}$, $PVer(m, \sigma') = V_{\text{DDH}}(g, h_1, H(m), \sigma')$.

Converting algorithm. The TTP run this algorithm $Convert(m, \sigma')$ to convert $PS(m)$ to $FS(m)$: it will first check whether $PVer(m, \sigma') = \textbf{true}$, if holds, it outputs $FS(m) = \sigma' H(m)^{x_2}$.

Dodis and Reyzin prove the DR signature scheme is just as secure as the normal GDH signature scheme (Theorem 3 in [8]).

2.2.2. A secure zero-knowledge proof

As described above, to verify DR signature, one must verify $V_{\text{DDH}}(g, h, H(m), \sigma)$, that is to test whether $\log_g h = \log_{H(m)}\sigma$, where $\sigma = H(m)^x$. Alice must generate a proof to convince the truth to Bob using following method:

Generating proofs. Alice selects a random number $u < p$ and calculates

$$\begin{cases} \Omega = g^u (\operatorname{mod} p), \\ M = H(m)^u (\operatorname{mod} p), \\ v = h(\Omega, M), \\ r = u + vx (\operatorname{mod} p). \end{cases} \tag{1}$$

In this way, the proof of the $PS(m)$, denoted by $pf(PS(m))$, is (r, Ω, M).

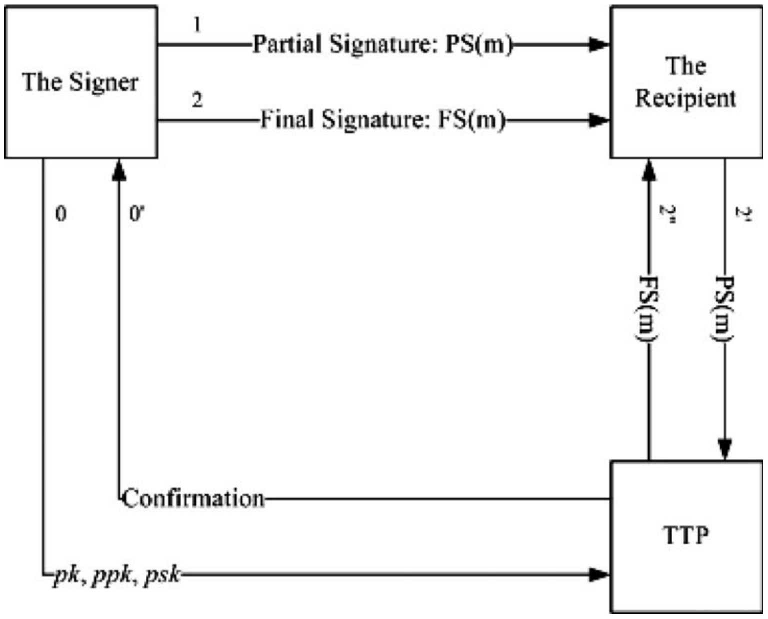


Fig. 1. DR signature scheme.

Verifying proofs. When Bob gets the $PS(m)$ and $pf(PS(m))$, he calculates

$$v = h(\Omega, M) \tag{2}$$

and verifies

$$\begin{cases} \Omega PS(g)^v = g^r (\mathrm{mod}\, p), \\ MPS(H(m))^v = H(m)^r (\mathrm{mod}\, p). \end{cases} \tag{3}$$

If they hold, Bob can be convinced that $\log_g h = \log_{H(m)} \sigma$.

2.3. Basic protocol model

As we have mentioned, the exchange protocol involves three parties: Alice, Bob, and the TTP (see Fig. 2). Besides the exchange of electronic goods and check, Alice and Bob also need to exchange non-repudiation evidences proving they have actually participated in this exchange. Alice need to give Bob her signed evidence of origin (**NRO**) proving she actually send the goods and because the check will have Bob's signature, it can be used as the evidence of receipt (**NRR**) proving Bob has received the goods.

The basic idea of the protocol is as follows: (1) Exchange. Alice and Bob will perform two rounds of message exchanges and Alice initiates the protocol by sending out the first message. The first round is to exchange the encrypted goods and partially signed check (also known as commitment) and the second round is to exchange the secret key k for the goods and fully signed check; (2) Dispute resolution. If Bob does NOT respond in the first round, Alice will run the abort sub-protocol with TTP and TTP will not recover messages of this exchange any more. In the second round, if Alice or Bob does NOT receive message from the other party, s/he will run the recover sub-protocol with TTP and TTP can recover the k and the fully signed check for both parties.

Our protocol has following assumptions:

Communication network. We assume the communication channel between Alice and Bob is unreliable and channels between exchangers (Alice/Bob) and TTP are resilient (discussion of unreliable channels between Alice/Bob and TTP can be found in Section 5). Messages in a resilient channel can be delayed but will eventually arrive. On the contrary, messages in unreliable network may be lost. We also assume that both types of network cannot be eavesdropped by any third party.

Cryptographic tools. Encryption tools including symmetric encryption, asymmetric encryption, and normal signature are secure. In addition, the adopted signature scheme is message recovery.

Honest TTP. The TTP should send a valid and honest reply to every request. Honest means that when the TTP is involved, if a recover decision is made, Alice gets the payment and Bob gets the goods; if a abort decision is made, Alice and Bob get the abort confirmation and they cannot recover the exchange in any future time.

Local systems. Local systems of Alice, Bob, and TTP are recoverable with proper message logging.

To describe the protocol, we need to use some notations concerning the necessary cryptographic tools:

- $E_k()/D_k()$: a symmetric-key encryption/decryption function under key k.
- $E_X()/D_X()$: a public-key encryption/decryption function under pk_X/sk_X.

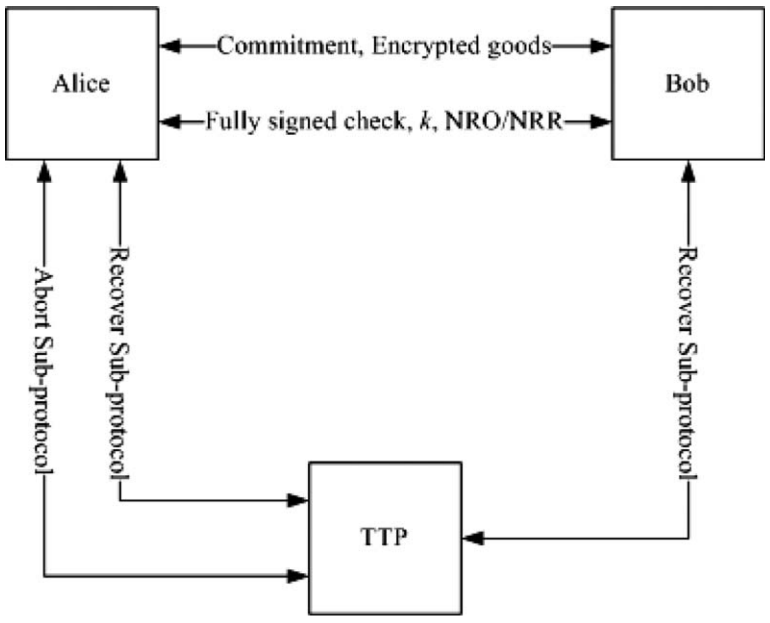


Fig. 2. The basic protocol model.

- $S_X()$: ordinary signature function of X.
- k: the key used to cipher goods.
- pk_X/sk_X: public key/secret key of X.
- $cipher = E_k(\text{goods})$: the cipher of goods under k.
- $X \to Y$: transmission from entity X to Y.
- $h()$: a collision-resistant one-way hash function.
- $PS_X()/FS_X()$: partial/final signature function of X.
- *Check*: the check destined for A, it contains transaction identity, item identity, price information, B's account information, etc.
- l: a label that uniquely identifies a protocol run.
- f: a flag indicating the purpose of a message.

3. The dependable payment protocol

In this section, we present the dependable protocol described in the payment scenario mentioned in Section 1. This protocol adds two parts to the basic model described in Section 2.3: the register sub-protocol and the inquiry sub-protocol (see Fig. 3). The register sub-protocol is presented because both parties must negotiate with TTP on some common parameters like shared secret keys. The registration protocol between the Alice/Bob and TTP needs to be run only once. And the resulting common parameters can be used for any number of transactions. The inquiry sub-protocol is used by Alice/Bob after she/he recovers her/his system after failures and gets proper messages to protect her/his own benefits. Also this inquiry sub-protocol can help recover from some communication failures with TTP.

When implementing the protocol, we follow the principles proposed by Gurgens et al. [12] as briefly described below:

(1) Label design principles
Verifiability. The creation of a label should be verifiable by everyone;
Uniqueness. The label should be able to uniquely identify a protocol run;
Secrecy. The values that are used to compute the label must not reveal any useful information about the exchange items (i.e., the goods).

(2) Message construction principles
Authenticity. All message parts should be included in the respective signature (in plaintext or as hash);
Verifiability. Every recipient should be able to verify this message;
Context of message. It should be possible for the recipient of a message to identify the protocol run to which its parts belong.

The protocol is described in the form of program modules (similar to Vogt et al. [14]) and the notation ⟨event⟩ : ⟨description⟩ to describe the steps of every module. The ⟨event⟩ can be sending a message from X to Y (denoted by $X \to Y$) or some local operations of a participant (denoted by his/her name, i.e., A, B, or TTP). The ⟨description⟩ is a brief explanation of contents of the message being sent or operations performed locally.

Evidences. During the protocol, following evidences are generated:

The recovery request: $\mathbf{Rec}_X = S_X(f_{\mathbf{Rec}X}, Y, l)$.
The inquiry request: $\mathbf{Inq}_X = S_X(f_{\mathbf{Inq}X}, Y, l)$.
The abort request: $\mathbf{Abort} = S_A(f_{\mathbf{Abort}}, TTP, l)$.
The abort confirmation: $\mathbf{Con}_a = S_{TTP}(f_{\mathbf{con}a}, A, B, l)$.

Registration Sub-protocol. To participate in a payment protocol, both Alice and Bob need to run the register procedure with the TTP as required by DR signature.

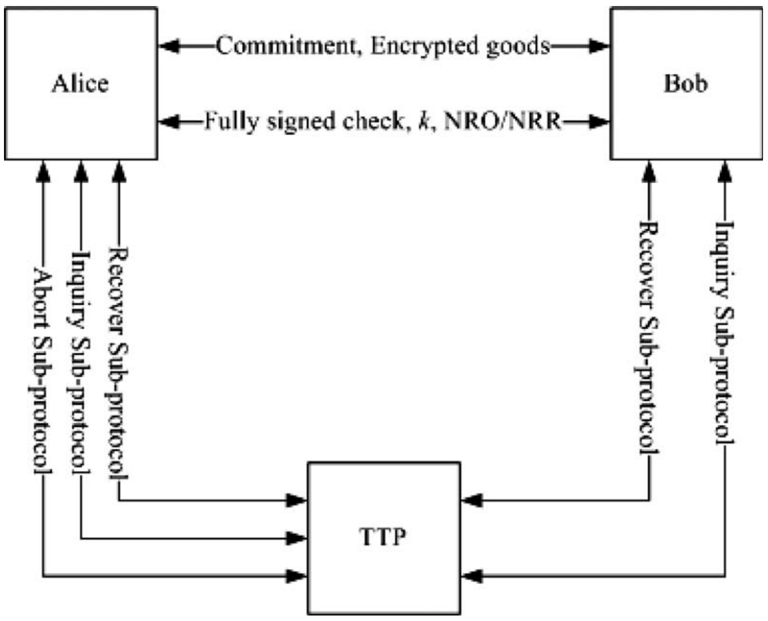


Fig. 3. The dependable protocol.

3.1. Main protocol

After Alice and Bob settle the price and the goods, they can follow the main protocol. The label for a protocol run is computed by Alice: $l = h(A, B, TTP, h(cipher), h(k))$. And the secret key is encrypted with l, i.e., $E_{TTP}(l,k)$, to ensure this encrypted key cannot be decrypted in a different protocol run. In our protocol, we denote the content to be signed by Alice as $a = (f_{\mathbf{NRO}}, B, l, h(k), cipher, E_{TTP}(l,k))$ and Bob's as $b = (A, l, check)$, so the signed check will be $PS_B(b)$ and $FS_B(b)$. With this construction, the signed check can only be used by Alice to claim the money. The logging function for all parties is $log(m)$ which will store m into reliable storage and the message is only accessible by the owner. Note that they both make their own messages logged on reliable storage before run the protocol. The main protocol is as follows (see also Table 1):

Step 1. Alice sends encrypted goods (*cipher*) with the key k encrypted by the TTP's public key ($E_{TTP}(l,k)$), her partial signature on them to initiate the payment process.
Step 2. If Bob decides to give up or he does not receive Alice's message in time, he can simply quit and retain fairness. When he receives the message, he will first run $PVer(a, \sigma'_A)$, if it equals **true**, he will send his *check* and his partial signature on it ($PS_B(b) = \sigma'_B$) to Alice. Otherwise, he quits the protocol.
Step 3. If Alice decides to give up or she does not receive Bob's message in time, she can invoke the *abort* sub-protocol to prevent a later resolution by the TTP. When she receives the message, she will first run $PVer(b, \sigma'_B)$, if it equals **true**, she will log the message and the state information, then send k and her full signature on a ($FS_A(a) = \sigma_A$) to Bob. Otherwise, she also invokes the *abort* sub-protocol.
Step 4. If Bob detects that his channel with Alice is broken or does not receive the message in time, he can invoke the *recover* sub-protocol. When he receives the message, he will check whether k can decrypt the *cipher* and the *goods* is satisfactory, also he will run $Ver(a, \sigma_A)$, if all these checking pass, he will log the message and the state information, then send his *check* and his full signature on it ($FS_B(b) = \sigma_B$) to Alice. Otherwise, he will invoke the *recover* sub-protocol.
Step 5. If Alice detects that her channel with Bob is broken or does not receive the message in time, she can invoke the *recover* sub-protocol. When she receives the message, she will run $Ver(b, \sigma_B)$. If it equals **true**, she will accept the check. Otherwise, she will invoke the *recover* sub-protocol.

3.2. Recover sub-protocol

Whenever necessary, Alice/Bob (noted by X) will invoke the *recover* protocol to let the TTP decide whether finish or abort the payment process. The TTP keeps a record on whether the protocol has been recovered or aborted (denoted by two variables: *aborted* and *recovered*). The recover sub-protocol is as follows (see also Table 2):

Step 1. X sends to the TTP $E_{TTP}(l,k)$, $PS_A(a) = \sigma'_A$, *check*, $PS_B(b) = \sigma'_B$ to initiate a recover process. Because of the resilient channel between X and the TTP, this message will eventually arrive at the TTP.
Step 2. When the TTP receives the message, it will first check whether the protocol has already been recovered or aborted. If so, it will stop because it is sure that both parties have got the recovered items or the abort confirmation. Then, it will decrypt $E_{TTP}(l,k)$ with its secret key sk_{TTP}, if succeeds, it will run $PVer(a, \sigma'_A)$ and $PVer(b, \sigma'_B)$. If both equals **true**, the TTP will run $Convert(a, \sigma'_A)$ and $Convert(b, \sigma'_B)$. After all these operations succeed, TTP will log the message and the variable *recovered*, then send the $FS_B(b) = \sigma_B$ to Alice and $FS_A(a) = \sigma_A$ & k to Bob. If either checking fails, it will abort the protocol and send confirmations to Alice and Bob.

Table 1
Main protocol

Main protocol
A: $log(B, l, a, cipher, k)$
B: $log(A, l, check)$
$A \rightarrow B$: $f_{\mathbf{EOO}}$, B, l, $h(k)$, $cipher$, $E_{TTP}(l,k)$, $PS_A(a)$, $pf(PS_A(a))$
B: **if not** $Ver(a, PS_A(a))$ **then** stop
else $log(A, l, h(k), cipher, E_{TTP}(l,k), PS_A(a), pf(PS_A(a)))$
$B \rightarrow A$: $f_{\mathbf{EOR}}$, A, l, $PS_B(b)$, $pf(PS_A(b))$
A: **if** times out **then abort**
elseif not $Ver(b, PS_B(b))$ **then abort**
else $log(B, l, PS_B(b), pf(PS_A(b)))$
$A \rightarrow B$: $f_{\mathbf{NRO}}$, B, l, k, $FS_A(a)$
B: **if** times out **then** call **recover**[$X{:=}B$, $Y{:=}A$]
else $log(A, l, k, FS_A(a))$
$B \rightarrow A$: $f_{\mathbf{NRR}}$, A, l, $FS_B(b)$
A: **if** A times out **then** call **recover**[$X{:=}A$, $Y{:=}B$]

Table 2
Recover sub-protocol

Recover sub-protocol
$X \rightarrow TTP$: $f_{\mathrm{RecX}}, f_{\mathrm{Sub}}$, Y, l, $h(cipher)$, $\mathrm{h}(k)$, $E_{TTP}(l,k)$, $\mathbf{Rec_X}$, $PS_A(a)$, $PS_B(b)$
TTP: $log(f_{\mathrm{RecX}}, f_{\mathrm{Sub}}, A, B, l, h(cipher), h(k), E_{TTP}(l,k), \mathbf{Rec_X},$
$PS_A(a), PS_B(b))$
if $h(k) \neq h(D_{TTP}(E_{TTP}(l,k)))$ **or** *aborted* **or** *recovered* **then** stop
else if $PVer(a, PS_A(a)) \neq 1$ or $PVer(a, PS_A(a)) \neq 1$ **then** stop
else *recovered* = true
$Convert(PS_A(a), x_{2A})$ and $Convert(PS_B(b), x_{2B})$
$log(A, B, l, recovered, FS_A(a), k, FS_B(b))$
$TTP \rightarrow A$: $f_{\mathbf{NRR}}$, A, l, $FS_A(a)$
$TTP \rightarrow B$: $f_{\mathbf{NRO}}$, B, l, k, $FS_B(b)$

Table 3
Inquiry sub-protocol

```
Inquiry sub-protocol
$X \rightarrow TTP$: $f_{\mathrm{InqX}}$, $\mathbf{Inq_X}$
  $TTP$: **if** *aborted* **then**
$TTP \rightarrow X$: $f_{\mathbf{Con}a}$, $A$, $B$, $l$, $\mathbf{Con}_a$
  $TTP$: **elseif** *recovered* **then**
        **if** $X = A$ then
$TTP \rightarrow A$: $f_{\mathbf{NRR}}$, $A$, $l$, $FS_A(a)$
        **else**
$TTP \rightarrow B$: $f_{\mathbf{NRO}}$, $B$, $l$, $k$, $FS_B(b)$
     **else**
$TTP \rightarrow X$: **null**
```

3.3. Inquiry sub-protocol

After recovering from local system failures, Alice/Bob (denoted as X) can invoke the *inquiry* sub-protocol to check the current status of the transaction and get what s/he deserves. The inquiry sub-protocol is as follows (see also Table 3):

Step 1. X sends an inquiry request to the TTP. Because of the resilient channel between X and the TTP, this message will eventually arrive at the TTP.
Step 2. On the inquiry request, TTP will check the current status of the protocol according to the label l. If no record is available, that means that protocol has not been submitted to TTP and X can directly recover the protocol run with Y. So TTP will just need to return a *null* message to X. If the protocol has been recovered, TTP will send the recovered message to X, that is, $FS_A(a)$, k (for Bob) or $FS_B(b)$ (for Alice). If the protocol has been aborted, TTP will send the abort confirmation to X.

3.4. Abort sub-protocol

In step 2 of the main protocol, Alice can invoke this sub-protocol to make the TTP abort this payment protocol run. The abort sub-protocol is as follows (also see Table 4):

Step 1. Alice sends an abort request to the TTP. Because of the resilient channel between X and the TTP, this message will eventually arrive at the TTP.
Step 2. If the protocol has not been recovered or aborted, the TTP will abort the protocol and log the message and the variable *aborted*, then send confirmations to both parties.

Table 4
Abort sub-protocol

```
Abort sub-protocol
$X \rightarrow TTP$: $f_{\mathbf{Abort}}$, $l$, $B$, **abort**
  $TTP$: **if** *aborted* **or** *recovered* **then** stop
      **else** *aborted*=true
         $log(A, B, l, aborted)$
$TTP \rightarrow A$: $f_{\mathbf{Con}a}$, $A$, $B$, $l$, $\mathbf{Con}_a$
$TTP \rightarrow$ B: $f_{\mathbf{Con}a}$, $A$, $B$, $l$, $\mathbf{Con}_a$
```

4. Analysis of the protocol

Following is the analysis with respect to the dependability definition in Section 2.

Claim 1. *Assuming the channel between Alice and Bob is unreliable and adopted cryptographic tools are secure, the protocol satisfies the effectiveness requirement.*

Proof. When both Alice and Bob are honest, thus they will follow the protocol to send messages. If the probability of successful transmission in the unreliable channel is δ, then the probability of successful execution of one main protocol run will roughly be δ^4. Even it is small, successful execution without TTP's involvement is still possible. Thus, the protocol satisfies the effectiveness requirement. □

Claim 2. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools are secure and the TTP is honest, the protocol satisfies the fairness requirement.*

Proof. The fairness can be proved considering three aspects: fairness for Alice, fairness for Bob, and recovered fairness after system failures at TTP.

Fairness for Alice. Assuming Alice is honest, then risks she may face include:

(1) She did not receive any message or the message is invalid in Step 3. She can request abort to prevent that Bob may call a recovery later. If Bob's recovery request arrives at the TTP before her abort request, the TTP still will send the recovered item and evidence to her. Thus, will not affect her benefit.
(2) She did not receive any message or the message is invalid in Step 5. She can submit a recovery request, because the TTP is honest, the exchange will be forced to complete. If Bob sent a recovery request during this period, the result will be the same; if Bob sent an abort request which arrived before Alice's recovery request, the exchange will be aborted by the TTP, and no party can gain advantage.
(3) *Local system failures*. After Alice recovers from local system failure, she can instantly invoke inquiry sub-protocol to check the current status; if she has submitted abort or recover request before her system fails, she will get proper messages (abort confirmation or recovered messages) from TTP; if Bob has submitted recover request before or during the failure, she will get recovered messages from TTP; if no TTP's involvement before or during the failure, she can simply contact Bob to continue the transaction. So her fairness is assured.

Fairness for Bob. Assuming Bob is honest, then risks he may faces include:

(1) He did not receive any message or the message is invalid in Step 2. He can simply stop without any risk. And at this time, Alice cannot call recovery.
(2) He did not receive any message or the message is invalid in Step 4. He can request recovery and the exchange will be forced to complete. If Alice request recovery at the same time, the result will be the same.
(3) *Local system failures*. After Bob recovers from local system failure, he can instantly invoke inquiry subprotocol to check the current status; if he has submitted recover request before the failure, he will get recovered messages from TTP; if Alice has submitted abort or recover request before or during the failure, he will get proper messages (abort confirmation or recovered messages) from TTP; if no involvement before or during the failure, he can simply contact Alice to continue the transaction. So his fairness is assured.

Recovered fairness after TTP system failures. Cases of TTP system failures include:

(1) Alice has submitted abort request before TTP system failure, and TTP has sent both parties the abort confirmation. Because TTP has logged request message and the variable *aborted*, so after TTP recovers the information about this protocol run, the TTP will deny any later recovery request by either Alice or Bob.
(2) Alice/Bob has submitted recover request before TTP system failure, and TTP has sent both parties the recovered messages. Because TTP has logged the request message and the variable recovered, after TTP recovers the information about this protocol run, the TTP can re-run the recovery operations (if necessary) and will ignore Alice's later abort request.
(3) Alice/Bob has submitted abort/recover request during system failure at TTP. Alice/Bob can re-submit request after TTP's recovery or TTP can actively broadcast the failure information so that all requesting parties can re-submit their requests. □

Claim 3. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools are secure and the TTP is honest, the protocol satisfies timeliness requirement.*

Proof. Alice can conclude the protocol in one of the two ways:

(1) Requesting abort before sending the message of Step 3.
(2) Requesting recovery in any other time.

Bob can conclude the protocol in one of the two ways:

(1) Stopping at any time before sending the message of Step 2.
(2) Requesting recovery in any other time.

With the channel assumption, the abort confirmation or the recovered information will arrive to both parties in a finite amount of time. And all these conclusions, as discussed in the proof of Claim 2, will not hurt either party's interests. So the timeliness is guaranteed. □

Claim 4. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools (including the adopted zero-knowledge proof method) are secure, the TTP is honest, the protocol satisfies non-repudiation requirement.*

Proof. When the exchange succeeds, either by following the main protocol or recovered by the TTP (including recovered message after inquiry), Alice will get $FS_B(b) = \sigma_B$, and Bob will get $FS_A(\text{a}) = \sigma_A$ & k. If a payment protocol succeeds, by showing $FS_B(b)$, Alice can convince outside parties that Bob has received *goods* and claim her money from Bob's bank. Similarly, Bob can prove that Alice has sent *goods*. In this way, the non-repudiation requirement is satisfied. □

With all these claims, we can easily see that the protocol is dependable:

Theorem 1. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools (including the adopted zero-knowledge proof method) are secure and the TTP is honest, the protocol is dependable.*

Now the protocol is effective, fair, timely, non-repudiable, and transparently recoverable, as defined in Section 2, the protocol is dependable.

5. Conclusions

In this paper, we produce a dependable transaction protocol with transparent TTP. We have shown that the protocol are practical as it has high recoverability and can survive relatively unreliable (asynchronous) network. Several discussions are as following:

5.1. Discussions

5.1.1. Observability with transparent TTP

As we can see now, the TTP can recover the partial messages in the first round exchange and make the outcome messages indistinguishable from the ones of successful exchange. One may argue that this kind of properties may not be desirable in some circumstances as it definitely decreases the observability of the exchanging process.

Now we show how this protocol can be slightly altered and made fully observable. In the recover sub-protocol, the TTP can include a tag signed with its signature, like $S_{TTP}(l, A, B)$ along with the recovered messages to both parties. If either Alice or Bob claims falsely that TTP has not been involved, the other party can show this TTP-signed tag to prove that it is not true. If both parties jointly claim that TTP has not been involved, the TTP can show the $FS_A(a)$ and $FS_B(b)$ to prove its involvement.

5.1.2. Encrypted item validation

In the first round, Alice sends the encrypted goods as commitment, but Bob has no way of checking whether it is actually the goods he wants and the problem of encrypted item validation arises. We have proposed a solution using *item registry* in [17]. The basic idea is that the goods (called item) owner must register the item with an item certifying authority (ICA) and get a certificate for the encrypted item. But this solution is only sensible in the case that the owner has a large number of this kind of item and need to exchange many times. Because the cost of getting a certificate will be rather high if the exchange will be performed only a few times.

When the item will be exchanged only a few times, we argue that the self-generated certificate of the encrypted item should be enough. The self-generated certificate is constructed by signing the encrypted item with the owner signature, i.e., $S_A(cipher)$ in our scenario. If Alice cheats by sending the wrong encrypted item to Bob, when the exchange ends and Bob finds out the goods is the wrong one, he can show the $S_A(cipher)$ to the TTP and prove that Alice has cheated.

5.1.3. Protocol information storing in TTP

As the inquiry sub-protocol is introduced, TTP needs to store the protocol information long enough so that TTP can answer inquiries from either parties. But problem arises in this case as the TTP does not know when the information can be deleted, and consequently the TTP's storage will be occupied by much already-succeeded transaction information. One solution is to add variables indicating that both parties have received the aborted/recovered messages: *receiptA* and *receiptB*. When these two variables both turn to be true, meaning that both parties have claimed they get the aborted/recovered messages, TTP can safely delete all information about this protocol run.

5.1.4. Unreliable channels between Alice/Bob and the TTP

In Section 3, we have assumed resilient channels between Alice/Bob and the TTP. So what if those channels are unreliable? Cases of failures are as following:

Case 5.1. One of the two channels (say the channel between Alice and TTP) is broken and message in it is lost when the TTP is sending abort confirmation or recovered messages. Alice can still get the proper messages by invoking the inquiry sub-protocol.

Case 5.2. The channel is broken and message in it is lost when either party (say Alice) is submitting a request. Then, the requesting party is kept waiting while TTP does not know anything about it. In this case, the 2-phase communication with message acknowledgement needs to be used when either party is submitting request to overcome the network unreliability.

One may ask why not use the ftp-get operation as in Zhou and Gollmann [9]. Problem will arise as discussed in Section 5.1.3 that TTP need to forever store information of every protocol run.

5.2. Related work

As we mentioned in Section 1, there are only few efforts invested into dependable exchange protocol. The SbML logging method by Liu et al. [2] works in protocols with online TTP, e.g., the protocol by Zhou and Gollmann [9]. But when it comes to offline TTP, fairness after failures can be potentially broken. Cases of broken fairness are as following:

Case 5.3. After Alice sends out the first message, her system fails; when Bob gets the message, he can invoke the recover sub-protocol to get the final expected messages; if Alice fails to recover her system before TTP's recovered messages arrive, her fairness will be broken. So simply using their logging method is not enough to guarantee fairness.

Case 5.4. The offline TTP has not logged the variables: *recovered* and *aborted*, if system fails at TTP after a successful abort operation requested by Alice; at this time, Alice has quitted the transaction since her request has been confirmed; but if Bob submits a recover request after TTP recovers, TTP will recover the transaction and send proper recovered messages to Alice and Bob; in this case, the message cannot arrive at Alice, so fairness for Alice is broken.

The work by Ezhilchelvan and Shrivastava [10] has not proposed fault-tolerant protocol with offline TTP for asynchronous channels.

5.3. Future work

To be more precisely about effect of every factor in the protocol like network/system reliability, honesty of both parties, etc., we are modeling the exchange protocol in rewriting logic based on the work by Chadha et al. [18] and trying to propose a new fair mechanism design (recently proposed problem by Sandholm and Wang [19]) method. Then, we can study how we can build automatic exchange agents for real-world applications.

Acknowledgements

The authors thank Tao Wang, Botong Xu for instructive discussions. Furthermore, we thank the anonymous reviewers of WICS'05 and Computer Communications for valuable comments.

References

[1] H. Wang, H. Guo, M. Lin, J. Yin, Q. He, J. Zhang, Dependable transaction for electronic commerce, in: Proceedings of 3rd International Workshop on Internet Communications Security (WICS'05), Singapore, Lecture Notes in Computer Science, vol. 3482, Springer-Verlag, Berlin, 2005, pp. 691–700.
[2] P. Liu, P. Ning, S. Jajodia, Avoiding loss of fairness owing to process crashes in fair data exchange protocols, in: Proceedings of the IEEE International Conference on Dependable Systems and Networks, Workshop on Dependability despite Malicious Faults, IEEE Computer Society Press, New York, 2000, pp. 631–640.
[3] C.H. Wang, Untraceable fair network payment protocols with off-line TTP, in: Advances in Cryptology – ASIACRYPT 2003, Taipei, Lecture Notes in Computer Science, vol. 2894, Springer-Verlag, Berlin, 2003, pp. 173–187.
[4] H. Pagnia, F.C. Gartner, On the impossibility of fair exchange without a trusted third party, Tech. Rep. TUD-BS-1999-02 (March), Darmstadt University of Technology, 1999.
[5] N. Asokan, M. Schunter, M. Waidner, Optimistic protocols for fair exchange, in: Proceedings of the Fourth ACM Conference on Computer and Communications Security, ACM Press, New York, 1997, pp. 6, 8–17.
[6] S. Micali, Certified e-mail with invisible post offices (Available from author), an invited presentation at the RSA'97 conference, 1997.
[7] J.M. Park, E.K.P. Chong, H.J. Siegel, Constructing fair-exchange protocols for E-commerce via distributed computation of RSA signatures, in: Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing, ACM Press, New York, 2003, pp. 172–181.
[8] Y. Dodis, L. Reyzin, Breaking and repairing optimistic fair exchange from PODC 2003, in: Proceedings of the 2003 ACM Workshop on Digital Rights Management, ACM Press, New York, 2003.
[9] J. Zhou, D. Gollmann, A fair non-repudiation protocol, in: Proceedings of 1996 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Silver Spring, MD, 1996, pp. 55–61.
[10] P. Ezhilchelvan, S. Shrivastava, Systematic development of a family of fair exchange protocols, in: Proceedings of 17th IFIP WG 11.3 Working Conference on Database and Applications Security, Colorado, USA, 2003.
[11] H. Wang, H. Guo, Achieving fairness in wireless environment, Proceedings of IEEE 6th CAS Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication, vol. 1, IEEE Press, Shanghai, China, 2004, pp. 117–120.
[12] S. Gurgens, C. Rudolph, H. Vogt, On the security of fair non-repudiation protocols, in: Proceedings of 2003 Information Security Conference, Lecture Notes in Computer Science, vol. 2851, Springer-Verlag, Berlin, 2003, pp. 193–207.
[13] N. Asokan, V. Shoup, M. Waidner, Asynchronous protocols for optimistic fair exchange, in: Proceedings of 1998 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Silver Spring, MD, 1998, pp. 86–99.
[14] H. Vogt, H. Pagnia, F.C. Gartner, Modular fair exchange protocols for electronic commerce, in: Proceedings of the 15th Annual Computer Security Applications Conference, IEEE Computer Society Press, Silver Spring, MD, 1999.
[15] S. Kremer, O. Markowitch, J. Zhou, An intensive survey of fair non-repudiation protocols, Comput. Commun. 25 (17) (2002) 1606–1621.
[16] H. Wang, H. Guo, Fair payment protocols for e-commerce, in: Proceedings of Fourth IFIP Conference on e-Commerce, e-Business, and e-Government (I3E'04), Building the E-Society: E-Commerce, E-Business and E-Government, Kluwer Academic Publishers, Dordrecht, 2004, pp. 227–245.
[17] H. Wang, H. Guo, M. Lin, J. Yin, A fair item-item exchange protocol satisfying newly introduced requirements, in: Proceedings of International Conference on Information Technology and Applications (ICITA05), vol. 2, Australia, IEEE Computer Society Press, Silverspring, MD, 2005, pp. 502–507.
[18] R. Chadha, J. Mitchell, A. Scedrov, V. Shmatikov, Contract signing, optimism and advantage, in: CONCUR 2003 – Concurrency Theory, 14th International Conference, Lecture Notes in Computer Science, vol. 2761, Springer-Verlag, Berlin, 2003, pp. 366–382.
[19] T. Sandholm, X. Wang, (Im)possibility of safe exchange mechanism design, in: R. Ram (Ed.), Proceedings of Eighteenth National Conference on Artificial Intelligence (AAAI XVIII), AAAI Press, Menlo Park, CA, 2002.

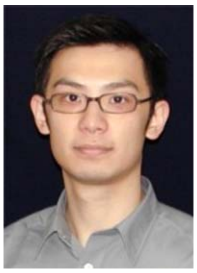
Hao Wang, Ph.D. candidate, School of Computer Science and Engineering, South China University of Technology. Main research interests include fair exchange in electronic transaction, game-based mechanism design in multi-agent systems (http://member.acm.org/~iswanghao for more details).

Heqing Guo, Professor, School of Computer Science and Engineering, South China University of Technology. Main research interests include network-based systems, electronic commerce.

Manshan Lin, Ph.D. candidate, Professor, School of Computer Science and Engineering, South China University of Technology. Main research interests include web services and communications.

Jianfei Yin, Ph.D. candidate, Professor, College of Information Engineering, Shenzhen University, Shenzhen 518060, China. Main research interests include model-driven system development.

Qi He, Ph.D. student, School of Computer Engineering, Nanyang Technological University. Main research interests include web-based systems.


Jun Zhang, Assistant professor, School of Computer Engineering, Nanyang Technological University. Main research interests include indexing techniques and query optimization in spatial and spatio-temporal databases and warehouses (http://www.ntu.edu.sg/home/jzhang/ for more details).