# Generic non-repudiation protocols supporting transparent off-line TTP

**Article** *in* Journal of Computer Security · November 2006

DOI: 10.3233/JCS-2006-14504 · Source: DBLP



**1 author:**

Guilin Wang
Huawei Technologies
**117** PUBLICATIONS **1,512** CITATIONS

*Book Title*
*Book Editors*
*IOS Press, 2003*



# Generic Non-Repudiation Protocols Supporting Transparent Off-line TTP

Guilin Wang [1]
*Institute for Infocomm Research ($I^2R$)*
*21 Heng Mui Keng Terrace, Singapore 119613*

**Abstract.** A non-repudiation protocol enables the *fair* exchange of an electronic message and an irrefutable digital receipt between two mistrusting parties over the Internet. That is, at the end of any execution instance of such a protocol, either both parties obtain their expected items or neither party does. In this paper, we first argue that it is really meaningful in practice to exploit *generic* fair non-repudiation protocols with *transparent* off-line trusted third party (TTP). Namely, in those protocols, each involved party could use *any* secure digital signature algorithm to produce non-repudiation evidences; and the issued evidences are the same regardless of whether the TTP is involved or not. Then, we present such a fair non-repudiation protocol to overcome some limitations and shortcomings in previous schemes. Technical discussions are provided to show that our protocol is not only secure but also the most efficient solution, compared with existing non-repudiation protocols. In addition, some potential extensions are also pointed out.

**Keywords.** non-repudiation, certified e-mail, fair exchange, security protocol

## 1. Introduction

### *1.1. Background*

Non-repudiation services are essential for many electronic transactions, where irrefutable evidences need to be generated, exchanged, and validated via computer networks. After the completion of such a transaction, each involved party should obtain the expected items. Therefore, if any dishonest party denies his/her participation in a specific transaction, others can refute such a claim by providing the related evidences to a judge.

A non-repudiation protocol allows two potentially mistrusting parities to exchange an electronic message together with the evidence of origin (EOO) and the corresponding evidence of receipt (EOR) over the Internet in a *fair* way, i.e., each party gets the other's item(s), or neither party does. To simplify the description, we assume that the sender Alice wants to deliver an electronic message $M$ to the receiver Bob under the guarantee that Bob can access the content of $M$ if and only if she can obtain an irrefutable receipt EOR from Bob showing that Bob has already received $M$. At the same time, Bob also

[1]Correspondence to: Guilin Wang, Systems & Security Department, Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore 119613. Tel.: +65 6874 1954; Fax: +65 6775 5014 ; E-mail: glwang@i2r.a-star.edu.sg. Preliminary version of this paper appears in the Proc. of *IWAP'05* [54].

needs to be convinced that he is able to get the message $M$ and EOO from Alice when he issues such an undeniable receipt to Alice. While non-repudiation evidences can be provided by some standard cryptographic primitives such as digital signatures, fairness is much more difficult to be achieved [33,30,9,48].

Technically speaking, non-repudiation protocol belongs to a wider topic: fair exchange. Namely, how to design a protocol that allows two (or multiple) mutually mistrusted parties to exchange their digital items over public computer networks in a fair way. More specifically, we require that *any* possible execution of such a protocol must end with one of the following two results (rather than others): (a) All parties (could) obtain their expected items; (b) No party can get the item he/she expects. In other words, a dishonest party (or a coalition of colluding parties) cannot obtain the expected items from a honest party in a cheating way such that the honest parties are unable to get the corresponding items.

Actually, fair exchange includes the following different but related issues: non-repudiation protocols [57,58,37,30,33], certified e-mail systems [18,56,23,6,31,40], fair exchange of digital signatures [17,3,7,5], contract signing protocols [2,26,40,8,55], and e-payment solutions [12,36,51,52]. In a certified e-mail scheme, a sender Alice wants to deliver a digital message to a receiver Bob with the guarantee that Bob can access the content of the e-mail *if and only if* Alice obtains an irrefutable receipt from Bob. So, we know that the purposes of non-repudiation protocols and certified e-mail schemes are almost the same, except that a certified e-mail scheme may not provide evidence of origin. For more references and discussions on the relationships among those variants of fair exchange, please refer to [2,4,47,33]. In this paper, we shall focus on the topic of non-repudiation protocols and certified e-mail schemes in the traditional two-party setting, i.e., there are two users who want to exchange digital items (under the possible help of a trusted third party).

### *1.2. Our Contribution*

In this paper, we propose an efficient and secure generic fair non-repudiation protocol supporting transparent off-line trusted third party (TTP), which overcomes some limitations and security weaknesses in existing schemes. Specifically, the proposed protocol satisfies the following six desirable properties.

(1) **Generic Construction**: Each involved party can *independently* exploit *any* (secure) standard signature scheme to generate non-repudiation evidences. In particular, two involved parties are *not* required to use the same signature scheme.
(2) **Transparent TTP**: The formats of generated non-repudiation evidences are the same regardless of whether the TTP is involved or not in the protocol execution.
(3) **Off-line TTP**: The TTP is involved only in the situation where one party is cheating or the communication channel is interrupted. So it could be expected that the TTP is asked to settle unfair situations *rarely*, due to the fact that fairness are always achieved, i.e., cheating is not beneficial to the cheater.
(4) **Fairness**: At the end of any protocol execution, *either* the sender Alice obtained the evidence of receipt (EOR) and the receiver Bob got the corresponding message as well as the evidence of origin (EOO) simultaneously, *or* none of them can get those items. This property implies that even a dishonest party who tries to cheat in any possible way cannot get an advantage over the honest party.

(5) **Timeliness**: At any possible state in a protocol execution, each honest party can terminate the protocol without loss of fairness *unilaterally*, i.e., there is no need for any cooperation of the other (potentially malicious) party.
(6) **High Performance**: In the normal case, our non-repudiation protocol is finished by exchanging 3 message flows and performing 6 asymmetrically cryptographic operations. This implies our solution is the *most* efficient scheme, compared with the art-of-the-state work [40,30].

In the above list, fairness is the most essential security requirement for all non-repudiation protocols, which should provide EOR and EOO to the sender and the receiver respectively. In later disputes, EOR consists of the main evidence proving non-repudiation of receipt (NRR), while EOO proving non-repudiation of origin (NRO). However, as we mentioned before, EOO is not provided in some certified e-mail schemes [6,40]. At the same time, timeliness is also important in practice since it enables both parties to quit or finish a protocol run at any moment in the protocol execution. Actually, some schemes [57,58,40] only meet *weak* timeliness, i.e., one party may need to wait for the other party's answer in a given period but not forever. An optional security requirement is *confidentiality*, i.e., except the two parties involved, nobody else (including the TTP) cannot know the content of the delivered message. Our protocol can be adapted in a simple way to meet confidentiality.

Here, we want to stress that other properties on basic features and efficiency are also very meaningful in real-world systems. First of all, generic constructions are undoubtedly important in practice since in the real world users almost inevitably exploit different signature algorithms. Second, to reduce the running cost of the TTP, it is natural to choose non-repudiation protocols supporting transparent off-line TTP due to the following four reasons. (a) It could be expected that the TTP is only involved to settle unfair situations *rarely*, since fairness implies that cheating is not beneficial to the cheater, except some essentially unharmful disturbance to the honest party. (b) In a system with transparent TTP the non-repudiation evidences may become relatively simple since they are only some standard signatures generated by the involved parties. (c) The TTP is free from the burden of generating affidavits as the whole or part of non-repudiation evidences. This also means the TTP's liability in our solution is further reduced, and hence the cost for the TTP's service could be cut accordingly. (d) Exploiting transparent TTP is also helpful to avoid bad publicity in the scenario of e-commerce, because the intervention of the TTP perhaps results from the communication failure instead of a party's dishonest behavior, as pointed out in [37]. Finally, as practice-oriented systems non-repudiation protocols and certified e-mail schemes should be implemented as efficient as possible on both aspects of computation and communication, while meeting the desired security properties.

### *1.3. Organization*

The rest of the paper is organized as follows. Section 2 reviews related work in fair exchange and the security problems in previous protocols. In Section 3, we introduce the assumptions and notations. Then, Section 4 presents the new protocol in detail. After that, we discuss the security of our protocol in Section 5, and point out some extensions in Section 6. Finally, we compare our scheme with existing ones in Section 7, and conclude the paper in Section 8.

## 2. Related Work

The first fact we need to know is that realistic implementations of fair exchange have to rely on a trusted third party (TTP), due to the following two reasons. On the one hand, as early as in 1980, Even and Yacobi [21] proved that it is *impossible* to realize *fairness* in a *deterministic* two-party fair exchange protocol. This result can be informally explained as follows. Fairness implies that from the initial fair state, in which neither party has the expected items, we need to achieve the desired final fair state, in which both parties obtained those items. However, information exchange in computer networks is asymmetric and non-simultaneous, so there must be an unfair intermediate state we have to go through.

On the other hand, both of the two known approaches for direct fair-exchange without TTP are not practical for the majority of applications. Gradual exchange solutions [27,22,17] can only achieve *computational fairness* with the meanings that the advantage of one party over the other can be significantly small (but not negligible), under the assumption that the two involved parties have equivalent or related computational resources. This may be not a reasonable assumption in many practical applications. Moreover, computational fairness is not only weaker than the above introduced standard fairness (called *strong fairness* in [33]), but also costs at the price of expensive computation and communication since the two parties usually have to exchange their items "bit-by-bit" for many rounds. To avoid the assumption of equivalent computation in gradual exchange solutions, probabilistic fair exchange protocols [11,35] have been proposed. Those protocols can achieve *probabilistic fairness*, which means that any execution of such a protocol can realize standard fairness with an overwhelming probability $1 - \varepsilon$, where $\varepsilon$ should be set as a negligible value. However, to provide an adequate security level those probabilistic protocols have to repeat some successive rounds for a large numbers. For example, Markowitch-Roggenman protocol (refer to [35] or Section 3 of [33]) achieves the security level $\varepsilon = 1/n$ at the price of $2n + 2$ rounds of transmissions. Obviously, such a cost of communication (and computation) is unacceptable in many environments.

Therefore, to get practically efficient solutions the best one can hope for is to reduce the TTP's involvement as much as possible. Actually, the TTP can be resorted to fair exchange protocols in anyone of the following three different ways [58]: in-line, on-line or off-line.

### 2.1. In-line and On-line TTP Protocols

A relatively simple but unsatisfactory approach is to exploit an *in-line* TTP as in the previous schemes [15,18,56]. Specifically, by treating the TTP as a *delivery authority*, two involved parties first generate and submit proper digital items to the TTP, then the TTP checks the correctness of those items and forwards each item to the corresponding recipient. The shortcoming is that the in-line TTP is likely to become a bottleneck in those systems, due to the fact that it is involved in each step of every exchange. An ingenious solution is proposed in the protocol of Zhou and Gollmann [57] by using an *on-line* TTP. That is, in their scheme only a short symmetric key is forwarded and notarized by the TTP, who serves as *light-weight notary*, while the whole message in ciphertext is directly transferred to the receiver. However, both of the above two approaches are actually in-

efficient and expensive in the real world because the TTP must take part in the protocol execution for every exchange (though may not in every step). Just due to this reason, all those protocols are sometimes called solutions with on-line TTP, for simplicity. However, the point is that the TTP needs to be paid in some way for all provided services.

### 2.2. Off-line TTP Protocols

Another paradigm is to design non-repudiation protocols with an *off-line* TTP [58]. Actually, such protocols are also called *optimistic* [2,4], since the TTP is not invoked in the execution of exchange unless one of the two parties misbehaves or the communication channel is out of order. Generally speaking, optimistic fair exchange requires that the digital items to be exchanged have special properties, i.e., *generatability* or *revocability* [2]. That is, to resolve possible unfair conflicts the TTP should be able to either generate or revoke such digital items, when some auxiliary information is given. As observed by Vogt in [51], most of optimistic fair exchange protocols in the literature have been proposed by exploiting generatable items, instead of revocable items. In the following, therefore, we roughly classify those optimistic protocols into three types according to the basic technical methods used to produce generatable items. At the same time, we also review their security shortcomings and some other disadvantages.

To realize an off-line TTP in their non-repudiation protocol, Zhou and Gollmann [58] exploited the following basic idea: split the non-repudiation evidences into two parts, one for the ciphertext $C$ of a message $M$ and the other for the submission of a secret key $K$, which is used to encrypt the message $M$. After the exchange of ciphertext $C$ together with the evidences of origin and receipt on it, the sender Alice sends the receiver Bob the secret key $K$ together with the evidence of origin on it. If those items are correct, the receiver Bob returns his receipt on $K$ showing that he already received the secret key. However, Bob may maliciously refuse to reveal such a receipt to Alice. In this case, Alice needs to submit relevant information to the TTP so that the TTP can issue an affidavit showing that Alice properly submitted the key $K$, and forward the secret key $K$ to the receiver Bob. Except supporting off-line TTP, this constructive method does not need to use any specific features of underlying digital signature algorithms to generate non-repudiation evidences. Hence, many protocols [59,60,13,23,36,37,33,30] have been proposed following their paradigm.

Later, Gürgens et al. [29,30] intensively studied the security of those schemes. According to their results, the desired fairness in most of those protocols may not be guaranteed under some subtle but reasonable attacks: (1) The protocol labels used in [57,58] need to be modified in a small but important way; (2) The protocols proposed in [36,37,33] enable a dishonest receiver to access the message without issuing a receipt to the sender (so unfair for the sender)[1]; and (3) There are two potential weaknesses in the solution [59,60] and its improvement [13], since the sender Alice may submit incorrect encrypted key or reuse of labels and keys and then lead to unfairness for the receiver Bob. To avoid those security shortcomings, Gürgens et al. further presented a new non-repudiation protocol in [30]. At the same time, Wang et al. [53] pointed out that both

[1] More specifically, Gürgens et al. just identified such an attack against the scheme [33]; but we notice that the protocols presented in [36] and [37] are vulnerable to similar attacks. In addition, Cathalo et al. [14] also showed that those two schemes are insecure by cryptanalyzing their common building block, a verifiably committed signature scheme based on GPS and RSA.

the FPH scheme [23] and its improvement [41] are insecure. The most serious security flaw is that the receiver can access the delivered message without issuing a receipt to the sender, since the receiver can get the secret key from the TTP by colluding with one of his friends. In summary, all the above mentioned schemes except the GRV protocol [30] have some security weaknesses in different senses.

In [39,40], Micali proposed simple certified e-mail schemes with *transparent* TTP (*invisible post office*, in his terminology). His basic idea is to use the technique of *double encryptions*. That is, to send Bob a message $M$ Alice first encrypts it as ciphertext $C$ under Bob's public key and then re-encrypts $C$ together with the identities of Alice and Bob as $Z$ under the TTP's public key. In the beginning, Alice sends $Z$ and her signature on $Z$ to Bob as the evidence of origin. If Alice's signature is correct, Bob then returns his signature on $Z$ as the receipt for Alice. Finally, upon receiving Bob's valid signature Alice reveals $C$ to Bob. However, if dishonest Alice refuses to do so, the receiver Bob can get $C$ from the TTP directly by submitting $Z$ together with corresponding signatures.

However, we remark that Micali's schemes have two weaknesses. First of all, his schemes are inefficient (though simple) if the delivered message is long, since messages here are doubly encrypted under asymmetric instead of symmetric encryption algorithms. About this shortcoming, Micali did pointed out that messages could be encrypted alternatively with a symmetric key $K$, while $K$ is encrypted again under the receiver's public key. Actually, this idea is adopted by most of existing non-repudiation protocols [57,58,59,60,37,33,31,30]. The point is that this conversion is *not* trivial, according to the relevant researches [29,30,48]. Moreover, there is a potential attack again his certified e-mail schemes [39,40], where the sender Alice could cheat the receiver Bob by mixing identities of different TTPs. Consequently, Alice will get a valid receipt, but Bob perhaps cannot receive the message. Such an attack is effective in practice since it seems unreasonable to assume that Bob knows the existence of all TTPs and may contact each TTP one by one for help. To improve the efficiency of Micali's schemes, Imamoto and Sakurai [31] designed a new protocol supporting transparent TTP. But this protocol is also suffers from the same attack against Micali's schemes.

Another method to realize fair exchange is to exploit some proper cryptographic tools, such as *verifiably encrypted signatures* (VES) [7,5,6,42], *verifiably committed signatures* (VCS) [12,36,37,45,19], or *verifiable escrows* [3,4]. The basic ideas behind those cryptographic primitives are similar, as explained below. To get an expected digital item from Bob, Alice first sends Bob her committed signature on a given message, which may be a partial signature or a full signature encrypted under the TTP's public key. The key point is that such a committed signature is *verifiable* in the sense that both Bob and the TTP can check whether it indeed corresponds to Alice's full signature, via interactive or non-interactive proofs. If the verification is ok, Bob returns his digital item to Alice. After obtaining the expected item from Bob, Alice finally reveals her whole signature to Bob. In case Alice refuses to do so after getting Bob's item, however, Bob can get the whole signature from the TTP since only the TTP (except Alice) has the ability to recover Alice's full signature, by generating the other part of partial signature on behalf of Alice or decrypting Alice's encrypted signature directly, or something like that.

The disadvantage of this approach is that to get efficient constructions, one has to exploit the specific features of different signature algorithms, such as did in [7,5,6,36, 37,45,19]. That is, the schemes constructed in this way are usually *not generic* proto-

cols. On the other hand, though Asokan et al.'s techniques [3,4] can be applied to a variety of signature schemes, the overheads of computation and communication are relatively too expensive, since the inefficient cut-and-choose technique [24] is used to prove the correctness of committed signatures. In addition, we remark that even though those primitives (VES, VCS etc.) have been well studied in cryptography, the corresponding fair exchange protocols may still have security problems. For example, Bao [9] identified a collusion attack against the fair-exchange schemes for Schnorr and ElGamal signatures proposed in [5]; Dodis and Reyzin [19] broke Park et al.'s e-payment protocol [45] based on the RSA signature [46]. In addition, both Ateniese and Nita-Rotaru [6], and Nenadić et al. [42] designed certified e-mail schemes based on verifiable encrypted RSA signature, though their concrete implementation techniques are different. However, there are some common shortcomings in the AN scheme [6] and NZS schemes [42]: (1) Those are specific constructions; (2) Do not support timeliness; and (3) The receiver in those schemes is required to register with the TTP before using the system, similar as in [12,45,55]. Moreover, the AN scheme does not provide the evidence of origin (EOO), but in some scenarios the receiver may need EOO to prove the origin of a message.

We now summarize the above brief survey on non-repudiation protocols and certified e-mail schemes as follows. First, most of them are *generic* constructions except the schemes in [37,6,42]. Second, most of them have some security weaknesses except the AN scheme [6], the GRV scheme [30], and the NZS scheme [42]. Finally, only the protocols in [37,6,31,40,42] support transparent TTP, but all of those schemes have some limitations or security flaws (check Table 1 in Section 7 for comparison). Consequently, it is desirable to propose a new generic fair non-repudiation protocol with transparent off-line TTP to avoid those shortcomings. This paper is motivated to propose such a protocol satisfying the six properties listed in Section 1.2.

## 3. Assumptions and Notations

In the framework of fair exchange, three kinds of communication channels are typically distinguished: *unreliable* channels, *resilient* channels, and *reliable* channels. Messages inserted into an unreliable channel may be lost, so the expected recipient may be unable to get the message forever. A resilient channel can deliver a message to the expected recipient after a finite but unknown delay, i.e., such a message will arrive at its destination eventually though may be delayed arbitrarily. Resilient channels are also called *asynchronous* networks [33,51]. In reliable channels, sometimes named as *synchronous* or *operational* networks, messages can be delivered to the expected recipient within a known and finite delay. Usually, open, heterogeneous networks like the Internet cannot be supposed to be reliable channels. In real life, however, such reliable communications could be implemented via dedicated networks or normally dependable computer networks compensated by other traditional communication means, such as fax, telephone, or express mail service etc.

Of course, protocols with weak assumptions on communications may be more desirable since they can be applied in more environments. However, this also means we need to pay more overheads on the communication and computation (including more cryptographic operations) if the same security requirements are to be achieved. In our protocol, we assume that the TTP is linked with both Alice and Bob by resilient communication

channels, while the communication channel between Alice and Bob is unreliable. Actually, this is a common treatment in most of non-repudiation protocols, since it is reasonable to assume the communications to and from the TTP are more robust than that between users. Due to this reason, we do not compare existing non-repudiation protocols in this aspect (Check Table 1 in [33] for such a comparison).

In our system, we assume that each party has a unique identity. The identities of the sender Alice, the receiver Bob, and the TTP, are denoted by $A$, $B$, and $T$, respectively. We suppose that Alice, Bob and the TTP can all sign messages using (any) secure digital signature schemes, which are existentially unforgeable against an adaptive chosen message attack as defined by Goldwasser et al. in [28]. Party $X$'s signature on a message $m$ is denoted by $S_X(m)$, which can only be produced by party $X$ who knows the signing key. However, $S_X(m)$ is publicly verifiable, i.e., anybody can validate it by using the publicly known signature verification key of party $X$.

Furthermore, let $(E_T(\cdot), D_T(\cdot))$ be the TTP's public key encryption/decryption algorithm pair, which is CCA2 secure, i.e., secure against adaptive chosen ciphertext attack as defined by Dolev et al. in [20]. To emphasize a random number $R$ is utilized to encrypt message $m$, we write $c = E_T^R(m)$. Note that with the pair $(m, R)$, anybody can verify whether a string $c$ is the ciphertext of $m$ with respect to the TTP's public key. We also explicitly assume (as it is generally true) that, from the ciphertext $c$, the TTP can recover not only the message $m$ but also the randomness $R$. For simplicity, we call this property *randomness recoverability* and denote this fact as $(m, R) = D_T(c)$. For example, this property is satisfied by the OAEP series of encryption schemes [10,50] (especially the OAEP-RSA [25], refer to Appendix A), but not by the Cramer-Shoup cryptosystem [16].

In addition, $(E_K(\cdot), D_K(\cdot))$ denotes a pair of secure symmetric encryption and decryption algorithms with respect to a secret key $K$, such as AES in CBC mode. When Alice wants to deliver message $M$ to the recipient Bob, $M$ will be encrypted as $C = D_K(M)$. Finally, let $H(\cdot)$ be a collision-resistant one-way hash function. More notations are listed as follows.

- $M$: message delivered to the receiver Bob by the sender Alice.
- $K$: secret key used to encrypt message $M$.
- $f_K, f_{EOO}, f_{EOR}, f_{AT}, f_{Rec}, f_{Con}$: publicly known *unique* flags that indicate distinct purposes of different messages in our protocol. We also assume those flags containing a unique protocol identifier as a prefix or suffix to avoid potential interleaving attacks from different (non-repudiation) protocols. Such a protocol identifier also specifies the underlying algorithms used.
- $L = H(A, B, T, HC, HK)$: unique label to identify a protocol instance. That is, label $L$ means that Alice sends message $M$ to Bob (with/without the TTP's help), where $M$ is determined by a ciphertext $C$ and a symmetric key $K$ such that $M = D_K(C)$, $HC = H(C)$ and $HK = H(K)$.
- $EK = E_T^R(f_K, L, K)$: encrypted secret key, which is the ciphertext of $(f_K, L, K)$ under the public encryption key of the TTP by selecting a random number $R$.
- $EOO = S_A(f_{EOO}, L, EK)$: evidence of origin, showing that Alice sent a message $M$ to Bob, if both $EOO$ and $EK$ are valid.
- $EOR = S_B(f_{EOR}, L, EK)$: evidence of receipt, showing that Bob received a message $M$ from Alice, if both $EOR$ and $EK$ are valid.

- $AT = S_A(f_{AT}, L)$: abort token issued by Alice to cancel the protocol run indexed by label $L$.
- $Rec = S_B(f_{Rec}, L, EK)$: recover request from Bob to resolve the protocol run indexed by label $L$.
- $Con = S_T(f_{Con}, L, AT)$: confirmation issued by the TTP to Alice showing that the TTP has already approved Alice's abort request on the protocol run indexed by label $L$.

**Remark 1**: Here, we assume that the secret key $K$ and ciphertext $C$ are long enough (e.g. at least 128 bits) so that $H(K)$ and $H(C)$ do not reveal any useful information on $K$ and $C$. If this is not true, one can pad random numbers to $K$ and $C$ with proper lengths before hashing them.

## 4. The Proposed Non-Repudiation Protocol

The basic idea of our protocol can be briefly explained as follows. Alice first sends Bob $(C, EK, EOO)$, where the evidence for non-repudiation of origin (NRO) is mainly defined by the concatenation of $EOO$ and $(K, R)$, i.e., the correct content of $EK$. So, to get the committed pair $(K, R)$, Bob has to submit his signature $EOR$ to Alice or the TTP. At the same time, if Alice prepared $EK$ improperly, the corresponding $EOR$ issued by Bob cannot be interpreted as a valid evidence for non-repudiation of receipt (NRR). This idea is partially inspired by the previous work in [40,8,30]. However, as we mentioned before, our protocol is different from though related to those protocols (Check Table 1 for comparison): (1) It overcomes the weaknesses in Micali's certified e-mail schemes [40]; (2) It supports the transparent TTP and has better performance, while the GRV scheme [30] does not support transparent TTP and is less efficient; (3) It is a non-repudiation protocol, while the scheme proposed in [8] is for contract signing where the contract is already known for two parties involved. Another challenge is that many subtle problems should be dealt properly, due to the well-known fact that security protocols [1], including non-repudiation protocols [29,30,48], are notoriously error prone.

Like most non-repudiation protocols, our scheme consists of a dispute resolution policy, and three sub-protocols, i.e., the exchange protocol, the abort protocol, and the recovery protocol. The *dispute resolution policy* defines the format of evidences for non-repudiation of origin (NRO) and non-repudiation of receipt (NRR), and the procedures how a judge settles potential disputes over NRO or NRR between different parties. The *exchange protocol* is the main protocol, which is the unique protocol that needs to be executed jointly by the sender Alice and the receiver Bob in the *normal situation*, i.e., both involved parties behave honestly according to the protocol specification and the communication channel is in order. In abnormal situations, however, the *abort protocol* and the *recovery protocol* are further run to achieve fairness under the help of the TTP for the sender Alice and the receiver Bob, respectively. Specifically, the abort protocol allows the sender Alice to cancel a protocol instance in the following situations: (1) Bob does not respond at all; (2) Bob does not respond correctly or timely; (3) The communication channel between Alice and Bob interrupts; or (4) Alice changed her mind so she does not want to complete this protocol run with Bob any more. Similarly, the recovery protocol protects the receiver Bob from the possible cheating of Alice or the failure of communications.

### 4.1. The Exchange Protocol

Assume that Alice wants to deliver a message $M$ to the receiver Bob with the guarantee that Bob can access the message $M$ if and only if she can obtain a receipt from Bob. To this end, the sender Alice and the receiver Bob run the following exchange protocol (Fig. 1) jointly.

| | |
|---|---|
| (e1). | $A \longrightarrow B$: $f_{EOO}, A, B, T, C, HK, EK, EOO = S_A(f_{EOO}, L, EK)$ |
| | `if` $B$ gives up `then` quits |
| (e2). | $B \longrightarrow A$ : $f_{EOR}, EOR = S_B(f_{EOR}, L, EK)$ |
| | `if` $A$ gives up `then` runs the abort protocol |
| (e3). | $A \longrightarrow B$ : $f_K, K, R$ |
| | `if` $B$ gives up `then` runs the recovery protocol |

**Figure 1.** The Exchange Protocol

We now further explain the exchange protocol in detail as follows. Firstly, Alice chooses a session key $K$ and a random number $R$ uniformly and independently, then computes $C = E_K(M)$, $HK = H(K)$, $HC = H(C)$, $L = H(A, B, T, HC, HK)$, $EK = E_T^R(f_K, L, K)$, and $EOO = S_A(f_{EOO}, L, EK)$. Then, Alice sends message flow (e1) to Bob. Upon receiving (e1), Bob first checks whether $(A, B, T)$ are correct identities. If yes, he sets label $L = H(A, B, T, H(C), HK)$, and verifies whether $EOO$ is Alice's valid signature on message $(f_{EOO}, L, EK)$. If this is not the fact or he would not like to respond Alice, Bob just gives up and quits this protocol instance without any liability. If $EOO$ is indeed valid, Bob could reply Alice by sending his signature $S_B(f_{EOR}, L, EK)$ as $EOR$ in step (e2), since he is convinced that (1) if $EK$ is correctly prepared, either Alice or the TTP can reveal $(K, R)$ to him; and (2) if $EK$ is incorrectly prepared, $EOR$ issued by himself is not a valid NRR evidence and so useless for Alice (and anybody else).

When message flow (e2) is arrived, Alice checks whether $EOR$ is Bob's signature on message $(f_{EOR}, L, EK)$. If this is true, Alice has gotten the evidence for non-repudiation of receipt (NRR) from Bob, i.e., NRR$= (A, B, T, M, K, R, EOR)$. Thus, she reveals the values of $(K, R)$ in message flow (e3). However, if Alice only received incorrect $EOR$ or does not receive the $EOR$ timely (according to her definition), she can run the abort protocol (see Section 4.2) to cancel this protocol instance.

Upon receiving message flow (e3), Bob checks whether $EK \equiv E_T^R(f_K, L, K)$ and $HK \equiv H(K)$. If both of those two equalities hold, Bob has obtained the evidence for non-repudiation of origin (NRO) from Alice, i.e., NRO$= (A, B, T, M, K, R, EOO)$. On the other hand, if $(K, R)$ is incorrect (i.e., $EK \neq E_T^R(f_K, L, K)$ or $HK \neq H(K)$), or Bob does not receive message flow (e3) at all, he can ask for the TTP's help by initiating the recovery protocol (see Section 4.3).

**Remark 2**: Here, we implicitly assume that both the validity and matching of all flags ($f_K$, $f_{EOO}$, $f_{EOR}$ etc) are checked (by the system automatically) in each step of the above exchange protocol, and other sub-protocols in Sections 4.2, 4.3, and 4.4. This is obviously necessary to assure that the two parties are running the same protocol with the same specified underlying algorithms. We do not mention this anymore in later description and discussion.

**Remark 3**: Note that the above exchange protocol does not protect the content of message $M$, since it is possible for an eavesdropping attacker to intercept both ciphertext $C$ and the pair $(K, R)$ and hence drive the message $M$ by computing $M = D_K(C)$. However, we can make some small changes in our exchange protocol such that the confidentiality for message $M$ is supported. One method is to require Alice encrypts $(K, R)$ by using Bob's public key, and then sends Bob the resulting ciphertext instead of $(K, R)$ itself in message flow (e3). In this way, only Bob (and perhaps the TTP, but no others) can derive message $M$, since an eavesdropper cannot obtain the secret key $K$ though the encrypted message $C$ could be intercepted. The TTP may be able to get message $M$ if it actively intercepted message flow (e1) and then obtained $C$ and $EK$. Then, the TTP gets $M$ by computing $M = D_K(C)$, where $K$ is decrypted from $EK$ by using its private key. However, the TTP maybe unlikely do so since it is a trusted party. In addition, this is different from the situation by submitting both $C$ and $EK$ to the TTP in the recovery protocol, since in the latter case even an honest but curious TTP can really decrypt $M$ from $C$. Anyway, we may consider this method supporting *weak confidentiality*, which could be used in the scenario where message $M$ is not so much private. To implement strong confidentiality, all message flows (e1), (e2) and (e3) can be encrypted symmetrically under another secret key $K'$. That is, Alice can choose $K'$ at the beginning and transfers it to Bob by encrypting $K'$ under Bob's public key and then attaching the resulted ciphertext into the first message flow. In this way, even the TTP cannot access the message $M$, since it cannot get the ciphertext $C$ though it may know the secret key $K$ if either the abort protocol or recovery protocol is executed. Note that this method does not add much overhead actually, since only two additional public key operations are needed to transfer the secret key $K'$, i.e., encrypting and decrypting $K'$.

### *4.2. The Abort Protocol*

After delivering message flow (e1), if the sender Alice does not receive the expected value of $EOR$ from Bob correctly or timely, she can execute the following abort protocol (Fig. 2) with the TTP and then cancel the protocol instance that was run together with the receiver Bob.

| | |
|---|---|
| (a1). | $A \longrightarrow T: \quad f_{AT}, A, B, T, HC, HK, E_T(AT = S_A(f_{AT}, L))$ |
| | `if` (abort token $AT$ is invalid) `then` stop |
| | `if` (`state=recovered`) `then` retrieve $EOR$ |
| | $\quad T \longrightarrow A: \quad f_{EOR}, L, EOR$ |
| | `if` (`state=aborted`) `then` retrieve $Con$ |
| | $\quad T \longrightarrow A: \quad f_{Con}, L, Con$ |
| | `else` set (`state=aborted`) and issue $Con$ |
| (a2). | $T \longrightarrow A: \quad f_{Con}, L, Con$ |
| (a3). | $T \longrightarrow B: \quad f_{AT}, A, B, T, HC, HK, AT = S_A(f_{AT}, L)$ |

**Figure 2.** The Abort Protocol

To cancel a protocol run indexed by label $L$, Alice first produces an abort token $AT = S_A(f_{AT}, L)$, computes $E_T(AT)$, and then sends message flow (a1) to the TTP. Upon receiving the abort request (a1), the TTP sets $L = H(A, B, T, HC, HK)$, decrypts $E_T(AT)$, and then checks whether the resulting plaintext is Alice's signature

on message $(f_{AT}, L)$. If this is not the fact, the TTP ignores the request. If $AT$ is valid, the TTP checks whether the label $L$ is recorded in its database. If this is true, it knows this protocol instance has been aborted or recovered, so the TTP just retrieves related items and then forwards them to Alice. Otherwise, the TTP first issues $Con = S_T(f_{Con}, L, AT)$ and then records the following information in its database: ($L$,`state=aborted`,$Con$) together with message (a1). After that, it forwards the message $(f_{AT}, A, B, T, HC, HK, AT)$ to Bob, and sends $(f_{Con}, L, Con)$ to Alice by showing that her abort request is approved. Note that in the message flow (a1) the abort token $AT$ is transferred in ciphertext. The purpose is to prevent Bob from intercepting $AT$ before the TTP accepts Alice's abort request.

**Remark 4**: Note that in our protocol, the functionality of $Con$ and $AT$ is different. $Con$ is just used to convince the sender Alice that the protocol instance indexed by $L$ is already aborted, i.e., the receiver Bob cannot get the secret key $K$ from the TTP by initializing the recovery protocol. In other words, $Con$ issued by the TTP does not consist of a part of non-repudiation evidences (otherwise, our protocol cannot support transparent TTP). However, the abort token $AT$ is a potential part of NRR evidences. That is, if the receiver Bob can show a valid abort token $AT$, a correct $EOR$ cannot be judged as a legal proof showing that Bob received message $M$ (see more details in Section 4.4).

### *4.3. The Recovery Protocol*

Similarly, if the receiver Bob has sent $EOR$ to the sender Alice but does not receive the expected $(K, R)$ from Alice correctly or timely, he can run the recovery protocol (Fig. 3) with the TTP, and then get the values of $(K, R)$ alternatively, if $EK$ is indeed valid.

| | |
|---|---|
| (r1). | $B \longrightarrow T:$ $f_{Rec}, A, B, T, HC, HK, EK, EOO, EOR, Rec$ |
| | `if` any of signatures $EOO, EOR, Rec$ is invalid `then` stop |
| | `if` (`state=aborted`) `then` retrieve $AT$ |
| | $T \longrightarrow B:$ $f_{AT}, L, AT$ |
| | `if` (`state=recovered`) `then` retrieve $(K, R)$ |
| | $T \longrightarrow B:$ $f_K, L, K, R$ |
| | `if` $EK$ is invalid `then` send an error message to Bob |
| | `else` set (`state=recovered`) |
| (r2). | $T \longrightarrow A:$ $f_{EOR}, L, EOR$ |
| (r3). | $T \longrightarrow B:$ $f_K, L, K, R$ |

**Figure 3.** The Recovery Protocol

When the TTP receives a recovery request message flow (r1) from Bob, it first sets $L = H(A, B, T, HC, HK)$, and then checks whether $EOO$, $EOR$ and $Rec$ are all correct signatures on the expected messages. If any of those signatures is invalid, the TTP ignores the request. Otherwise, the TTP further checks whether the label $L$ is recorded in its database. If this is true, it means this protocol instance has been aborted or recovered successfully, so the TTP just retrieves the related item and then forwards it to Bob. If $L$ is not recorded, the TTP needs to determine the validity of $EK$. $EK$ is called *valid* if and only if (1) $(f_K, L, K, R) = D_T(EK)$, where $K$ is a symmetric key and $R$ is

a random number; and (2) $H(K) \equiv HK$. That is, a valid $EK$ is the ciphertext of $(f_K, L, K)$ with respect to a random number $R$ under the TTP's public key, and the secret key $K$ is consistently committed by $HK$. If $EK$ is invalid [2], the TTP sends Bob an error message to tell him this fact. Otherwise, i.e., $EK$ is indeed valid, the TTP records ($L$,`state=recovered`,$(K, R)$) together with message flow (r1) in its database, and sends $EOR$ to Alice and $(K, R)$ to Bob respectively.

**Remark 5**: Note that if the recovery request is valid, the TTP needs to forward $EOR$ to Alice. If this is not the case, Bob can get $(K, R)$ from the TTP directly via running the recovery protocol and so thwart Alice getting the value of $EOR$. In addition, we stressed that in both of the above abort and recovery protocols the TTP is only provided with the hashed value $H(C)$ instead of the whole ciphertext $C$. In this way, our protocol achieves better privacy and efficiency: (1) Without the ciphertext $C$, a curious TTP cannot derive the message $M$ delivered between Alice and Bob; and (2) The communication overheads between the TTP and users are independent of the length of the message $M$ exchanged from Alice to Bob. At the same time, to provide the confidentiality of message $M$ the pair $(K, R)$ in the recovery protocol may need to be delivered from the TTP to Bob in ciphtertext, as we mentioned in the exchange protocol.

### *4.4. The Dispute Resolution Policy*

In some day after the completion of a protocol execution (with or without the TTP's participation), a judge may need to settle the following two types of dispute.

- **Repudiation of Origin**. If Alice denies having sent message $M$ to Bob, Bob could submit NRO=$(A, B, T, M, K, R, EOO)$ to a judge as a dispute resolution request. Then, the judge performs as follows:
  (1) Check that $(A, B, T)$ are identifies of Alice, Bob, and a TTP who provides services for fair non-repudiation exchange.
  (2) Compute $C = E_K(M)$, $L = H(A, B, T, H(C), H(K))$, and $EK = E_T^R(f_K, L, K)$.
  (3) Check whether $EOO$ is Alice's valid signature on message $(f_{EOO}, L, EK)$.
  (4) Accept Bob's claim if all above checks hold. Otherwise, reject Bob's claim.
- **Repudiation of Receipt**. Similarly, if Bob denies having received message $M$ from Alice, Alice could provide NRR= $(A, B, T, M, K, R, EOR)$ to a judge for dispute resolution. Then, the judge acts in the following way:
  (1) Check that $(A, B, T)$ are identifies of Alice, Bob, and a TTP who provides services for fair non-repudiation exchange.
  (2) Compute $C = E_K(M)$, $L = H(A, B, T, H(C), H(K))$, and $EK = E_T^R(f_K, L, K)$.
  (3) Check whether $EOR$ is Bob's valid signature on message $(f_{EOR}, L, EK)$.
  (4) If any of the above checks fails, reject Alice's claim.
  (5) Enquire whether Bob or the TTP has abort token $AT$. If a valid $AT$ is provided, reject Alice's claim. Otherwise, Alice's claim is accepted.

[2] Namely, $EK$ is *invalid* in all of the following cases: (a) The TTP cannot decrypt $EK$, i.e., $EK$ is not a well-formed ciphertext; (b) The derived plaintext of $EK$ is not equivalent to the expected tuple ($f_K$, $L$, $K$, $R$), i.e., the concatenation of two known values $(f_K, L)$ and two unknown values $(K, R)$; (c) $H(K) \neq HK$, i.e., the derived key $K$ is inconsistent with the committed key $HK$.

Note that when solving the repudiation of receipt, the judge is required to enquire whether Bob or the TTP holds a valid abort token $AT$ indexed by label $L$. The reason is that perhaps Alice obtained a valid $EOR$ as well as successfully aborted a protocol run in the following two scenarios: (a) After she aborted a protocol run, (honest) Alice gets a valid $EOR$ from Bob due to the communication delay; (b) After a valid $EOR$ has been received, dishonest Alice promptly launches the abort protocol. By doing so, Alice has got NRR, but Bob can only get the abort token $AT$, instead of the desired pair $(K, R)$, from the TTP. Hence, a valid $AT$ should counteract the power of correct but illegal $EOR$. However, it is a different story in the procedure of repudiation of origin, because valid NRO and $AT$ only imply Alice is trying to cheat: She revealed the pair $(K, R)$ to Bob as well as executed the aborted protocol! Therefore, in this case a valid NRO is exactly considered as the legal proof showing that Alice has delivered message $M$ to Bob, regardless of whether there exists a valid $AT$. Anyway, this abides by the commonsense: The abort tocken $AT$, as Alice's digital signature, cannot serve as a valid proof for Alice herself.

**Remark 6**: Note that our protocol is a *3-move* optimistic non-repudiation protocol supporting *both* transparent TTP and (strong) timeliness. That is, in our exchange protocol only 3 message flows are transferred between Alice and Bob. To the best of our knowledge, there exist only four previous 3-move optimistic schemes [23,31,40,51]. However, our protocol can be differentiated from those existing schemes in following senses. First of all, according to Wang et al.'s work [53] the FPH protocol [23] and its improvement [41] have security flaws. Furthermore, in our expertise it seems very difficult to repair the FPH protocol and its variant. Secondly, both the IS scheme [31] and Micali schemes do not respect (strong) timeliness, since no abort sub-protocols are provided at all. Finally, Vogt's protocol [51] does not support transparent TTP, since in his abstract protocol the TTP is assumed to be able to revoke some *revocable items* (e.g. e-cash) in order to maintain fairness. As Vogt observed, revocation is usually only possible in payment schemes. Actually, his scheme in [51] is such an e-payment system (rather than a non-repudiation protocol) based on revocable items. Our non-repudiation protocol is designed independently from Vogt's e-payment scheme, though both of them use some mechanisms to "*revoke*" digital items. The Vogt e-payment scheme realizes the revocation of e-cash by allowing the TTP to enforce the bank cancelling of a finished transaction. In contrast, our protocol is special due to the fact that the abort applicant Alice herself (rather than the TTP) issues an abort token so that we can get a 3-move optimistic protocol supporting both transparent TTP and (strong) timeliness. Based the above observations, we claim that our solution is the *first* 3-move fair non-repudiation protocol supporting transparent off-line TTP and (strong) timeliness.

In addition, we remark that our construction does not contradict the theoretical results on optimal efficiency given by Schunter (Chapter 4 of [47]), which state that one needs at least 4 message flows in the exchange protocol to achieve both fairness and timeliness. Because in our protocol, to resolve non-repudiation of receipt the judge should contact with Bob or the TTP to know whether they possess a valid abort token $AT$. However, this situation is not considered in the framework of [47].

**Remark 7**: As mentioned before, dispute resolution policy is an inseparable part for every non-repudiation protocol and certified e-mail scheme, since the judge (and any verifier) settles all possible disputes exactly according to the specified procedures in the dispute resolution policy. Unfortunately, some previous schemes (including [31]) *do not*

provide clear enough details for this procedure as pointed out in [48]. In contrast, those procedures in our protocol are clearly specified.

## 5. Security Discussions

Based on previous descriptions and discussions, we know that in the normal situation, i.e., both involved parties are honest and the communication channel is in order, the sender will receive valid NRR, while the receiver will get the message $M$ and valid NRO, and the TTP is not involved. In other words, our scheme is *complete* and *optimistic* (i.e. supporting off-line TTP). *Timeliness* is also satisfied, since both the sender and the receiver can terminate a protocol instance unilaterally by initiating the abort protocol or recovery protocol, respectively. It is obvious that our protocol is a *generic* scheme, since we do not exploit any specific features of the underlying (secure) signature schemes. In addition, *transparent* TTP is also supported in our protocol because no part of the non-repudiation evidences is generated by the TTP regardless of whether the TTP is asked to take part in the protocol execution or not.

Now, we discuss the most important security requirement for a fair exchange protocol: *fairness*. That is, we need to show that in our scheme, none of the two involved parties can take advantage over the other at the end of each possible protocol execution, even if he or she behaves dishonestly. We classify our discussion into two cases: (1) Alice is honest, but Bob is trying to cheat; and (2) Bob is honest, but Alice is trying to cheat [3].

**Case 1:** *Alice is honest, but Bob is trying to cheat*. Since Alice is honest now, the whole message flow (e1) delivered to Bob is correctly prepared. Therefore, when message flow (e1) is received, Bob will find $EOO$ is indeed Alice's valid signature on message $(f_{EOO}, L, EK)$, where the label $L = H(A, B, T, H(C), HK)$. After that, Bob has to make a choice whether he wants to return his signature $EOR$ to Alice. If Bob does, honest initiator Alice will reveal the correct values of $(K, R)$ as Bob expects. In such situation, Bob gets the valid NRO evidence as well as the message $M$ from Alice, while Alice also obtains the valid NRR evidence from Bob simultaneously. So the protocol execution is fair. If Bob does not send $EOR$ or only sends an incorrect $EOR$ to Alice, he cannot get correct $(K, R)$ from Alice via message flow (e3). However, $EK$ is a ciphertext produced by Alice using the TTP's public encryption algorithm, which is CCA2 secure. So, except Alice, only the TTP can derive $(K, R)$ from the ciphertext $EK$. This means that the TTP is Bob's last resort to get $(K, R)$ by running the recovery protocol. To do this, there are only two strategies.

The first one is to send the TTP all correct items in message flow (r1), including $EK$, $EOO$, $EOR$ etc. In this case, if Alice already aborted the protocol run indexed by the label $L$, Bob can only get a valid abort token $AT$. If this protocol run is not aborted yet, Bob can successfully get the value of $(K, R)$ from the TTP, but Alice will obtain valid $EOR$ too. So, both of those two situations are fair. Another possible strategy is to initiate

[3] Note that there is no need to discuss fairness in the situation where both involved parties are cheaters, because the purpose of all fair exchange protocols is to guarantee that a honest party will not be cheated by other possible dishonest parties. In other words, fairness should be guaranteed just for all honest parties, who properly follow the protocol specifications. Actually, a dishonest party in any fair exchange protocol can always breach fairness for himself or herself trivially by sending his or her digital items directly to the other party without care the other party's items at all.

the recovery protocol by fabricating some new items in message flow (r1). However, to get $(K, R)$ from $EK$ and keep the consistency between $EK$ and $HK$, Bob has to submit original $HK$ and $EK$. The reason is that without the knowledge of secret key $K$, Bob cannot create another valid ciphertext $EK'$ satisfying $EK' = E_T^{R'}(f_K, L', K)$ or something like that, due to the assumption that $E_T$ is a CCA2 secure public key encryption algorithm. Therefore, Bob can only submit the TTP a message flow (r1') with the following format: $A', B', T, HC', HK, EK, EOO_{A'}, EOR_{B'}, Rec_{B'}$, where $A'$ and $B'$ are the identities of some parties colluding with Bob. But this attack is useless again, since the TTP will find the new label $L' = H(A', B', T, HC', HK)$ does not equal to the label $L$ committed in $EK$ (except with negligible probability), since $H(\cdot)$ is assumed to be a collision-resistant one-way hash function. That is, it is computationally *infeasible* to get two different strings such that their hashed values are the same. Therefore, even if Bob asks for the TTP's help in a cheating way, our protocol is still fair for the honest sender Alice.

**Case 2:** *Bob is honest, but Alice is trying to cheat*. The purpose of a dishonest sender Alice is to get a valid NRR evidence from Bob such that Bob cannot access the message $M$ or does not receive the corresponding NRO evidence. In our protocol, Alice may dishonestly execute any or some of the following steps: (e1), (e3), and the abort protocol. In message flow (e1), the identities $(A, B, T)$ and $EOO$ should be valid. Otherwise, according to the specification of our exchange protocol, honest Bob will not respond Alice at all. So, it seems Alice can send out incorrect information (even random strings) for the following items: $C$, $HK$, and $EK$. However, $C$ and $HK$ determine the message $M$ by $M = D_K(C)$ and $H(K) = K$. So, if any of those two items is incorrect, both $EOR$ and $EOO$ cannot be interpreted as valid NRR and NRO evidences. Furthermore, if Alice prepared $EK$ incorrectly, $EOR$ is also useless for anybody (including Alice), since $EOR$ includes both $EK$ and the label $L$, while $L$ is embedded in $EK$ again. At the same time, the symmetric key committed by both $HK$ and $EK$ should be consistent. Otherwise, neither of $EOO$ and $EOR$ is useful. Therefore, we conclude that to get valid $EOR$ from honest Bob, Alice has to correctly prepare message flow (e1).

Another cheating strategy for Alice is to get a valid $EOR$ first and then refuse to reveal $(K, R)$ and/or run the abort protocol. Just refusing to reveal $(K, R)$ does not harm the receiver Bob in essence, since he can get correct $(K, R)$ from the TTP by executing the recovery protocol. On the other hand, if Alice not only refuses to reveal $(K, R)$ but also initiates the abort protocol, Bob will get a valid abort token $AT$ from the TTP though Bob cannot get the values of $(K, R)$ indeed. However, according to the specification of our dispute resolution policy, when repudiation of receipt occurs Bob can provide this valid $AT$ and then invalidate the $EOR$ which is submitted by Alice. Hence, our protocol is fair for the honest receiver Bob too.

Based on the above analysis, we conclude that in the proposed non-repudiation protocol a dishonest party cannot take advantage over the other honest party. In other words, our protocol satisfies the property of *fairness*.

**Remark 8**: In [42], twelve potential attacks are considered against their two certified e-mail schemes. We also checked those attacks in the scenario of our protocol and found that none of them can succeed. However, our protocol is more complicated than theirs since the existence of an abort protocol. So we need to check more possible attacks. The following is an interesting attack from Alice but it cannot succeed again. To prevent the

receiver Bob to get the pair $(K, R)$ the sender Alice may mount the following attack by running the abort protocol earlier before the execution of the real exchange protocol. First, Alice sends the TTP message flow (a1) to abort a protocol instance indexed by a label $L$. Since this protocol instance is never aborted or recovered, the TTP will approve Alice's abort request due to the fact that the TTP does not know this protocol instance does not begin at all. Hence, the sender Alice will get $Con$ while the receiver Bob will get an abort token $AT$. However, Bob may delete the abort token $AT$ together with all information delivered in message flow (a3), since he does not run such a protocol instance with Alice at all. If this is the case, Alice can initialize the exchange protocol sometime later such that Bob cannot successfully apply recovery though Bob does not know this fact. That is, after getting $EOR$ in message flow (e2) Alice maliciously refuses to reveal $(K, R)$. The point is that even in this situation, the receiver Bob can still get the valid abort token $AT$ again from the TTP by launching the recovery protocol, recalling that the TTP stored $AT$ and other relevant information in its database once the abort protocol has been successfully executed. So, our protocol remains fair against this attack.

## 6. Extensions

In this section, we briefly discuss some possible extensions of the proposed non-repudiation protocol. Firstly, we implicitly assume that in our protocol the TTP always stores all the state information in its searching database. Especially, it is necessary to *correctly* record whether a protocol instance indexed by a label $L$ has been aborted or recovered. Otherwise, the TTP may mistakenly confirm Alice's abort request as well as accept Bob's recovery request. In practice, the TTP's storage is limited. To reduce the size of such data in the TTP's searching database, we can introduce a time limit $t$ into our protocol. For example, define $L = H(A, B, T, HC, HK, t)$, where $t$ indicates both the beginning time $t_1$ and the end time $t_2$. That is, both the abort protocol and the recovery protocol cannot be executed before time $t_1$ or after time $t_2$. After $t_2$ expires, the TTP can remove all state information related to $t_2$ from its searching database into a log system or just print them out as archives. Naturally, time limit $t$ should be agreed by both Alice and Bob, and long enough for them (e.g. 24 hours). In this way, the performance of the TTP can be improved if it serves for many users. In a concrete implementation, however, cares should be paid for the possible attacks resulting from the drift of clocks among different parties.

Secondly, another variant could be obtained by deleting the abort protocol but adding time limit $t$ in our scheme. In other words, after Alice delivered message flow (e1) to Bob, she cannot abort this protocol instance at all, but she only needs to wait $EOR$ for a limit time, i.e., until $t_2$. If Alice neither receives a valid $EOR$ from Bob nor from the TTP after time $t_2$, this protocol run is deemed to be cancelled since the TTP will not accept Bob's recovery request any more. This extension has twofold meanings: (1) The whole non-repudiation protocol becomes simpler; and (2) This variant further supports *stateless TTP*, i.e., in theory the TTP has no need to store any state information to maintain fairness. Therefore, the unique task of the TTP is to use its private key to determine the validity of Bob's recovery request and then respond accordingly. Naturally, the recovery protocol and the dispute resolution policy should be modified correspondingly. Moreover, in this variant neither Bob nor the TTP needs to be enquired in the procedure

of non-repudiation of receipt, since the abort token is not used anymore. The disadvantage of this variant is that the sender Alice may need to wait up to a finite time for the termination of a protocol instance. That is, only *weak timeliness* is achieved.

Note that due to the usage of a time limit $t$, the above two variants require reliable channels for all communications from and to the TTP. Let $\Delta$ be the given fixed communication delay for the reliable channels linked to the TTP. To achieve fairness, the following two conditions should be satisfied simultaneously: (a) $t_2 - t_1 > \Delta$ (Otherwise, neither party has enough time to run the abort or recovery protocol.); (b) If necessary, each party should apply the TTP's help before the time $t_2 - \Delta$ since the messages delivered to the TTP can be delayed up to the amount of time $\Delta$ and the TTP does not accept an application in which the corresponding deadline $t_2$ expires.

Thirdly, using the techniques of threshold cryptography (e.g. [49]) the proposed protocol could be extended for the scenarios where the trust on a single TTP needs to be distributed into multiple relatively less trustworthy TPPs, or a non-repudiation evidence is required to be signed only by a given quota of members in a group cooperatively.

Finally, by exploiting the techniques from [32,43,44], it may be possible to extend our protocol as a multi-party non-repudiation scheme where one sender is able to deliver the same message (or different messages) to many recipients *efficiently*. That is, the execution cost of such a multi-party protocol should be (much) less than that of running a classic two-party protocol for $n$ times individually, where $n$ denotes the number of recipients.

**Remark 9**: In a real system, our basic protocol and all the above variants can be implemented and integrated together so that users can choose the proper variant needed. However, since we require that the protocol identifier should be unique, each variant needs to have a different identifier. Naturally, all the corresponding flags should be changed too because those flags contain the protocol identifier, according to the specification given in Section 3. However, those identifiers can share a same segment to recognize that all of those variants originate from the same family.

## 7. Comparison

Table 1 shows the comparison of basic features, security, and efficiency between our new protocol and a number of the state-of-the-art non-repudiation protocols [6,23,31,33, 30,37,40,42,57,58,59,60]. Note that Vogt's protocol [51] is not listed here for comparison, since it is an e-payment scheme rather than a non-repudiation protocol, as we mentioned in Remark 6. In Table 1, those schemes are listed in an order for easy comparison, not according to the chronology. In the category of basic features, we consider five properties: generic scheme or not, transparent TTP or not, off-line or on-line TTP, and whether both EOR and EOO are provided. The result shows that five previous schemes [37,6,31,40,42], as well as our new protocol, support transparent TTP. Among them, only the IS scheme [31] and Micali schemes [40] are generic constructions, while the MK scheme [37], the AN scheme [6] and the NZS schemes [42] are specific schemes, as we mentioned in Section 2.

Here, we only compare two main security requirements: fairness and timeliness. Five previous schemes satisfy timeliness by providing both the abort and recovery protocols, while the ZG schemes [57,58] and Micali's schemes [40] meet only weak timeliness

due to the usage of a deadline. As we discussed above, using deadlines is an interesting method to achieve stateless TTP. However, the schemes in [6,42,31] do not support timeliness at all. Except the three schemes proposed in [30,6,42], fairness in all other schemes is affected by some attacks (in different flavors) identified by Gürgens et al.'s [29,30] and us. Fortunately, it seems that except the FPH protocol [23,41], all those schemes could be repaired by more or less modifications, though the security of revised schemes should be checked carefully again. Due to this reason, we mark those schemes with "Yes*" under the column of "fairness".

In the efficiency evaluation, we compare the costs of both communication and computation in the *normal* case. In other words, the overloads of the abort and recovery protocols are not discussed here, since those events are supposed to occur abnormally and rarely. To complete a successful exchange, only the FPH scheme, the IS protocol, Micali's schemes, and our solution need to transfer 3 message flows between the two involved parties, while this number is 4 or 5 in other schemes. In the exchange protocol of our scheme, Alice and Bob are required to perform the following main computations: (a) Encrypt and verify the encrypted symmetric key $EK$ under the TTP's public key; and (b) Sign and verify two signatures, i.e., $EOO$ and $EOR$. Therefore, our protocol needs 6 asymmetrical cryptographic operations. However, this number representing computational cost varies from 8 to 17 in other schemes. Note that we do not count into the computation cost of symmetric encryption and decryption, i.e., $C = E_K(M)$ and $M = D_K(C)$, since symmetrical operations are much faster than asymmetric operations, especially for common messages (not very long).

Actually, our protocol could become surprisingly efficient if specific algorithms are exploited. For example, if the sender Alice, the receiver Bob and the TTP all use RSA cryptosystems with 1200-bit modulus, and their public keys are some short exponents, e.g., 3, 17 or $625537 = 2^{16} + 1$. In this case, our exchange protocol can be carried out by only 2 modular exponentiations with full exponents, while computational overheads for other 4 modular exponentiations with short exponents can be ignored. Namely, the essential computational costs for Alice and Bob are to sign $EOO$ and $EOR$ respectively, because it is very easy to verify those two signatures as well as generate and verify the encrypted key $EK$. Due to this reason, our protocol may be useable even in mobile or wireless networks, where the available devices usually have limited resources on computation, communication, storage, and power supply.

## 8. Conclusion

In this paper, we first briefly reviewed a number of non-repudiation protocols and certified e-mail schemes. In particular, we identified a potential attack on the schemes in [31,40], where a sender can cheat a receiver by mixing identities of different TTPs when they execute the non-repudiation protocol. Consequently, we concluded that to overcome some limitations and security shortcomings in previous schemes, it is desirable in practice to propose a new *generic* fair non-repudiation protocol supporting *transparent* off-line TTP. To this end, we proposed such a new fair non-repudiation protocol, and pointed out some possible extensions. Compared with the existing solutions, our protocol is not only secure but the *most* efficient. Actually, this new protocol is the *first* three-move fair non-repudiation protocol supporting timeliness and transparent off-line TTP. As the fu-

**Table 1.** Comparison of Basic Features, Security and Efficiency

| | Generic Constr. | Transp. TTP | Off-line TTP | EOR | EOO | Fair-ness | Time-liness | ♯ of Mess. | ♯ of Oper. |
|---|---|---|---|---|---|---|---|---|---|
| ZG [57] | Yes | No | No | Yes | Yes | Yes* | Weak | 5 | 9 |
| ZG [58] | Yes | No | Yes | Yes | Yes | Yes* | Weak | 4 | 8 |
| ZDB [59,60] | Yes | No | Yes | Yes | Yes | Yes* | Yes | 4 | 12 |
| KMZ [33] | Yes | No | Yes | Yes | Yes | Yes* | Yes | 4 | 12 |
| GRV [30] | Yes | No | Yes | Yes | Yes | Yes | Yes | 4 | 10 |
| FPH [23,41] | Yes | No | Yes | Yes | Yes | No | Yes | 3 | 8 |
| MK [37] | No | Yes | Yes | Yes | Yes | Yes* | Yes | 4 | 12 |
| AN [6] | No | Yes | Yes | Yes | No | Yes | No | 4 | 17 |
| NZS [42] | No | Yes | Yes | Yes | Yes | Yes | No | 4 | 14 |
| IS [31] | Yes | Yes | Yes | Yes | Yes | Yes* | No | 3 | 8 |
| Micali [40] | Yes | Yes | Yes | Yes | No | Yes* | Weak | 3 | 8 |
| **Ours** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **3** | **6** |

ture work, we are considering to conduct a formal verification of the proposed protocol using game theory [34] or related tools [38]. At the same time, we also plan to implement a prototype for our non-repudiation protocol, for example, in the scenario of certified e-mail delivery.

## Acknowledgements

The author would like to thank all anonymous referees, on behalf of IWAP 2005 and Journal of Computer Security, for their very helpful and detailed comments on the protocol, as well as Dr. Steve Kremer and Dr. Paul Robinson for their good suggestions to improve this paper.

## Appendix A: Review of OAEP

The Optimal Asymmetric Encryption Padding (OAEP) is first introduced by Bellare and Rogaway in [10], which is used to convert a trapdoor permutation to a public key encryption cryptosystem with semantic security against adaptive chosen-ciphertext attacks, i.e., CCA2 security [20]. Fujisaki et al. [25] formally proved that OAEP is CCA2 secure in the random oracle model under the partial-domain one-wayness of the underlying permutation. Shoup proposed a slightly modified version of OAEP, called OAEP+, which is provably secure under the one-wayness of the permutation (weaker than Fujisaki et al.'s assumption). However, since partial-domain one-wayness of the RSA function is equivalent to the (fulldomain) one-wayness, the security of RSA-OAEP can actually be proven under the one-wayness of the RSA function.

Let $k_0, k_1, k$ and $n$ be proper security parameters, such that $n = k - k_0 - k_1$. In addition, $G$ and $H$ denote two hash functions:

$$G : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{k-k_0} \quad \text{and} \quad H : \{0,1\}^{k-k_0} \longrightarrow \{0,1\}^{k_0}.$$

The following is the description of the OAEP cryptosystem $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ obtained from a permutation $f$, whose inverse is denoted by $g$.

- **Key generation algorithm** $\mathcal{K}(1^k)$: given the security parameter $k$, probabilistically outputs an instance of the function $f$ with its inverse $g$. Then, sets $f$ as the public key $pk$, and $g$ as the private key $sk$.
- **Encryption algorithm** $\mathcal{E}_{pk}(m;r)$: for a given message $m \in \{0,1\}^n$, selects a random number $r \in \{0,1\}^{k_0}$, then computes

$$s = (m||0^{k_1}) \oplus G(r) \quad \text{and} \quad t = r \oplus H(s),$$

  and outputs the ciphertext $c = f(s,t)$.
- **Decryption algorithm** $\mathcal{D}_{sk}(c)$: given ciphertext $c$, first using the private $sk = g$ extracts

$$(s,t) = g(c), \quad r = t \oplus H(s), \quad \text{and} \quad M = s \oplus G(r).$$

  Then, if $[M]_{k_1} \equiv 0^{k_1}$, the algorithm outputs plaintext $[M]^n$; otherwise it returns "reject". Here, $[M]_{k_1}$ denotes the $k_1$ least significant bits of $M$, while $[M]^n$ denotes the $n$ most significant bits of $M$.

According to the above specification, it is obvious that OAEP series cryptosystems [10,50] satisfy the *randomness recoverability*, i.e., the random number $r$ can be recovered in the decryption procedure. Therefore, in our protocol, if Alice refuses to reveal the pair $(K, R)$ to Bob, the TTP can decrypt the encrypted secret key $EK$ and then forward both the secret key $K$ and the random number $R$ to Bob. At the same time, the values of $(K, R)$ allow any third party to verify whether the alleged encrypted key $EK$ correctly corresponds to the message $(f_K, L, K)$ under the TTP's public key by deterministically checking $EK \equiv E_T^R(f_K, L, K)$. That is what we need in our fair non-repudiation protocol.

## References


[1] M. Abadi and R. Needham, Prudent engineering practice for cryptographic protocols, *IEEE Transactions on Software Engineering*, 22(1) (1996), 6-15.

[2] N. Asokan, M. Schunter, and M. Waidner, Optimistic protocols for fair exchange, in: *Proc. of ACM Conference on Computer and Communications Security* (*CCS'97*), pp. 7-17, ACM Press, 1997.

[3] N. Asokan, V. Shoup, and M. Waidner, Optimistic fair exchange of digital signatures, in: *Proc. of Advances in Cryptology - EUROCRYPT'98*, LNCS 1403, pp. 591-606, Springer-Verlag, 1998.

[4] N. Asokan, V. Shoup, and M. Waidner, Optimistic fair exchange of digital signatures, *IEEE Journal on Selected Areas in Communications*, 18(4) (2000), 593-610.

[5] G. Ateniese, Efficient verifiable encryption (and fair exchange) of digital signature, in: *Proc. of ACM Conference on Computer and Communications Security* (*CCS'99*), pp. 138-146, ACM Press, 1999.

[6] G. Ateniese and C. Nita-Rotaru, Stateless-receipient certified e-mail system based on verifiable encryption, in: *Proc. of the Cryptographers's Track at the RSA Conference 2006* (*CT-RSA '02*), LNCS 2271, pp. 182-199, Springer-Verlag, 2002.

[7] F. Bao, R.H. Deng, and W. Mao, Efficient and practical fair exchange protocols with off-line TTP, in: *Proc. of IEEE Symposium on Security and Privacy*, pp. 77-85, IEEE Computer Society, 1998.

[8] F. Bao, G. Wang, J. Zhou, and H. Zhu, Analysis and improvement of Micali's fair contract signing protocol, in: *Proc. of Information Security and Privacy* (*ACISP'04*), LNCS 3108, pp. 176-187, Springer-Verlag, 2004.

[9] F. Bao, Colluding attacks to a payment protocol and two signature exchange schemes, in:*Proc. of Advances in Cryptology - ASIACRYPT '04*, LNCS 3329, pp. 417-429, Springer-Verlag, 2004.

[10] M. Bellare and P. Rogaway, Optimal asymmetric encryption - How to encrypt with RSA, in: *Proc. of Advances in Cryptology - EUROCRYPT '94*, LNCS 950, pp. 92-111, Springer-Verlag, 1994.

[11] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, A fair protocol for signing contracts, *IEEE Transactions on Information Theory*, 36(1) (1990), 40-46.

[12] C. Boyd and E. Foo, Off-line fair payment protocols using convertible signatures, in: *Proc. of Advances in Cryptology - ASIACRYPT '98*, LNCS 1514, pp. 271-285, Springer-Verlag, 1998.

[13] C. Boyd and P. Kearney, Exploring fair exchange protocols using specification animation, in: *Proc. of Information Security* (*ISW'00*), LNCS 1975, pp. 209-223, Springer-Verlag, 2000.

[14] J. Cathalo, B. Libert, J.-J. Quisquater, Cryptanalysis of a verifiably committed signature scheme based on GPS and RSA, in: *Proc. of Information Security Conference* (*ISC'04*), LNCS 3225, pp. 52-60, Springer-Verlag, 2004.

[15] T. Coffey and P. Saidha, Non-repudiation with mandatory proof receipt, *Computer Communication Review*, 26(1) (1996), 6-17.

[16] R. Cramer and V. Shoup, A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack, in: *Proc. of Advances in Cryptology - CRYPTO '98*, LNCS 1462, pp. 13-25, Springer-Verlag, 1998.

[17] I.B. Damgård, Practical and provably secure release of a secret and exchange of signatures, *Journal of Cryptology*, 8(4) (1995), 201-222.

[18] R. Deng, L. Gong, A. Lazar, and W. Wang, Practical protocol for certified electronic mail, *Journal of Network and Systems Management*, 4(3) (1996), 279-297.

[19] Y. Dodis and L. Reyzin, Breaking and repairing optimistic fair exchange from PODC 2003, in: *Proc. of ACM Workshop on Digital Rights Management* (*DRM'03*), pp. 47-54, ACM press, 2003.

[20] D. Dolev, D. Dwork, and N. Naor, Non-meallleable cryptography, *SIAM Journal on Computing*, 30(2) (2000), 391-437.

[21] S. Even and Y. Yacobi, Relations among public key signature schemes, Technical Report 175, Computer Science Dept., Technion, Israel, 1980.

[22] S. Even, O. Goldreich, and A. Lempel, A randomized protocol for signing contracts, *Communications of the ACM*, 28(6) (1985), 637-647.

[23] J. L. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet i Rotger, An efficient protocol for certified electronic mail, in: *Proc. of Information Security Workshop* (*ISW'00* ), LNCS 1975, pp. 237-248, Springer-Verlag, 2000.

[24] A. Fiat and A. Shamir, How to prove yourself: Practical solutions to identification and signature problems, in: *Proc. of Advances in Cryptology - CRYPTO '86*, LNCS 263, pp. 186-194, Springer-Verlag, 1987.

[25] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, RSA-OAEP is secure under the RSA assumption, in: *Proc. of Advances in Cryptology - CRYPTO '01*, LNCS 2139, pp. 260-274, Springer-Verlag, 2001.

[26] J. Garay, M. Jakobsson, and P. MacKenzie, Abuse-free optimistic contract signing, in: *Proc. of Advances in Cryptology - CRYPTO '99*, LNCS 1666, pp. 449-466, Springer-Verlag, 1999.

[27] O. Goldreich, A simple protocol for signing contracts, in: *Proc. of Advances in Cryptology - CRYPTO '83*, pp. 133-136, Plenum Press, 1984.

[28] S. Goldwasser, S. Micali, and R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks, *SIAM Journal of Computing*, 17(2) (1988), 281-308.

[29] S. Gürgens and C. Rudolph, Security analysis of (un-) fair nonrepudiation protocols, in: *Prof.*

*of Formal Aspects of Security* (*FASec'02*), LNCS 2629, pp. 97-114, Springer-Verlag, 2003.

[30] S. Gürgens, C. Rudolph, and H. Vogt, On the security of fair non-repudiation protocols, in: *Proc. of Information Security Conference* (*ISC'03*), LNCS 2851, pp. 193-207, Springer-Verlag, 2003.

[31] K. Imamoto and K. Sakurai, A cerified e-mail system with receiver's selective usage of delivery authortiy, in: *Proc. of Progress in Cryptology - INDOCRYPT '02*, LNCS 2551, pp. 326-338, Springer-Verlag, 2002.

[32] S. Kremer and O. Markowitch, A multi-party non-repudiation protocol, in: *Prof. of Information Security for Global Information Infrastructures* (*IFIP/SEC'00*), pp. 271-280, Kluwer, 2000.

[33] S. Kremer, O. Markowitch, and J. Zhou, An intensive survey of fair non-repudiation protocols, *Computer Communications*, 25(17) (2002), 1606-1621.

[34] S. Kremer and J.-F. Raskin, A game-based verification of non-repudiation and fair exchange protocols, *Journal of Computer Security*, 11(3) (2003), 399-430.

[35] O. Markowitch and Y. Roggeman, Probabilistic non-repudiation without trusted third party, in: *Proc. of 2nd Conference on Security in Communication Networks* (*SCN '99*), Amalfi, Italy, 1999.

[36] O. Markowitch and S. Saeednia, Optimistic fair exchange with transparent signature recovery, in: *Proc. of Financial Cryptography '01*, LNCS 2339, pp. 339-350, Springer-Verlag, 2002.

[37] O. Markowitch and S. Kremer, An optimistic non-repudiation protocol with transparent trusted third party, in: *Proc. of Information Security Conference* (*ISC'01*), LNCS 2200, pp. 363-378, Springer-Verlag, 2001.

[38] C. Meadows, Formal methods for cryptographic protocol analysis: Emerging issues and trends, *IEEE Journal on Selected Areas in Communications*, 21(1) (2003), 44-54.

[39] S. Micali, Simultaneous electronic transactions, US Patent No. 5666420, September 1997.

[40] S. Micali, Simple and fast optimistic protocols for fair electronic exchange, in: *Proc. of 22th Annual ACM Symp. on Principles of Distributed Computing* (*PODC'03*), pp. 12-19, ACM Press, 2003.

[41] J.R.M. Monteiro and R. Dahab, An attack on a protocol for certified delivery, in: *Proc. of Information Security Conference* (*ISC'02* ), LNCS 2433, pp. 428-436, Springer-Verlag, 2002.

[42] A. Nenadić, N. Zhang, and Q. Shi, RSA-based verifiable and recoverable encryption of signatures and its application in certified e-mail delivery, *Journal of Computer Security*, 13(5) (2005), 757-777.

[43] J. A. Onieva, J. Zhou, M. Carbonell, and J. Lopez, A multi-party non-repudiation protocol for exchange of different messages, in: *Proc. of Security and Privacy in the Age of Uncertainty* (*IFIP/SEC '03*), pp. 37-48, Kluwer, 2003.

[44] J.A. Onieva, J. Zhou, and J. Lopez, Enhancing certied email service for timeliness and multicasting, in: *Proc. of 4th International Network Conference* (*INC'04*), pp. 327-336, Plymouth, UK, July 2004.

[45] J. M. Park, E. Chong, H. J. Siegel, and I. Ray, Constructing fair exchange protocols for e-commerce via distributed computation of RSA signatures, in: *Proc. of 22th Annual ACM Symp. on Principles of Distributed Computing* (*PODC'03*), pp. 172-181, ACM Press, 2003.

[46] R.L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, 21(2) (1978), 120-126.

[47] M. Schunter, Optimistic fair exchange, *PhD. Dissertation*, Saarland University, Oct. 2000. Available at *http://www.semper.org/sirene/lit/sirene.lit.html*.

[48] M.-H. Shao, G. Wang, and J. Zhou, Some common attacks against certified email protocols and the countermeasures, *Computer Communications* 29(15) (2006), 2759-2769.

[49] V. Shoup, Practical threshold signatures, in: *Proc. of Advances in Cryptology - EUROCRYPT '00*, LNCS 1807, pp. 207-220, Springer-Verlag, 2000.

[50] V. Shoup, OAEP reconsidered, *Journal of Cryptology*, 15(4) (2002), 223-249.

[51] H. Vogt, Asynchronous optimistic fair exchange based on revocable items, in: *Proc. of Fi-*

*nancial Cryptography 2003*, LNCS 2742, pp. 208-222, Springer-Verlag, 2003.
[52] C.-H. Wang, Untraceable fair network payment protocols with off-line TTP, in: *Proc. of Advances in Cryptology - ASIACRYPT '03*, LNCS 2894, pp. 173-187, Springer-Verlag, 2003.
[53] G. Wang, F. Bao, and J. Zhou, On the security of a certified e-mail scheme, in: *Proc. of Progress in Cryptology - INDOCRYPT '04*, LNCS 3348, pp. 48-60, Springer-Verlag, 2004.
[54] G. Wang, Generic fair non-repudiation protocols with transparent off-line TTP, in: *Proc. of the 4th International Workshop for Applied PKI* (*IWAP'05*), vol. 128 of Frontiers in Artificial Intelligence and Applications, pp. 51-65, IOS Press, 2005.
[55] G. Wang, An abuse-free fair contract signing protocol based on the RSA signature, in: *Proc. of the 14th International World Wide Web Conference* (*WWW'05*), pp. 412-421, ACM Press, 2005.
[56] J. Zhou and D. Gollmann, Certified electronic mail, in: *Proc. of Computer Security - ESORICS'96*, LNCS 1146, pp. 160-171, Springer-Verlag, 1996.
[57] J. Zhou and D. Gollmann, A fair non-repudiation protocol, in: *Proc. of the IEEE Symposium on Security and Privacy*, pp. 55-61, IEEE Computer Society Press, 1996.
[58] J. Zhou and D. Gollmann, An efficient non-repudiation protocol, in: *Proc. of the 10th Computer Security Foundations Workshop* (*CSFW'97*), pp. 126-132, IEEE Computer Society Press, 1997.
[59] J. Zhou, R. Deng, and F. Bao, Evolution of fair non-repudiation with TTP, in: *Proc. of Information Security and Privacy* (*ACISP'99*), LNCS 1587, pp. 258-269, Springer-Verlag, 1999.
[60] J. Zhou, R. Deng, and F. Bao, Some remarks on a fair exchange protocol, in: *Proc. of Public Key Cryptography* (*PKC '00*), LNCS 1751, pp. 46-57, Springer-Verlag, 2000.