Int J Inf Secur (2005) 4: 253–262 / Digital Object Identifier (DOI) 10.1007/s10207-004-0063-7

On the security of fair non-repudiation protocols

Sigrid Gürgens[1], **Carsten Rudolph**[1], **Holger Vogt**[2]

[1] Fraunhofer – Institute for Secure Information Technology SIT, Rheinstrasse 75, 64295 Darmstadt, Germany
e-mail: {guergens,rudolphc}@sit.fraunhofer.de
[2] SRC Security Research & Consulting GmbH, Graurheindorfer Str. 149a, 53117 Bonn, Germany
e-mail: Holger.Vogt@src-gmbh.de

Published online: 3 February 2005 –

Abstract. We analyzed two non-repudiation protocols and found some new attacks on the fairness and termination property of these protocols. Our attacks are enabled by several inherent design weaknesses, which also apply to other non-repudiation protocols. To prevent these attacks, we propose generic countermeasures that considerably strengthen the design and implementation of non-repudiation protocols. The application of these countermeasures is finally shown by our construction of a new fair non-repudiation protocol.

Keywords: Security protocols – Fair non-repudiation – Protocol design – Protocol analysis

1 Introduction

In many applications for electronic business and other binding telecooperations one essential security requirement is that either all parties reach their goals or no party reaches its goal. In other words, the protocols have to provide fairness [2]. Furthermore, undeniability of an exchange of data may be required for commercial transactions. While non-repudiation [23] can be provided by standard cryptographic mechanisms like digital signatures, fairness is more difficult to achieve. A variety of protocols has been proposed in the literature to solve the problem of fair message transfer with non-repudiation. One possible solution comprises protocols based on a trusted third party (TTP) with varying degrees of involvement. In early published protocols, the messages were forwarded by the TTP [10]. A more efficient solution was proposed by Zhou and Gollmann [27]. Here, the TTP acts as a lightweight notary. Instead of passing the complete message through the TTP and thus possibly creating a bottleneck, only a short-term key is forwarded by the TTP and the encrypted message is directly transferred to the recipient. Based on this approach, several protocols and improvements have been proposed. Recent evolution of these protocols has resulted in efficient, *optimistic* [3, 4] versions where the TTP is involved only if something goes wrong and resolve and abort subprotocols are supposed to guarantee that every party can complete the protocol in time without having to wait for actions of the other potentially malicious party [14, 25, 26]. These protocols only require that the communication channels to and from the TTP be *asynchronous*, i.e., that messages be delivered after an arbitrary but finite amount of time. As no messages are lost, this guarantees a response from the TTP in the resolve and abort subprotocols.

Remarkably, many of the proposed protocols do not satisfy all security properties they are supposed to provide. Possible attacks are not based on the underlying ideas but are enabled by several inherent design weaknesses involving details of the realization. This observation has motivated the development of the design principles proposed in this paper.

The remainder of the paper is organized as follows. After defining fair non-repudiation with timeliness we demonstrate attacks on two protocols for efficient fair non-repudiation with timeliness. Some of these attacks are related to previously published ones [9, 13]. Next, the design weaknesses enabling the attacks are discussed, and design principles to avoid these known attacks are proposed. Finally, we introduce a new efficient protocol that has been developed in accordance with our design principles.

2 Fair non-repudiation respecting timeliness

This paper concentrates on message transfer protocols with certain security properties: Party A sends message

M to party B such that A can prove that B has received M (non-repudiation of receipt) and B can prove that A has sent M (non-repudiation of origin). Furthermore, the protocol should not give the originator A an advantage over recipient B, or vice versa (fairness), and both parties should be able to reach, without depending on the other party's actions, a state where they can stop the protocol while preserving fairness.

Nonrepudiation of origin and non-repudiation of receipt require evidence of origin (EOO) and evidence of receipt (EOR). All parties participating in the protocol have to agree that these evidences constitute valid proofs that the particular receive or send event has happened. In case of a dispute an arbitrator or judge has to verify EOO and EOR. Therefore, for every non-repudiation protocol one has to specify what exactly constitutes valid EOO and EOR. This can be done by specifying the verification algorithm the judge has to execute in order to verify the evidence for dispute resolution.

Even in fair non-repudiation protocols there are intermediate states where one party seems to have an advantage, for example, if a TTP has transmitted evidence first to one party and the other party is still waiting for the next step of the TTP. We say a protocol is fair if at the end of the protocol execution no party has an advantage over the other party. This means that if there is an unfair intermediate situation for one party, this party must be able to reach a fair situation without the help of other untrusted parties. For any party P we say a protocol execution has *ended* for P if either P has executed all protocol steps or any remaining protocol step depends on the execution of protocol steps by other untrusted parties. We say a protocol execution is *completed* for P if the protocol execution has ended and P can be sure that no action by any other party enables P to continue with the protocol execution.

In this paper we consider a refined version of the definition of fair non-repudiation with strong fairness and timeliness by Kremer et al. [17], according to which strong fairness is achieved if either A receives EOR and B receives M and EOO or neither party receives any valuable information. Our definition takes into account the worst case in which message M is considered to be valuable information for B. In case knowledge of M is irrelevant, the definition can easily be relaxed. We specify the security goals relative to the role in the protocol. The security goal of the originator of a message has to be satisfied in all scenarios where the originator acts in accordance with the protocol specification while the recipient may act maliciously. In contrast, the security goal of the recipient has to be satisfied in scenarios where the originator can act maliciously.

Definition 1. *A message transfer protocol for originator A and recipient B provides* fair non-repudiation with timeliness *if the following security goals are satisfied for all possible messages M.*

Security goal for A: Fair non-repudiation of receipt *At any possible end of the protocol execution from A's point of view either A owns a valid EOR by B for message M or B has not received M and B has no valid EOO by A for M.*

Security goal for B: Fair non-repudiation of origin *At any possible end of the protocol execution from B's point of view either B has received M and owns a valid EOO by A for M or A has no valid EOR by B for M.*

Security goal for A and B: Timeliness *At any possible state in the protocol execution each party can complete the protocol without any action of other untrusted parties.*

This definition of fair non-repudiation protocols is closely related to certified e-mail protocols [4, 7, 11, 15]. The substantial difference between both kinds of fair exchange protocols is the fact that the evidence of origin for the message is optional in certified e-mail protocols. All other requirements for certified e-mail match those for non-repudiation protocols. Thus, non-repudiation protocols as defined above are a generalization of certified e-mail protocols and can be applied to a certified e-mail scenario in a straightforward manner.

3 Attacks on two non-repudiation protocols

In this section we show different attacks on two non-repudiation protocols that are supposed to provide fairness and respect timeliness. One protocol was proposed by Zhou et al. [25, 26] and has already been analyzed and improved in [9]. The second protocol is a very similar one proposed by Kremer and Markowitch [14, 17]. This protocol has been analyzed in [18]. Several other protocols [15, 16, 20–22] build on this protocol. Both protocols (which we will henceforth call ZDB protocol and KM protocol, respectively) use an offline TTP, i.e., a TTP that is involved only in case of incorrect behavior of a malicious party or of a network error. The basic idea of the main protocol part not involving the TTP stems from [27]. The structure of the two protocols is very similar. However, small details in the protocol design permit several different attacks. Both protocols are designed to provide fair non-repudiation with timeliness as described in Definition 1, and both realizations are based on the same ideas.

3.1 The general idea

- Party A generates a label L the purpose of which is to link all protocol messages.
- In the first step of the main protocol, message M is sent to B encrypted using a symmetric key K computed by A: $C = e_K(M)$. Only after evidence of origin (EOO_C) and receipt for C (EOR_C) are exchanged, A sends K and EOO_K to B. B can then decrypt C, resulting in M, and returns evidence of receipt for K (EOR_K).

- Two subprotocols *abort* and *resolve* involving a trusted third party TTP shall provide fairness and timeliness.
- The *abort* subprotocol can be invoked by A at any stage of the protocol. If no *resolve* has happened earlier, the TTP confirms the *abort* and will answer any future *resolve* request of a principal for this combination of A, B, and label L with the *abort* message.
- Missing evidence of origin or receipt of key K and the missing key itself can be obtained by either party by using the *resolve* subprotocol. As the first message of the main protocol includes K encrypted with the public key of the TTP ($E_{\mathrm{TTP}}(K)$), the TTP can extract this key and therefore produce a signature con_K that serves for B as evidence of origin of K, and for A as evidence of receipt of K. Furthermore, TTP can submit K to B.

3.2 The ZDB protocol

For the description of the ZDB protocol we use the following notation.

- M: message being sent from A to B
- K: symmetric key generated by A
- M, N: concatenation of two messages M and N
- $H(M, K)$: a one-way hash function applied to message M and key K
- $e_K(M)$ and $d_K(M)$: encryption and decryption of message M with key K
- $C = e_K(M)$: committed ciphertext for message M
- $L = H(M, K)$: label to link C and K
- $f_1, f_2, \ldots, f_8$: message flags to indicate the purpose of the respective message
- $E_{\mathrm{TTP}}(K)$ encryption of key K with TTP's public key
- $sig_A(M)$: principal A's digital signature on message M with A's private signature key. Note that the plaintext is not recoverable from the signature, i.e., for signature verification the plaintext needs to be made available
- $\mathrm{EOO}_C = sig_A(f_1, B, L, C)$: evidence of origin of C
- $\mathrm{EOR}_C = sig_B(f_2, A, L, \mathrm{EOO}_C)$: evidence of receipt of C
- $\mathrm{EOO}_K = sig_A(f_3, B, L, K)$: evidence of origin of K
- $\mathrm{EOR}_K = sig_B(f_4, A, L, \mathrm{EOO}_K)$: evidence of receipt of K
- $sub_K = sig_A(f_5, B, L, K, \mathrm{TTP}, \mathrm{EOO}_C)$: evidence of submission of K to the TTP
- $con_K = sig_{\mathrm{TTP}}(f_6, A, B, L, K)$: evidence of confirmation of K by the TTP
- $abort = sig_{\mathrm{TTP}}(f_8, A, B, L)$: evidence of abortion
- $A \to B : M$: Party A sends message M with B being the intended recipient

The protocol consists of one main *exchange* protocol and two subprotocols *abort* and *resolve*. The protocol steps are shown in Table 1.

As already remarked in [12], there is some redundancy in the TTP's response to resolve requests. As EOR_C and f_2 have been sent by U, these values are not required in the TTP's response to U.

Table 1. The ZDB protocol

Exchange subprotocol			
1.	$A \to B$	:	f_1,f_5,B,L,C,TTP,$E_{\mathrm{TTP}}(K)$, EOO_C,sub_K
2. IF B gives up THEN quit ELSE			
	$B \to A$	:	$f_2, A, L, \mathrm{EOR}_C$
3. IF A gives up THEN abort ELSE			
	$A \to B$	:	$f_3, B, L, K, \mathrm{EOO}_K$
4. IF B gives up THEN resolve ELSE			
	$B \to A$	:	$f_4, A, L, \mathrm{EOR}_K$
5. IF A gives up THEN resolve			
Abort sub-protocol which can only be performed by A			
1.	$A \to \mathrm{TTP}$	:	$f_7, B, L, sig_A(f_7, B, L)$
2. IF resolved THEN			
	$\mathrm{TTP} \to A$	:	$f_2, f_6, A, B, L, K, con_K, \mathrm{EOR}_C$
	ELSE		
	$\mathrm{TTP} \to A$	:	$f_8, A, B, L, abort$
Resolve sub-protocol, where initiator U is either A or B			
1.	$U \to \mathrm{TTP}$	:	f_1,f_2,f_5,A,B,L,TTP,$E_{\mathrm{TTP}}(K)$, sub_K,EOO_C,EOR_C
2. IF aborted THEN			
	$\mathrm{TTP} \to U$	:	$f_8, A, B, L, abort$
	ELSE		
	$\mathrm{TTP} \to U$	:	$f_2, f_6, A, B, L, K, con_K, \mathrm{EOR}_C$

3.3 The KM protocol

For the description of the KM protocol we use the same notation as for the ZDB protocol, and the following ones:

- $f_{\mathrm{EOO}_C}, f_{\mathrm{EOR}_C}, f_{sub}, \ldots$: message flags to indicate the purpose of the respective message
- $\mathrm{EOO}_C = sig_A(f_{\mathrm{EOO}_C}, B, \mathrm{TTP}, L, H(C))$: evidence of origin of C
- $\mathrm{EOR}_C = sig_B(f_{\mathrm{EOR}_C}, A, \mathrm{TTP}, L, H(C))$: evidence of receipt of C
- $sub_K = sig_A(f_{sub}, B, L, E_{\mathrm{TTP}}(K))$: evidence of submission of K to the TTP
- $\mathrm{EOO}_K = sig_A(f_{\mathrm{EOO}_K}, B, L, K)$: evidence of origin of K
- $\mathrm{EOR}_K = sig_B(f_{\mathrm{EOR}_K}, A, L, K)$: evidence of receipt of K
- $Rec_X = sig_X(f_{Rec_X}, Y, L)$: recovery request
- $con_K = sig_{\mathrm{TTP}}(f_{con_K}, A, B, L, K)$: evidence of confirmation of K by the TTP
- $abort = sig_A(f_{abort}, B, L)$: abort request
- $con_a = sig_{\mathrm{TTP}}(f_{con_a}, A, B, L)$: evidence of abort confirmation

The protocol also consists of one main *exchange* protocol and two subprotocols *abort* and *resolve*. The steps of these three parts are shown in Table 2.

Table 2. The KM protocol

Exchange subprotocol			
1.	$A \to B$	:	f_{EOO_C},f_{sub},B,TTP,L,C,$E_{\mathrm{TTP}}(K)$, EOO_C, sub_K
2. IF B gives up THEN quit ELSE			
	$B \to A$	:	$f_{\mathrm{EOR}_C}, A, \mathrm{TTP}, L, \mathrm{EOR}_C$
3. IF A gives up THEN abort ELSE			
	$A \to B$	:	$f_{\mathrm{EOO}_K}, B, L, K, \mathrm{EOO}_K$
4. IF B gives up THEN resolve ELSE			
	$B \to A$	:	$f_{\mathrm{EOR}_K}, A, L, \mathrm{EOR}_K$
5. IF A gives up THEN resolve			
Abort subprotocol, which can only be performed by A, but involves B as well			
1.	$A \to \mathrm{TTP}$	:	$f_{abort}, L, B, abort$
2. IF resolved OR aborted THEN stop ELSE			
	$\mathrm{TTP} \to A$	:	$f_{con_a}, A, B, L, con_a$
	$\mathrm{TTP} \to B$	:	$f_{con_a}, A, B, L, con_a$
Resolve subprotocol, where X and Y are either A or B			
1.	$X \to \mathrm{TTP}$	:	f_{Rec_X},f_{sub},f_{EOR_C},f_{EOO_C},Y, L,$H(C)$,$E_{\mathrm{TTP}}(K)$,Rec_X,sub_K, EOR_C,EOO_C
2. IF aborted OR recovered THEN stop ELSE			
	$\mathrm{TTP} \to A$	:	f_{con_K},f_{EOR_C},A,B,L,K, con_K,EOR_C
	$\mathrm{TTP} \to B$	:	$f_{con_K}, A, B, L, K, con_K$

3.4 Attacks on the ZDB protocol and the KM protocol

In this section we explain the differences between the two protocols and possible attacks.

3.4.1 Reuse of $E_{\mathrm{TTP}}(K)$ in a different context

The first message of the main parts of both protocols essentially consists of L, C, the respective EOO_C, $E_{\mathrm{TTP}}(K)$, and a signature sub_K by A that can be used by B in the *resolve* subprotocol. This signature is supposed to confirm to the TTP that A has submitted the key K in the particular context, identified by participating parties and label L. A subtle difference in the content of sub_K enables our attack against the KM protocol, which is not possible on the ZDB protocol. In the KM protocol, sub_K contains only items sent as parts of the first message, in particular it contains $E_{\mathrm{TTP}}(K)$, while in the ZDB protocol the key K itself is signed. Thus in the KM protocol B can reuse $E_{\mathrm{TTP}}(K)$ in a different protocol run with B' and produce a valid sub'_K that consists of $sig_B(f_{sub}, B', L', E_{\mathrm{TTP}}(K))$, where L' is a new random label. By using this sub'_K together with appropriate values EOO'_C and EOR'_C in the *resolve* subprotocol, B gains K and thus learns the message M. A cannot receive any evidence of receipt for this message, as A has only enough information to run the *abort* subprotocol. But as B executes the *resolve* subprotocol under a different label L', the attack always succeeds. Thus, the protocol is unfair for A (assuming that knowledge of M is valuable information for B). This attack is not possible on the ZDB protocol because knowledge of K is necessary to generate a valid sub_K.

3.4.2 Sending wrong sub_K or $E_{\mathrm{TTP}}(K)$

On the other hand, signing the plaintext K has the drawback that B is not able to check the validity of the signature sub_K. This enables a different attack on the ZDB protocol published by Boyd and Kearney [9], which is not possible on the KM protocol. By sending an invalid sub_K, A can prevent the termination of B.

Even the improved version proposed in [9], where sub_K is included in EOO_C, is susceptible to a similar attack where A sends a wrong $E_{\mathrm{TTP}}(K)$. Then the *resolve* subprotocol stops with an error when started by B. This again prevents B from terminating. However, A can construct a resolve request with the correct encryption of the key, which allows A to complete the protocol at any time.

In [24] Zhou proposed a variant that is designed to prevent the original Boyd and Kearney attack, but it fails to do so. Instead, this variant only works against our new attack.

3.4.3 Reuse of labels and keys

Both protocols suffer from a new attack that is analogous to the attack on the Zhou-Gollmann protocol presented in [13]. This leads to unfairness for B. We have found several variants of this attack using the analysis method for cryptographic protocols based on the SH verification tool as described in [13]. The attacks are based on the following facts:

- The label $L = H(M, K)$ that is supposed to uniquely define a protocol run and identify messages belonging to this run cannot be verified by B before the last step. Furthermore, the label cannot be checked by the TTP as well, because the TTP never receives message M. Therefore, A can start a protocol run for some message M' but use the wrong label $L = H(M, K)$, and receive evidence of receipt for K either from B (if B does not check the label at all) or from the TTP in a resolve subprotocol.
- Apart from the label (which is chosen by A and cannot be verified by the TTP), there is no link between the evidence of receipt of K and message M'. Therefore, this evidence serves as evidence of receipt of K for a second protocol run, in which A starts using the same label $H(M, K)$, and this time the correct message M. B's response provides A with EOR_C for M, matching the evidence of receipt for K from the first protocol run.
- We may assume that B will not store evidence from past protocol runs. If the run was completed successfully, B will delete all evidence once the transaction

has been settled in such a way that further disputes cannot occur. On the other hand, B will not keep incomplete evidence for a run that failed because of a wrong label.
- In the second protocol run A can send wrong $E_{\mathrm{TTP}}(K)$ and sub_K to prevent a successful resolve by B.

This attack may also be successful against related protocols proposed in [9, 15, 24, 25]. Variants of this attack may also be applied to optimistic multi-party non-repudiation protocols like [16, 21].
Remark: The attack involves a risk for A. A has to reuse the key K, which is already known to B. Therefore, B might be able to decrypt M after the first message of the second protocol run and A would not get any evidence of receipt. However, assuming that A is the malicious party and B the honest party being attacked, there is no reason to assume that B might try old keys to decrypt a new message. Furthermore, remembering all old protocol runs and performing a check for every new protocol run would be an undesirable overhead for B.

4 Improving the security of fair exchange protocols

To overcome the weaknesses of previous non-repudiation protocols, we propose several improvements for their implementation. As our new design principles are quite generic, we aim at improving every kind of fair exchange protocol.

Our work is motivated by the observation that successful attacks on fair exchange protocols are often enabled by exploiting a combination of several small design weaknesses. Thus, we suggest some improvements that enhance the security of fair exchange protocols step by step. Even if no attacks are known for a protocol with a weak implementation part, we recommend replacing such a part by the strongest available implementation. As soon as new and stronger attacks are developed, such a redundancy of security mechanisms may just turn out to be the surplus of security that is required to defeat such attacks.

Our new design principles deal with different parts of fair exchange protocols like choosing a unique label, creating correctly formed messages, and implementing the TTP. We first motivate why some current implementations suffer from attacks and then propose our design principles that improve the security. Finally, we give some examples showing how our improvements should be implemented in practice. The complete picture of a secure implementation is then demonstrated by an example in Sect. 5.

Some of the design recommendations are related to design principles for cryptographic protocols proposed by Abadi and Needham [1]. Especially the design principles concerning the construction of messages given in Sect. 4.2 coincide with some of the principles about *explicit communication* and *naming* given in [1]. However, before using these principles to security protocols in general, one has to check that they apply for the particular scenario. Naming participants, for example, might be inappropriate in a scenario where anonymity is required.

Another related article on implementing non-repudiation protocols is written by Louridas [19]. This article concentrates on protocols with active TTP and provides rather high-level suggestions for protocol design. In contrast, our design principles aim at the more efficient optimistic protocols and give detailed advice on how to improve the security of these kinds of fair exchange protocols.

Besides the importance of how to construct labels, messages, etc., it should be noted that a necessary prerequisite for any non-repudiation protocol to work is that the parties agree, in a verifiable way, on those message parts that constitute valid evidence of origin and receipt, respectively. This includes the agreement on the particular algorithms and modes to use for encryption and signature generation and verification.

4.1 Unique labels

4.1.1 Motivation

All messages in a fair exchange protocol should contain a label that is assigned to identify the transaction to which such a message belongs. An ad hoc solution would be that one party simply chooses a random number for this label. However, such a label is not related to a certain protocol run so that it can be reused in a different transaction. For this reason, the label is often computed as a hash value of some unique identifying information.

Nonrepudiation protocols often propose a label of the form $L = H(M, K)$, where M is the exchanged message, K is the symmetric encryption key used to encipher this message M, and H is a cryptographic hash function (e.g., the KM protocol and the ZDB protocol use such a label). This construction has the disadvantage that it is impossible to verify the correct computation of this label until a party receives M and K. Thus, a cheating party can still send a random label and the receiving party will not be able to detect this before the exchange is already performed.

A straightforward improvement is to define a label $L = H(H(M), H(K))$ and reveal $H(M)$ and $H(K)$ to any party that wants to check the creation of the label. While this construction allows one to verify the label L, it reveals too much information: If there is only a limited number of possible values for M (e.g., if M is either "yes" or "no"), then the supposed receiver of the message can simply hash all possible values until he finds the one matching $H(M)$. Then he knows the message M and can still abort the exchange.

4.1.2 Design principles

For a label that is supposed to identify a certain exchange transaction we propose the following security properties:

Verifiability: The creation of a label should be verifiable by anybody. In other words, this means that all parties can compute the label on their own and compare this result to the received label.

Uniqueness: It should be ensured that a label uniquely identifies a protocol run and the corresponding messages. Choosing the same label again must always result in an identical exchange. This can be ensured by deriving the label from all the information that uniquely identifies this exchange transaction.

Secrecy: The values that are used to compute the label must not reveal any additional useful information about the exchanged items. This prevents the label from leaking information.

4.1.3 Example

For a non-repudiation protocol the label L can be computed using a cryptographic hash function H like this:

$$L = H(A, B, \mathrm{TTP}, H(C), H(K)) .$$

The values A, B, and TTP are unique identifiers for the participating parties. We assume that naming the TTP fixes all protocol parameters like the used symmetric encryption, signature algorithm, hash function, etc. Then $H(C)$ and $H(K)$ exactly specify the exchanged message. As the ciphertext $C = e_K(M)$ is the encryption of message M with a symmetric key K, the message M is uniquely determined by the decryption $M = d_K(C)$. The label is constructed with the hash values of C and K because a receiving party may not be in possession of C and K. Instead it is sufficient to know $H(C)$ and $H(K)$ to check the correct computation of this label. The hash function must be collision resistant, i.e., it must be hard to find two values X, X' with $X \neq X'$ and $H(X) = H(X')$. It follows that if two labels $L = H(A, B, \mathrm{TTP}, H(C), H(K))$ and $L' = H(A', B', \mathrm{TTP}', H(C'), H(K'))$ are equal, then in practice all components must be equal ($A = A'$, $B = B'$, $\mathrm{TTP} = \mathrm{TTP}'$, $C = C'$, and $K = K'$), which means that the same message is exchanged by the same parties.

The secrecy property is also met for this construction of a label: Revealing C and $H(K)$ does not help an attacker to find M, as we assume the encryption $C = e_K(M)$ to be confidential as long as the key K is unknown. Due to the one-wayness of $H(K)$, it is impossible to find K, as the probability of guessing a correct symmetric key K and then testing it against $H(K)$ is negligible in practice.

4.2 Construction of messages

4.2.1 Motivation

In the ZDB protocol [25, 26] only some parts of the first exchange message are signed by the sender. If the TTP detects an error in the unsigned part of this message, it cannot determine the cheating party and thus fails to resolve such a conflict. This example shows that not only the correct construction of a message must be verifiable, but also the party responsible for an incorrect message should be detectable.

The KM protocol [14, 17] suffers from another problem: The encrypted key can be reused in a different exchange transaction. As the TTP cannot detect this kind of cheating, it can be misused as a decryption oracle, thus revealing the secret key even if it belongs to a different transaction.

4.2.2 Design principles

These observations motivate our design principles for a more secure construction of protocol messages:

Authenticity of messages: All message parts should be included in the respective signature (in plaintext or as hash). This proves its authenticity to every recipient. If a part of a message is not included in the signature, then this part cannot be trusted by the recipient.

Verifiability of messages: Every recipient of a message should be able to verify this message. Such a verification of a message includes syntactic checks (e.g., is the data type of the message correct?) as well as semantic checks (e.g., is the message signed by the right party?).

Context of messages: It should be possible for the recipient of a message to identify the transaction to which its parts belong. Furthermore, it must be possible to determine the intention of such a message without doubt.

4.2.3 Examples

While signing the complete message may be easy to implement, there may be a conflict with the verifiability of this message: If the signed message contains a value that is not known to one of its intended recipients and has to be kept secret, then the hash of this secret value may be signed instead. If revealing this hash value still preserves the confidentiality of the secret value, the recipient should be provided with this hash value, which enables at least the verification of all other parts of the signed message. For example, a signature $sig_A(L, K)$ of a secret key K may be replaced by the signature $sig_A(L, H(K))$. Here the one-way property of the hash function is sufficient to prevent the computation of K from $H(K)$, as the cryptographically secure key K consists of enough random bits to defend against key-guessing attacks [i.e., guessing K' until $H(K')$ equals $H(K)$]. Including the hash value in the signature allows signature verification, which ensures that the label L has been signed by A. But note that both signatures differ in the fact that the first one proves to the verifier that A knows K, while the second one only says that A knows $H(K)$.

The possibility of identifying the intention of a message is important for the security of many protocols: A standard attack is to use a message intended for step k in step k'. This can be prevented (and is prevented by all protocols mentioned above) by including unique flags in the respective signatures. Otherwise, EOO_C of the KM protocol could be taken for EOR_C and vice versa, as the only difference besides the flags is the party's name included in the signature. Furthermore, a uniquely chosen label L shows which exchange transaction this signature belongs to. Thus, this label L should be incorporated in every protocol message.

In contrast to signed messages that are relatively easy to verify, an encrypted message can only be verified at decryption, which demands additional care for such a construction. A good example is an encryption like $E_{\mathrm{TTP}}(f_K, L, K)$ that allows a decrypting TTP to recover not only the key K, but also the flag f_K and the label L. These values f_K and L should then define the context in which the TTP is allowed to reveal the key K. To prevent attacks on such a ciphertext, the used encryption scheme should provide non-malleability, i.e., it should be impossible to modify this ciphertext to construct a different meaningfully related ciphertext.

4.3 TTP actions

4.3.1 Motivation

In the ZDB and the KM protocols the TTP cannot fully check the validity of the evidence that both parties want to exchange. This leads to the problem that the TTP may send some values that do not constitute valid evidence. But as the TTP is trusted by both parties, it should always send correct responses to these parties.

In the KM protocol the TTP sends only once the result of its conflict resolution to both parties. If one of these parties does not store the respective message, it is impossible to receive it from the TTP again. This leaves such a party in a state of uncertainty, which should be avoided not only for fairness reasons but also for fault tolerance reasons.

4.3.2 Design principles

These observations lead to the following requirements for the TTP behavior:

Meaningful TTP decisions: The decisions of the TTP should have a clear semantics. A resolve decision should guarantee that both parties can gain the desired items. An abort decision (which was initiated by a request of an honest party) should imply that none of the parties can obtain useful information about the desired items from the TTP.

Reply to every request: The TTP should send an answer to every request. This reply should inform the requesting party about the TTP's current state. This requirement is also important for fault tolerance reasons, as a lost message can be retrieved from the TTP by resending the previous request.

4.3.3 Examples

The TTP has to check all items of a resolve request before making any decisions. If they are correct and it can generate the desired items for both parties, the exchange should be resolved. If at least one item is not correct, the TTP should either send a failure message or abort this exchange. In case of abort, the TTP should try to determine who is responsible for this cheating attempt and then take the appropriate measures. For example, if one party creates an incorrect message and signs it, this proves its attempt to cheat. Then the TTP might decide to abort the exchange, even if this is unfair for the cheating party. But as fairness only has to be guaranteed to parties that execute the protocol correctly, this behavior against malicious parties is consistent with the fairness definition.

The replies of the TTP should be implemented in such a way that contacting the TTP a second time with the same values results in the same answer as at the first time if the first request has already been successfully answered (the ZDB protocol uses this feature). This can be implemented by simply remembering all previous decisions.

However, in practice the storage of the TTP is limited and the exchanged information cannot be remembered infinitely. To overcome this problem, time limits can be used, for example by adding the time parameters *start* and *duration* to the label. We do not discuss this further in this paper.

5 An improved asynchronous optimistic fair non-repudiation protocol

In accordance with the guidelines proposed in the previous section, we construct a new asynchronous optimistic fair non-repudiation protocol. We first give a short description of the protocol messages and then present our improved protocol.

5.1 Definition of protocol messages

Our fair non-repudiation protocol is designed to exchange a message M and the corresponding evidence of origin generated by party A for evidence of receipt of this message by party B. Party A encrypts the message M using a symmetric cipher with a random key K, resulting in a ciphertext $C = e_K(M)$. It is assumed that the message M contains enough redundancy so that decrypting C with a different key K' will always result in an invalid message M'. This can, for example, be achieved by demanding that M is a text signed by A and that the symmetric cipher is AES in CBC mode. The agreement on the particular algorithms and modes to use for

encryption and signature generation and verification is assumed to be implicitly defined by naming the TTP. In case the parties have to negotiate the encryption algorithms and parameters before the actual protocol starts, protocol messages used for evidence need to be expanded to contain the result of the negotiation in a verifiable way. However, specifying all protocol parameters and possible encryption algorithms is beyond the scope of this paper.

In accordance with our design principles for a unique label, in Sect. 4.1 we defined the label $L = H(A, B, \mathrm{TTP}, H(C), H(K))$. This label is verifiable by every party that knows A, B, TTP, C or $H(C)$, and K or $H(K)$. Furthermore, we have shown in Sect. 4.1 that it ensures secrecy and uniqueness.

Most protocol messages are signatures on a publicly known unique flag, a unique label identifying the current transaction, and the data to be signed. While the signature ensures authenticity of such a message, the label and the flag define the context to which this message belongs.

- f_K, f_{EOO_C}, f_{EOR_C}, f_{EOO}, f_{EOR}, f_{con}, f_{abort}, f_{aborted}: publicly known unique flags that indicate the purpose of a message
- K: symmetric key randomly chosen by A
- $C = e_K(M)$: symmetric encryption of A's message M
- $L = H(A, B, \mathrm{TTP}, H(C), H(K))$: unique label used to link messages of one protocol run
- $EK = E_{\mathrm{TTP}}(f_K, L, K)$: key K encrypted with the public key of the TTP. The asymmetric encryption scheme must be non-malleable to prevent any modifications of the label L. This ensures that this ciphertext cannot be used in a different context
- $\mathrm{EOO}_C = sig_A(f_{\mathrm{EOO}_C}, L, EK)$
- $\mathrm{EOR}_C = sig_B(f_{\mathrm{EOR}_C}, L, EK)$
- $\mathrm{EOO} = sig_A(f_{\mathrm{EOO}}, L, K)$: evidence of origin issued by A
- $\mathrm{EOR} = sig_B(f_{\mathrm{EOR}}, L, K)$: evidence of receipt issued by B
- $con = sig_{\mathrm{TTP}}(f_{con}, L, K)$: TTP's replacement for evidence of origin and receipt, respectively
- $sig_A(f_{\mathrm{abort}}, L)$: A's request to abort an exchange
- $\mathrm{aborted} = sig_{\mathrm{TTP}}(f_{\mathrm{aborted}}, L)$: TTP's decision to abort an exchange

The goal of our protocol is to exchange non-repudiation evidence, which we define as follows:

NRO evidence for message $M = d_K(C)$: A, B, TTP, C, K, and EOO (in case of an exchange without TTP) or *con* (with TTP involvement).

NRR evidence for message $M = d_K(C)$: A, B, TTP, C, K, and EOR (in case of an exchange without TTP) or *con* (with TTP involvement).

These evidences can be checked by any party with the following steps:

1. Compute $M = d_K(C)$ and $L = H(A, B, \mathrm{TTP}, H(C), H(K))$
2. Check signature EOO or *con* (or EOR or *con* in the case of NRR evidence). Then check that the signing party is mentioned at the correct position in L. If any of these checks fails, the evidence for message M is considered invalid.

5.2 The fair non-repudiation protocol

Our protocol consists, like the KM and the ZDB protocols, of a main part and two subprotocols for *abort* and *resolve*. The main part performs the exchange of non-repudiation evidence as described in Table 3.

We consider equivalent a message that is not correct and one that is not received at all, as in both cases the given actions for conflict resolution can be performed.

The purpose of the *abort* and *resolve* subprotocols is to resolve conflicts with the help of the TTP. They are executed mutually exclusively, which is ensured by the TTP by remembering the previous actions for this label L. The *abort* protocol is specified in Table 4.

The validity of an abort request is checked in the following manner:

1. Compute $L = H(A, B, \mathrm{TTP}, H(C), H(K))$.
2. Check that party A (identified by the first value that is hashed for computing L) supplied a valid signature $sig_A(f_{\mathrm{abort}}, L)$. Only after this check succeeds does the TTP execute this abort subprotocol.

The *resolve* protocol for party $P \in \{A, B\}$ is defined in Table 5.

The validity of a resolve request by party P is checked in the following manner:

Table 3. Our fair nonrepudiation protocol

A $\rightarrow$ B	:	A, B, TTP, C, $H(K)$, EK, EOO_C
B	:	Compute L and verify EOO_C (if not OK, quit exchange)
B $\rightarrow$ A	:	EOR_C
A	:	Verify EOR_C (if not OK, start abort subprotocol)
A $\rightarrow$ B	:	K, EOO
B	:	Test whether this constitutes a valid NRO evidence (if not, start resolve subprotocol)
B $\rightarrow$ A	:	EOR
A	:	Test whether this constitutes a valid NRR evidence (if not, start resolve subprotocol)

Table 4. The abort subprotocol for party A

A $\rightarrow$ TTP	:	A, B, $H(C)$, $H(K)$, $sig_A(f_{abort}, L)$
TTP	:	If A has sent a valid abort request:
		If the exchange has been resolved:
		TTP $\rightarrow$ A : *con*
		Else // exchange not resolved
		TTP $\rightarrow$ A : *aborted*
		If A has sent an invalid request:
		TTP $\rightarrow$ A : Error: invalid request

Table 5. The resolve subprotocol can be started by A or B

P $\rightarrow$ TTP	:	$A, B, H(C), H(K), EK, \mathrm{EOO}_C, \mathrm{EOR}_C$
TTP	:	If P has sent a valid request:
		If the exchange has not been aborted:
		If the exchange has been resolved before or decrypting EK succeeds:
		TTP $\rightarrow$ P : K, *con*
		If decrypting EK fails (A cheated and no resolve happened yet):
		TTP $\rightarrow$ P : *aborted*
		Else // exchange aborted
		TTP $\rightarrow$ P : *aborted*
		If P has sent an invalid request:
		TTP $\rightarrow$ P : Error: invalid request

1. Compute $L = H(A, B, \mathrm{TTP}, H(C), H(K))$.
2. Check that EOO_C and EOR_C are valid signatures generated by the parties named in the first and second message items, respectively. Only after these checks succeed does the TTP execute this resolve subprotocol for P.

The success of decrypting EK is checked as follows:

1. The TTP decrypts EK using its private key: $D_{\mathrm{TTP}}(EK) = (f_{\mathrm{decrypt}}, L_{\mathrm{decrypt}}, K_{\mathrm{decrypt}})$
2. Check that $f_{\mathrm{decrypt}} \equiv f_K$, that $L_{\mathrm{decrypt}} \equiv L$, and that K_{decrypt} was used to compute L. If any of these checks fails, the TTP considers the decryption of EK unsuccessful. This means that A must have generated an incorrect EK on purpose (i.e., A must have tried to cheat) since EK is contained in A's signature EOO_C.

After the TTP processes a valid request for conflict resolution, it has to store either (L, state=aborted, *aborted*) or (L, state=resolved, K, *con*) forever. If the TTP detects an invalid request, it just notifies the sender but does not store anything about it. All this considered, the TTP's reactions guarantee a response to every request, and its decision clearly determines whether an exchange is aborted or resolved.

5.3 Protocol extensions

5.3.1 Transparent TTP

Our protocol can be modified to make involvement of the TTP transparent for party B, which means that the NRO evidence always looks the same – regardless of whether or not the TTP was involved. We simply define EK as follows:

$$EK = E_{\mathrm{TTP}}(f_K, L, K, \mathrm{EOO})\,.$$

In the resolve protocol the TTP decrypts $D_{\mathrm{TTP}}(EK) = (f_{\mathrm{decrypt}}, L_{\mathrm{decrypt}}, K_{\mathrm{decrypt}}, \mathrm{EOO}_{\mathrm{decrypt}})$ and additionally checks that this EOO is A's signature on (f_{EOO}, L, K). If all the checks are successful, the TTP will provide EOO instead of *con* in a resolve request by B.

Making the NRR evidence transparent requires a major modification to our protocol. The idea for such a solution is to enable TTP to generate a modified signature EOR. This can be achieved with cryptographic primitives like verifiable encryption (e.g., [5, 6]) or convertible signatures (e.g., [8, 20]).

5.3.2 Message confidentiality

Our protocol can ensure message confidentiality if A sends C to B using an encrypted channel (e.g., SSL/TLS). Then even the TTP cannot learn anything about the exchanged message M.

6 Conclusions

In this paper we have discussed the analysis of two different non-repudiation protocols that are supposed to be fair and provide timeliness but fail to do so. Our analysis has shown that the KM protocol is unfair for the originator A, that the ZDB protocol does not provide timeliness for the recipient B, and that both protocols allow an attack that makes them unfair for B. The attack on the KM protocol that results in unfairness for A is possible because the context of the key K encrypted for the TTP is not determined, which enables B to use this message part in a different protocol run. The ZDB protocol allows A to send some signature not being verifiable by B, thus hindering B from performing a successful *resolve* subprotocol. Finally, the attack possible on both protocols uses the fact that, although all signatures contain the same label, the evidence of origin and receipt of ciphertext C and evidence of origin and receipt of K are in fact not connected, thus allowing the latter to be valid in more than one protocol run.

Motivated by our analysis, we have presented countermeasures to these attacks that serve as general design principles for fair exchange protocols. Finally, we have presented our own protocol that does not allow the attacks possible for the KM and ZDB protocols. Our protocol is very efficient, as the main protocol part only requires one signature per message, and EOO and EOR consist of only one signature each. Furthermore, only the first protocol message is dependent on the length of message M, which ensures that even the exchange of very long messages has just a minimal impact on the efficiency of our protocol.

References

1. Abadi M, Needham R (1996) Prudent engineering practice for cryptographic protocols. IEEE Trans Softw Eng 22(1):6–15
2. Asokan N (1998) Fairness in electronic commerce. PhD thesis, University of Waterloo, Canada
3. Asokan N, Schunter M, Waidner M (1997) Optimistic protocols for fair exchange. In: Matsumoto T (ed) 4th ACM conference on computer and communications security, Zürich, Switzerland, April 1997. ACM Press, New York, pp 6–17

4. Asokan N, Shoup V, Waidner M (1998) Asynchronous protocols for optimistic fair exchange. In: Proceedings of the IEEE symposium on research in security and privacy, Oakland, CA, May 1998. IEEE Press, New York,hack pp 86–99
5. Asokan N, Shoup V, Waidner M (1998) Optimistic fair exchange of digital signatures. In: Nyberg K (ed) Advances in Cryptology – EUROCRYPT '98, Espoo, Finland, June 1998. Lecture notes in computer science, vol 1403. Springer, Berlin Heidelberg New York, pp 591–606
6. Ateniese G (1999) Efficient verifiable encryption (and fair exchange) of digital signatures. In: Proceedings of the 6th ACM conference on computer and communications security (CCS '99), Singapore, November 1999. ACM Press, New York, pp 138–146
7. Ateniese G, Nita-Rotaru C (2002) Stateless-recipient certified e-mail system based on verifiable encryption. In: Topics in Cryptology – CT-RSA, San Jose, CA, 18–22 February 2002. Lecture notes in computer science, vol 2271. Springer, Berlin Heidelberg New York, pp 182–199
8. Boyd C, Foo E (1998) Off-line fair payment protocol using convertible signatures. In: Advances in Cryptology – ASIACRYPT '98, Beijing, China, October 1998. Lecture notes in computer science, vol 1514. Springer, Berlin Heidelberg New York, pp 271–285
9. Boyd C, Kearney P (2000) Exploring fair exchange protocols using specification animation. In: Information Security – ISW 2000, Wollongong, Australia, December 2000. Lecture notes in computer science, vol 1975. Springer, Berlin Heidelberg New York, pp 209–223
10. Coffey T, Saidha P (1996) Non-repudiation with mandatory proof of receipt. ACM SIGCOMM Comput Commun Rev 26(1):6–17
11. Deng RH, Gong L, Lazar AA, Wang W (1996) Practical protocols for certified electronic mail. J Netw Syst Manage 4(3):279–297
12. Ferrer-Gomila JL, Payeras-Capellà M, Huguet i Rotger L (2000) An efficient protocol for certified mail. In: Information Security – ISW 2000, Wollongong, Australia, December 2000. Lecture notes in computer science, vol 1975. Springer, Berlin Heidelberg New York, pp 237–248
13. Gürgens S, Rudolph C (2002) Security analysis of (un-) fair non-repudiation protocols. In: Formal Aspects of Security 2002 – BCS FASec 2002, London, UK, 18–20 December 2002. Lecture notes in computer science, vol 2629. Springer, Berlin Heidelberg New York, pp 97–114
14. Kremer S, Markowitch O (2000) Optimistic non-repudiable information exchange. In: Biemond J (ed) 21st symposium on information theory in the Benelux, Wassenaar, The Netherlands, May 2000, Werkgemeenschap Informatieen Communicatietheorie, Enschede, pp 139–146
15. Kremer S, Markowitch O (2001) Selective receipt in certified e-mail. In: Progress in Cryptology – INDOCRYPT 2001, Chennai, India, 16–20 December 2001. Lecture notes in computer science, vol 2247. Springer, Berlin Heidelberg New York, pp 136–148
16. Kremer S, Markowitch O (2003) Fair multi-party non-repudiation protocols. Int J Inf Secur 1(4):223–235
17. Kremer S, Markowitch O, Zhou J (2002) An intensive survey of fair non-repudiation protocols. Comput Commun 25(17):1606–1621
18. Kremer S, Raskin J-F (2001) A game-based verification of non-repudiation and fair exchange protocols. In: CONCUR 2001 – Concurrency Theory, Aalborg, Denmark, August 2001. Lecture notes in computer science, vol 2154. Springer, Berlin Heidelberg New York, pp 551–565
19. Louridas P (2000) Some guidelines for non-repudiation protocols. Comput Commun Rev 30(5):29–38
20. Markowitch O, Saeednia S (2001) Optimistic fair exchange with transparent signature recovery. In: Financial Cryptography – FC 2001, Grand Cayman, British West Indies, 19–22 February 2001. Lecture notes in computer science, vol 2339. Springer, Berlin Heidelberg New York, pp 339–350
21. Markowitch O, Kremer S (2000) A multi-party optimistic non-repudiation protocol. In: Information Security and Cryptology – ICISC 2000, Seoul, Korea, December 2000. Lecture notes in computer science, vol 2015. Springer, Berlin Heidelberg New York, pp 109–122
22. Markowitch O, Kremer S (2001) An optimistic non-repudiation protocol with transparent trusted third party. In: Information Security – ISC 2001, Malaga, Spain, October 2001. Lecture notes in computer science, vol 2200. Springer, Berlin Heidelberg New York, pp 363–378
23. Zhou J (1996) Non-repudiation. PhD thesis, University of London, December 1996
24. Zhou J (2001) Achieving fair non-repudiation in electronic transactions. J Organiz Comput Electron Commerce 11(4): 253–267
25. Zhou J, Deng R, Bao F (1999) Evolution of fair non-repudiation with TTP. In: Information Security and Privacy – ACISP '99, Wollongong, Australia, 7–9 April 1999. Lecture notes in computer science, vol 1587. Springer, Berlin Heidelberg New York, pp 258–269
26. Zhou J, Deng R, Bao F (2000) Some remarks on a fair exchange protocol. In: Public Key Cryptography – PKC 2000, Melbourne, Australia, January 2000. Lecture notes in computer science, vol 1751. Springer, Berlin Heidelberg New York, pp 46–57
27. Zhou J, Gollmann D (1996) A fair non-repudiation protocol. In: Proceedings of the IEEE symposium on security and privacy, Oakland, CA, May 1996. IEEE Press, New York, pp 55–61