On the Quest for Impartiality: Design and Analysis of a Fair Non-repudiation Protocol

J. Cederquist[1], R. Corin[1], and M. Torabi Dashti[2]

[1] University of Twente
[2] CWI Amsterdam

Abstract. We design and analyze a simple optimistic fair non-repudiation protocol. Our protocol is considerably simpler and more efficient than current proposals, due mainly to the avoidance of using session labels. We model-check both safety and liveness properties. The safety properties are verified using a standard intruder, and the liveness properties using an intruder that respects the resilient communication channels assumption. Finally, to provide further confidence in the protocol, several vulnerabilities on weaker versions of our protocol are exposed.

1 Introduction

During the last decades the use of open networks for exchanging information has undergone an impressive growth. As a consequence, new security issues like *non-repudiation* and *fair exchange* have to be considered. Repudiation is the denial of a previously uttered statement. In the situation where agent A sends a message to agent B, non-repudiation guarantees that A cannot deny having sent the message and that B cannot deny having received it. One of the major difficulties in designing non-repudiation protocols is to achieve fairness, i.e. to avoid that one of the entities gets its evidence without the other one being able to get its evidence as well.

It has been shown that achieving fair exchange is impossible without a trusted third party (TTP) [18]. However, using a TTP in every exchange is inefficient. So, to avoid bottlenecks, Asokan et al. [2] introduced the optimistic approach to fair exchange, where the TTP is used *only* in the case of session recovery or abortion (which are assumed to be infrequent).

In comparison to other security issues like secrecy or authentication, fairness has not been studied intensively. Secrecy and authentication are safety properties for which the Dolev-Yao intruder is the most powerful intruder [7] (under certain assumptions, such as perfect cryptography). However, we also aim at verifying (session) termination, a liveness property that cannot be verified using the standard Dolev-Yao model. Therefore, we use a modified Dolev-Yao intruder that respects the resilient communication channels assumption (saying that messages sent over the network will eventually be delivered) [6].

S. Qing et al. (Eds.): ICICS 2005, LNCS 3783, pp. 27–39, 2005.

In the literature, several fair non-repudiation protocols have been proposed, e.g. [14, 21, 13]. These protocols use *session labels* to identify session runs. A session label typically consist of a hash of all message components. However, using session labels does not only add computational cost, but also it may introduce vulnerabilities, as shown in [13].

In this paper we design an optimistic non-repudiation protocol which avoids using session labels altogether, and use a model checker to verify it. We refer the interested reader to [6] for detailed explanations regarding the adopted analysis technique and its comparison to other analysis approaches in the literature.

Contributions. Our contributions are threefold, as listed below.

- We propose a fair non-repudiation protocol which is simpler than existing ones. Existing fair non-repudiation protocols use labels to identify the session runs. Here we show that these labels can be avoided, allowing for a more efficient protocol. Our TTP distinguishes session runs by recognizing fresh keys, which the TTP receives in abort or resolve requests.
- We check a finite state model of our protocol, under the perfect cryptography assumption [8], using the technique of [6], briefly presented in Section 3. Our verification shows that an honest agent (that follows the protocol) will not be treated in an unfair way, even if the agent communicates with a dishonest agent that does not follow the protocol.
- To further validate the analysis method, we illustrate several vulnerabilities found by the model-checker when different fields are missing from our protocol. This also provides confidence in the full protocol and indicates that the fields are indeed needed.

The rest of the paper is organized as follows. In the next section we describe our fair non-repudiation protocol. The intruder model and the formal analysis are described in Section 3. In Section 4 we present the results of our formal analysis. We conclude with some related work and final remarks in Section 5.

2 A Fair Non-repudiation Protocol

In this section we describe our fair non-repudiation protocol. We first describe the underlying cryptographic assumptions and requirements on the trusted third party (TTP). Then we present our protocol, and finally we describe the evidences each party obtains and the fair exchange properties the protocol satisfies.

Cryptographic assumptions. In our analysis the cryptographic operations are assumed to be "ideal", as in Dolev and Yao [8]: First, we assume to have a secure one way hash function h. Also, we have symmetric encryption of a message M with key K, denoted by $\{M\}_K$. In $\{M\}_K$, M can only be extracted using K. We let $\{K\}_{TTP}$ denote K encrypted asymmetrically using the trusted third party TTP's public key. Finally, $(X)_A$ denotes X signed by A (using A's private key). The signature can be verified using A's public key, and X can be extracted.

TTP assumptions. Our TTP is assumed to have a persistent database of aborted or resolved sessions, containing entries of the form $\langle status : X\ Y\ W\ Z\rangle$. In our protocol, *status* is either *aborted* or *resolved*; X and Y are agent identities, W is a cryptographic key and Z is the hash of a cipher text. A *query* to this database is denoted by a predicate $status(X, Y, W, Z)$, which holds when entry $\langle status : X\ Y\ W\ Z\rangle$ exists in the TTP's database.

2.1 Protocol

The non-repudiation protocol that we present below allows an agent A to send a message M to agent B in a *fair* manner, meaning that A gets *evidence of receipt* (EOR) iff B receives M as well as *evidence of origin* (EOO). The EOR allows A to prove that B did indeed receive M, whilst the EOO allows B to prove that it was A who sent M. The protocol consists of three sub-protocols:

Main protocol. Agent A wants to send M to B, using TTP for session abort or resolution. Initially, A chooses a fresh key K. The main protocol is:

1. $A \rightarrow B : \{M\}_K, EOO_M$ for $EOO_M = (B, TTP, h(\{M\}_K), \{K, A\}_{TTP})_A$
2. $B \rightarrow A : EOR_M$ for $EOR_M = (EOO_M)_B$
3. $A \rightarrow B : K$
4. $B \rightarrow A : EOR_K$ for $EOR_K = (A, h(\{M\}_K), K)_B$

First A sends $\{M\}_K$, along with EOO_M, which consists of B and TTP's identities, a commitment to send M using K in the form of a hash $h(\{M\}_K)$, and K encrypted with the TTP's public key (along with A's identity) in case the session is later resolved. On receipt, B stores $\{M\}_K$, checks the signature of EOO_M to ensure that the message is genuinely coming from A, and extracts the values for performing more tests: Firstly, it checks that the leftmost value of EOO_M is B's identity; Secondly, that TTP is a valid TTP for B, whom B trusts for recovering a session; Thirdly, B checks that the included hash commitment is indeed the hash of $\{M\}_K$. When all this is verified, B signs EOO_M with his private key to obtain EOR_M and sends it to A. When A gets this message, it checks whether the signature is that of B. If this is the case, A sends K to B. Then B sends EOR_K, signing K along with A's identity and $h(\{M\}_K)$. Note that B does not need to check whether the key in message 3 decrypts $\{M\}_K$, since $(A, h(\{M\}_K), K')_B$ would not be a valid evidence of receipt of M for A.

Abort protocol. If A does not receive a valid EOR_M from B, at step 2 in the main protocol, then A can invoke the *abort* protocol, for canceling the exchange:

1. $A \rightarrow TTP : (abort, h(\{M\}_K), B, \{K, A\}_{TTP})_A$
2. $TTP \rightarrow A :$
$$\begin{cases} E_{TTP} \text{ for } E_{TTP} = (A, B, K, h(\{M\}_K))_{TTP}, \\ \qquad \textit{if } \textit{resolved}(A, B, K, h(\{M\}_K)) \\ AB_{TTP} \text{ for } AB_{TTP} = (A, B, h(\{M\}_K), \{K, A\}_{TTP})_{TTP}, \\ \qquad \textit{otherwise} \end{cases}$$

First A sends to TTP an *abort request* message consisting of an *abort* flag, the commitment $h(\{M\}_K)$, B's identity and $\{K, A\}_{TTP}$. On receipt, TTP checks A's signature, and checks that it can decrypt the message $\{K, A\}_{TTP}$. If the decryption succeeds, TTP checks that the included identity A next to the key K matches the signature of the whole abort request message. Next, TTP queries its database with $\mathit{resolved}(A, B, K, h(\{M\}_K))$. If the query holds, it means that this session has been resolved earlier. The answer from TTP to A is then E_{TTP}, including the key K signed by the private key of TTP. In the case that the query fails, TTP declares that the session is aborted and stores the entry $\langle \mathit{aborted} : A\ B\ K\ h(\{M\}_K)\rangle$ in its database. The answer AB_{TTP} signed by the TTP is returned to A, as an acknowledgment of the successful abortion. Note that this message does not include K in the clear.

Resolve protocol. If B does not get K or A does not get EOR_K, then both parties may *resolve* the protocol by consulting TTP:

$$1.\ P \rightarrow TTP : ((B, TTP, h(\{M\}_K), \{K, A\}_{TTP})_A)_B$$

$$2.\ TTP \rightarrow P : \begin{cases} AB_{TTP}, & \mathit{if}\ \ \mathit{aborted}(A, B, K, h(\{M\}_K)) \\ E_{TTP}, & \mathit{otherwise} \end{cases}$$

Here P is the party that is resolving the session (i.e. A or B). First P sends EOR_M, as a *resolve request* message. On receipt, TTP checks the validity of the signatures, and the successful decryption and matching of $\{K, A\}_{TTP}$. Then, TTP queries its database for $\mathit{aborted}(A, B, K, h(\{M\}_K))$ to find out whether the session has been previously aborted. If the session has not been aborted, the resolve request is accepted and TTP stores $\langle \mathit{resolved} : A\ B\ K\ h(\{M\}_K)\rangle$ in its database, and answers with E_{TTP} evidence containing key K, which is signed with TTP's private key. If the session is already aborted, TTP answers with AB_{TTP}, a message representing the session abortion.

2.2 Evidences and Dispute Resolution

In case of a dispute, the parties present evidences to an external judge. In our protocol, the evidence of receipt EOR for A is EOR_M and $\{M\}_K$, plus either EOR_K or E_{TTP}. The evidence of origin EOO for B is EOO_M, $\{M\}_K$ and K.

Dispute resolution. Suppose B claims that it did not receive M from A, when A possesses EOR. Then A presents EOR_M, $\{M\}_K$ and either EOR_K or E_{TTP} to the judge. The messages EOR_M and $\{M\}_K$ provide proof that B committed in the session to receive M, while EOR_K or E_{TTP} represent that either B received K, or he can receive it from TTP, respectively.

Suppose A claims that it did not send M to B, when B possesses EOO. Then B presents EOO_M, $\{M\}_K$ and K to the judge, who can check that A had indeed committed to communicate M to B. Since K was freshly created by A, B could only have received it from A directly or from TTP, who checked that A provided the correct K in EOO_M.

2.3 Fair Exchange Properties

We aim at verifying *effectiveness*, *fairness* and *timeliness* (cf. requirements for fair exchange in [1]). These properties are illustrated in the case where A is the initiator and B the responder:

- Effectiveness says that if A and B behave according to the protocol and A does not abort, then the protocol session will reach a state where B has received the message M and EOO, and A has received EOR, and both A and B *terminate*, i.e. have no further pending operations to perform in that protocol session.
- Fairness expresses that when the protocol session has terminated then B has received M and EOO if and only if A has received EOR.
- Timeliness means that protocol sessions terminate for all honest parties. In other words, after an honest agent X has initiated a protocol session with some Y, then X will reach its termination[1]. Moreover, timeliness also specifies that *after* this termination the *degree of fairness* does not decrease for X: if X did not get his evidence before termination then it cannot be that Y gets her evidence without X also getting his.

Effectiveness is a functional sanity check, and may thus be verified in a system without intruder. For the other two properties, we can first verify termination and then check fairness and timeliness assuming that the protocol sessions terminate. This has the benefit of reducing the two properties to safety properties. Thus, termination is the only *liveness* property that needs to be checked.

3 Formal Analysis

We now implement the necessary machinery to formally analyze whether the protocol proposed in Section 2.1 meets the properties described in Section 2.3.

3.1 Communication Model

We consider two different communication models. The first model is used for verifying effectiveness. In this model there is no intruder (all agents are honest): A set of agents communicate over a network, performing send and receive actions. These actions are synchronized, meaning that an agent A can only send a message m to B (denoted by $send(A, m, B)$), if B at the same time receives it from A (denoted by $recv(A, m, B)$). The synchronization between $send(A, m, B)$ and $recv(A, m, B)$ actions is denoted by $com(A, m, B)$.

We use a second model to verify all the remaining properties. In this model there is an intruder I with complete control over the network. When an agent A sends a message m with the intention that it should be received by B, it is in fact the intruder that receives it, and it is also only from the intruder that B may receive m. Also in this model send and receive actions are synchronized.

[1] Here termination refers to that particular agents' session. An agent X may continue executing subsequent sessions after one session is finished.

3.2 The μCRL Specification Language and Toolset

We briefly describe the symbols used in the μCRL code of the intruders below. For a complete description of the syntax and semantics of μCRL we refer to [12]. The symbols '.' and '+' are used for the sequential and alternative composition ("choice") operator, respectively. The operator $\sum_{d \in D} P(d)$ behaves like $P(d_1) + P(d_2) + \cdots$. The process expression $p \lhd b \rhd q$, where b is a Boolean term and p and q are processes, behaves like p if b is true, and like q if b is false. Finally, τ represents an internal action, and the constant δ expresses that, from then on, no action can be performed.

The formalization of the protocol described in Section 2 is carried out in μCRL [12]. The μCRL toolset includes an automatic state space generator and symbolic state space reduction tools. The fair exchange properties are expressed in the regular alternation-free μ-calculus [16]. The model checker EVALUATOR 3.0 [16] from the CADP tool set [9] is then used to verify these properties.

3.3 Intruder Models

We use two different intruder models. For safety properties the normal Dolev-Yao intruder [8] is used. As mentioned earlier, this intruder is not suitable for verification of liveness properties [17], so to verify termination we use the intruder suggested in [6]. This intruder is shown to be equivalent, w.r.t. termination, to the Dolev-Yao intruder that respects the resilient communication channels assumption (RCC, messages sent over the network will eventually be delivered) [6], which is enough for our purposes. The Dolev-Yao intruder stores all received messages in a set X, representing its knowledge. The intruder uses X for synthesizing new messages (*synth* in the code below), using the usual rules of message (de)composition (in particular, the intruder can decrypt and sign messages only if it knows the corresponding key). The intruder can also block communications. Below we illustrate a specification of an intruder DY_B, in this case played by dishonest agent B. The intruder DY_B can perform a special evidence action $evidence_B(k, m)$. This action is parameterized by a key k and a message m, meaning that the gathered evidence regards message m and was provided in the session using key k. We allow DY_B to perform the action $evidence_B(k, m)$ only when it can synthesize $EOO(k, m)$. In general, the particular data that constitutes an evidence is protocol specific, denoted below by $EOO(k, m)$.

$$
\begin{aligned}
DY_B(X) = & \sum_{\substack{p \in Agent \\ m \in Message}} recv(p, m, B).DY_B(X \cup \{m\}) + \\
& \sum_{\substack{p \in Agent \\ synth(m,X)}} send(B, m, p).DY_B(X) + \\
& \sum_{\substack{k \in Key \\ m \in msg}} evidence_B(k, m).DY_B(X) \lhd synth(EOO(k, m)) \rhd \delta + \\
& \tau.\delta
\end{aligned}
$$

According to the operational semantics that underlies μCRL, a process $p + \delta$ behaves like p. So to express that the intruder shall be able to stop all communications at its own will, we let it perform an internal action τ before deadlock δ.

The intruder I_B for verifying termination maintains, besides X, a set Y for messages that have been received but not yet sent (cf. RCC). To distinguish the send actions that the intruder eventually has to perform (according to RCC) from the ones that it can perform (but does not have to), the send actions are tagged with X and Y, respectively. The synchronizations between send and receive actions are denoted com, com_X and com_Y referring to the synchronizations between $send$ and $recv$, $send_X$ and $recv$, and $send_Y$ and $recv$, respectively.

$$I_B(X,Y) = \sum_{\substack{p \in Agent \\ m \in Message}} recv(p,m,B).I_B(X \cup \{m\}, Y \cup \{m\}) + \sum_{\substack{p \in Agent \\ m \notin Y \\ synth(m,X)}} send_X(B,m,p).I_B(X,Y) + \sum_{\substack{p \in Agent \\ m \in Y}} send_Y(B,m,p).I_B(X, Y \setminus \{m\})$$

Note that when we split the fairness and timeliness properties into termination and two safety properties, as described in Section 2.3, we also verify these properties using respectively the two intruders above. This can be done since the intruder I_B is equivalent to the Dolev-Yao intruder that respects the communication channels assumption [6].

3.4 Regular Alternation-Free μ-Calculus

The regular alternation-free μ-calculus is used here to formulate properties of (states in) labeled transition systems. It is a fragment of μ-calculus that can be efficiently checked. Here we briefly describe what is needed for expressing the fair exchange properties of the protocol we investigate. For a complete description of the syntax and semantics we refer to [16]. The regular alternation-free μ-calculus is built up from three types of formulas: *action formulas*, *regular formulas* and *state formulas*. We use '.', '$\vee$', '$\neg$' and '*' for concatenation, choice, complement and transitive-reflexive closure, respectively, of regular formulas. The symbols F and T are used in both action formulas and state formulas. In action formulas they represent *no action* and *any action*, respectively. The meaning of F and T in state formulas are the empty set and the entire state space, respectively. The operators $\langle \cdots \rangle$ and $[\cdots]$ have their usual meaning ($\Diamond$ and $\Box$ in modal logics). The CADP toolset also allows wildcards '$*$' in action parameters.

3.5 The Fair Exchange Properties

Here we formalize the fair exchange properties that we verify. To enhance readability, the protocol implementations are extended with certain *abstract* actions

that do not affect the behavior of the agents. An agent P performs the actions $init_P(k, m)$ when it engages in a protocol session, $terminate_P(k, m)$ when the session is over from P's point of view, and $evidence_P(k, m)$ when it receives a valid evidence, for the key k and item m. The TTP performs $abort(k, m)$ when a session is successfully aborted, for the key k and item m.

First, we check that the protocol is effective. Note that this is verified in the model without intruder. Whenever agent P has started execution, then P's termination is inevitable:

$$[T^*.init_P(k, m)]\ \mu Z.(\langle T\rangle T \wedge [\neg terminate_P(k, m)]Z) \tag{1}$$

Also, if there is no abort, P receives its evidence before termination:

$$[(\neg(abort(k, m) \vee evidence_P(k, m))^*.terminate_P(k, m)]F \tag{2}$$

Now we turn to the fairness and timeliness properties, to be verified in the model with intruder. We assume that the intruder plays the role of Q. The properties below are thus defined to describe "fairness for P". The corresponding properties for Q are defined in a similar way. The properties fairness and timeliness are verified, as described above, by verifying termination separately, using the intruder described in Section 3.3

$$\begin{aligned}[T^*.init_P(k, m).(\neg terminate_P(k, m))^*] \\ \langle(\neg com_X(*, *, *))^*.terminate_P(k, m)\rangle T,\end{aligned} \tag{3}$$

i.e. whenever $init_P(k, m)$ has happened, but not yet $terminate_P(k, m)$, there is a path to $terminate_P(k, m)$ that does not contain com_X actions. This means that, whenever $init_P(k, m)$ has happened, but not yet $terminate_P(k, m)$, and assuming RCC, $terminate_P(k, m)$ will happen.

The remaining properties concern safety so we use the normal Dolev-Yao intruder. Fairness (for P) means that if Q gets its evidence, then so shall P:

$$\begin{aligned}[(\neg evidence_Q(k, m))^*.evidence_P(k, m).(\neg evidence_Q(k, m))^*. \\ terminate_P(k, m)]F\end{aligned} \tag{4}$$

This property says that P does not terminate in an unfair state for P. But since P will eventually terminate (property 3), P will indeed terminate in a fair state.

Finally, timeliness for P says that after P's termination, if P has not got his evidence, Q cannot get her evidence unless P initiates a new session with same key and item:

$$\begin{aligned}[(\neg evidence_P(k, m))^*.terminate_P(k, m). \\ (\neg init_P(k, m))^*.evidence_Q(k, m)]F\end{aligned} \tag{5}$$

In the case when P does initiate a new session with same key and item, P will get his evidence if Q gets hers (according to the properties 3 and 4).

4 Results

In this section we describe the results obtained from the formal analysis described in Section 3 performed on our protocol proposed in Section 2.

Honest scenario S_0: A and B are honest. We first encode a scenario in which both A and B are honest, along with the TTP. A exchanges items with B using fresh keys. To model timeouts, we use nondeterministic choices between communication actions. For instance, either A receives an answer timely from B in Message 2, in the main protocol, or it initiates the abort sub-protocol. Correspondingly, B has a choice between a send action and a τ action, which models an asynchronous communication in which the message is ignored by A. This scenario was model-checked and showed to be deadlock-free and effective.

Result 1. *The protocol in Section 2.1 is effective for scenario S_0, satisfying the properties (1) and (2) in Section 3.5.*

Dishonest scenario S_1: A dishonest and B honest. When A is dishonest and B is honest, we execute B along with the intruder, who takes the identity of A. We first verify the safety properties (4) and (5) using the standard intruder (the first intruder in Section 3.3). Then we model check whether A can unfairly generate B's evidence, and verify that this is impossible, thus rendering the protocol secure.

Result 2. *The protocol in Section 2.1 respects fair exchange and timeliness (properties (4) and (5) in Section 3.5, with respect to A) for scenario S_1.*

Second, we force the intruder to respect RCC (by using the second intruder in Section 3.3). This scenario, called S_1', is used to verify termination:

Result 3. *The protocol in Section 2.1 respects termination (property (3) in Section 3.5, with respect to A) for scenario S_1'.*

Dishonest scenario S_2: A honest and B dishonest. In the opposite case, in which A is honest and B is dishonest, we obtain similar results to the above statements.

4.1 Further Experiments

We now illustrate vulnerabilities found by analyzing *modified versions* of the protocol presented in Section 2. The protocol is modified in such a way that certain assumptions are removed or different message components are excluded. The encountered vulnerabilities expose the need for the particular assumptions or excluded message components.

Reuse of keys. Suppose that A reuses a key K in a subsequent session. Then our analysis reports that for dishonest scenario S_2, A may be attacked by B. The attack is reproduced in standard notation below:

$$
\begin{array}{ll}
a1.\ A \to B : \{M\}_K, EOO_M & \text{for } EOO_M = (B, TTP, h(\{M\}_K), \{K, A\}_{TTP})_A \\
\vdots & \\
b1.\ A \to B : \{M'\}_K, EOO_{M'} & \text{for } EOO_{M'} = (B, TTP, h(\{M'\}_K), \{K, A\}_{TTP})_A
\end{array}
$$

First A sends the message $a1$, initiating a session. Then the session runs normally. When A later starts another session by sending message $b1$, B can immediately obtain M' and thus obtain the evidence EOO, before A can obtain its corresponding evidence. The vulnerability above was found in a scenario where A is honest, and uses two items and one key, and B is dishonest. This violation of property (4) shows that A needs to use fresh keys for each new session.

Missing hash in EOO_M. Consider EOO_M, the second component of the main protocol in Section 2.1. If we exclude the hash $h(\{M\}_K)$, obtaining a new $EOO'_M = (B, TTP, \{K, A\}_{TTP})_A$, the following vulnerability is found:

$$
\begin{array}{ll}
1.\ A \to B : \{M\}_{K'}, EOO'_M & \text{for } EOO'_M = (B, TTP, \{K, A\}_{TTP})_A \\
2.\ B \to A : EOR'_M & \text{for } EOR'_M = (EOO'_M)_B
\end{array}
$$

Agent A starts a session with B, but uses a key K' to encrypt message M and embeds a *different* key K in EOO'_M. When B replies A can run the resolve protocol and obtain evidence EOR. However, when B wants to recover, TTP returns K which is not useful to decrypt $\{M\}_{K'}$, hence the evidences of A and B do not match. This vulnerability was found in a scenario where A is dishonest, and uses two keys, and B is honest. Again property (4) is violated which shows that including the hash in EOO_M is necessary for security of the protocol in Section 2.1.

Missing A's identity in EOO_M. We now consider the case in which A's identity is excluded from the component $\{K, A\}_{TTP}$ in EOO_M. Suppose then that $EOO'_M = (B, TTP, h(\{M\}_K), \{K\}_{TTP})_A$. The following attack is found:

$$
\begin{array}{ll}
a1.\ A \to B : \{M\}_K, EOO_M & \text{for } EOO'_M = (B, TTP, h(\{M\}_K), \{K\}_{TTP})_A \\
b1.\ B \to C : \{M\}_K, EOO_M & \text{for } EOO'_M = (C, TTP, h(\{M\}_K), \{K\}_{TTP})_B \\
2.\ C \to B : EOR'_M & \text{for } EOR'_M = (EOO'_M)_C
\end{array}
$$

When A starts a session with B, B immediately starts another session with another agent C, reusing the information that A used. When C answers, B resolves and obtains K and hence the evidence. However A cannot obtain EOO_M since B never answers to A's first message. When A is honest and B is dishonest, B can simply reuse its own identity and resolve to "itself" (we disallow the TTP to check this). Property (4) is thus violated, indicating that the identity of A is needed in $\{K, A\}_{TTP}$.

Missing A's identity in EOR_K. If A's identity is missing in EOR_K (so that $EOR_K = (h(\{M\}_K), K)_B$), the following vulnerability is found:

$a1.\ A \to B : \{M\}_K, EOO_M$ for $EOO_M = (B, TTP, h(\{M\}_K), \{K, A\}_{TTP})_A$
$a2.\ B \to A : EOR_M$ for $EOR_M = (EOO_M)_B$
$a3.\ A \to B : K$
$a4.\ B \to A : EOR_K$ for $EOR_K = (h(\{M\}_K), K)_B$
$b1.\ C \to B : \{M\}_K, EOO_M$ for $EOO_M = (B, TTP, h(\{M\}_K), \{K, C\}_{TTP})_C$
$b2.\ B \to C : EOR_M$ for $EOR_M = (EOO_M)_B$

Here, A runs a normal session a with B which terminates. A is allied to another user C, who starts a replay of the session by A: we assume A hands over M and K to C. Now, B replies with EOR_M, at which point C aborts the session with B. Then B is unable to obtain evidence, but C has evidence since EOR_K does not mention A nor C, and thus it constitutes valid evidence EOR for C as well. This vulnerability appears in our analysis when we hand out information to a dishonest A about previous sessions giving some EOR_K to A (which may be from an old session of B with some other agent X which we assume is allied to A). In such a scenario, property (4) is violated immediately when A runs the abort protocol after B answers its second message. Thus, we conclude that EOR_K needs to include A's identity.

Missing hash in EOR_K. Finally we consider the case in which $h(\{M\}_K)$ is missing in EOR_K, so $EOR_K = (A, K)_B$. The following attack is then possible:

$a1.\ A \to B : \{M\}_K, EOO_M$ for $EOO_M = (B, TTP, h(\{M\}_K), \{K, A\}_{TTP})_A$
$a2.\ B \to A : EOR_M$ for $EOR_M = (EOO_M)_B$
$a3.\ A \to B : K$
$a4.\ B \to A : EOR_K$ for $EOR_K = (A, K)_B$
$b1.\ A \to B : \{M'\}_K, EOO_M$ for $EOO_M = (B, TTP, h(\{M'\}_K), \{K, A\}_{TTP})_A$
$b2.\ B \to A : EOR_M$ for $EOR_M = (EOO_M)_B$

Similar to the previous case, A runs a normal session a with B. Then A starts another session, but now using a different message M', reusing the same key K. After obtaining an answer from B, A aborts the session. In this state A has valid evidence since the previous EOR_K is not bound to M, and thus it is valid also for an exchange between A and B with K. One could argue that B could also remember K and obtain M'. But B is not supposed to be stateful and save old keys, B just follows the protocol as is specified. In a scenario with A dishonest and B honest, property (4) is violated, exposing the mentioned vulnerability. This shows that EOR_K has to contain $h(\{M\}_K)$.

5 Conclusion and Related Work

We present a novel optimistic non-repudiation protocol, simpler than previous proposals. The simplicity is due to avoiding the usage of labels to identify sessions and assuming the usage of fresh keys per-session. We model-check the proposed protocol and verify the fair exchange properties using the technique in [6]. A full formalization can be found in an extended version of this document [5]. To

provide further confidence in our proposal we illustrate vulnerabilities when different fields are missing in the protocol.

Related Work. Several non-repudiation and fair exchange protocols have been previously proposed. Early tries on optimized exchange protocols [11, 10], i.e. those with only three message exchanges in honest protocol runs, have been found flawed [4, 20]. A recent optimized protocol suggested by Zhou [20], developed on previous ones, remarkably does not suffer from previously reported problems. But it has an elaborate dispute resolution phase requiring both participants to attend the court. We believe an evidence of receipt or origin must be self sufficient to settle a dispute, which requires the addition of a fourth message in our protocol.

More recently, Kremer and Markowitch [14] proposed a four-message protocol (KM) to achieve non-repudiation. Their protocol is analyzed by Kremer and Raskin [15] using a game-based technique. Quite similar to the KM protocol is Zhou-Gollman's protocol (ZG) [21]. Gürgens et al. [13] present several potential unfair situations that may happen in both the KM[2] and ZG protocols. These unfair situations arise from confusion in the *labels* used to identify the session runs. By carefully setting (complex) labels, Gürgens et al. propose a protocol for achieving fair exchange. The ZG protocol was also analyzed by Bella and Paulson [3] who used the theorem prover Isabelle to model the protocol by an inductive definition and to prove some desired properties. Another interesting approach to formal verification of fair exchange is the work by Shmatikov and Mitchell [19] who used the model checker Murφ to analyze fair exchange and contract signing protocols, using a Dolev-Yao intruder.

Acknowledgments. We thank Ana Almeida Matos, Sandro Etalle, Wan Fokkink, Pieter Hartel, Steve Kremer and the anonymous reviewers for helpful comments.

References

1. N. Asokan. *Fairness in electronic commerce.* PhD thesis, University of Waterloo, 1998.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *4th ACM Conference on Computer and Communications Security*, pages 7–17. ACM Press, 1997.
3. G. Bella and L. C. Paulson. Mechanical proofs about a non-repudiation protocol. In R. J. Boulton and P. B. Jackson, editors, *TPHOLs 2001*, volume 2152 of *LNCS*, pages 91–104. Springer-Verlag, September 2001.
4. C. Boyd and P. Kearney. Exploring fair exchange protocols using specification animation. In *Information Security Workshop (ISW)*, volume 1975 of *LNCS*, pages 209–223. Springer-Verlag, 2000.
5. J. Cederquist, R. Corin, and M. Torabi Dashti. On the quest for impartiality: Design and analysis of a fair non-repudiation protocol (extended version). Technical Report TR-CTIT-05-32, University of Twente, The Netherlands, 2005.

[2] The KM protocol was not originally designed to provide fair exchange but simply non-repudiation (private communication, 2004).

6. J. Cederquist and M. Torabi Dashti. An intruder model for verifying termination in security protocols. Technical Report TR-CTIT-05-29, University of Twente, Enschede, The Netherlands, 2005.
7. I. Cervesato. The Dolev-Yao Intruder is the Most Powerful Attacker. In J. Halpern, editor, *LICS'01*, Boston, MA, 16–19 June 2001. IEEE Computer Society Press.
8. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.
9. J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A protocol validation and verification toolbox. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *LNCS*, pages 437–440. Springer-Verlag, 1996.
10. J. Ferrer-Gomila, M. Payeras-Capella;, and L. Huguet i Rotger. A realistic protocol for multi-party certified electronic mail. In *Proceedings of the 5th International Conference on Information Security*, pages 210–219, UK, 2002. Springer-Verlag.
11. J. L. Ferrer-Gomila and L. H. Rotger. An efficient asynchronous protocol for optimistic certified mail. In *International Workshop on Cryptographic Techniques and E-Commerce (Cryptec)*, 1999.
12. J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. Springer-Verlag, 1995.
13. S. Gürgens, C. Rudolph, and H. Vogt. On the security of fair non-repudiation protocols. In *Information Security Conference (ISC)*, volume 2851 of *LNCS*, pages 193–207. Springer, 2003.
14. S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, November 2002.
15. S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In K. Larsen and M. Nielsen, editors, *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 551–565. Springer-Verlag, 2001.
16. R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Sci. Comput. Program.*, 46(3):255–281, 2003.
17. C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(2):44–54, 2003.
18. H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trused third party. Technical Report TUD-BS-1999-02, Darmstadt University, 1999.
19. V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.
20. J. Zhou. On the security of a multi-party certified email protocol. In *Proc. ICICS'04*, volume 3269 of *Lecture Notes in Computer Science*, pages 40–52. Springer, 2004.
21. J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 55–61, Oakland, CA, 1996. IEEE Computer Society Press.