Security Analysis of (Un-) Fair Non-repudiation Protocols *

Sigrid Gürgens and Carsten Rudolph
Fraunhofer – Institute for Secure Telecooperation SIT
Rheinstrasse 75, D-64295 Darmstadt, Germany
{guergens,rudolphc}@sit.fraunhofer.de

September 9, 2004

Abstract

An approach to protocol analysis using asynchronous product automata (APA) and the simple homomorphism verification tool (SHVT) is demonstrated on several variants of the well known Zhou-Gollmann fair non-repudiation protocol and on two more recent optimistic fair non-repudiation protocols. Attacks on all these protocols are presented and an improved version of the Zhou-Gollmann protocol is proposed.

1 Introduction

Non-repudiation is an essential security requirement for many protocols in electronic business and other binding tele-cooperations where disputes of transactions can occur. Especially the undeniable transfer of data can be crucial for commercial transactions. While non-repudiation 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 degree of involvement. In protocols published at first, the messages are forwarded by the TTP. A more efficient solution was proposed by Zhou and Gollmann [28]. Here, the TTP acts as a *light-weighted notary.* Instead of passing the complete message through the TTP and thus creating a possible 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 ([27, 10]). More recent protocols achieve further optimisation by reducing TTP involvement to cases with uncorrect or errorneous behaviour. Such *otimistic* fair non-repudiation protocols have been proposed by Asokan, et al. [1], Zhou, et al. [27],and Markowitch and Kremer [12].

Cryptographic protocols are error prone and the need for formal verification of cryptographic protocols is widely accepted. However, non-repudiation protocols are being developed only for a comparatively short time period, thus only very few of these protocols have been subject to a formal security analysis. Only a few existing protocol analysis methods have been extended and applied to non-repudiation protocols [2, 24, 30]. Recently, a new approach modelling non-repudiation protocols as games has been proposed by Kremer and Raskin [11]. Nevertheless, a variety of attacks and weaknesses have been found on fair non-repudiation protocols.

The development of formal methods for protocol analysis has mainly concentrated on authentication and key-establishment protocols. These methods cannot be directly applied to the security analysis of fair non-repudiation protocols. Obviously, formalisations for non-repudiation and fairness are required. Furthermore, the attack scenario for the analysis of fair non-repudiation protocols is different. Many models for protocol analysis consider an attacker to have control over the network, while protocol participants trust each other. In the case of the establishment of a new session key, the requirement of mutual trust results from the

*This paper describes results of work within the project CASENET partly being funded by the European Commission under IST-2001-32446.

security requirement of confidentiality for the new session key. A malicious protocol agent could simply publish a newly established key and therefore no protocol would achieve secure key-establishment in this scenario. In contrast, fair non-repudiation is explicitly designed for scenarios where protocol participants may act maliciously, since no participant (except a trusted third party) is assumed to behave in accordance with the protocol specification. This has to be reflected in the model for protocol analysis as well as in the formalisation of security properties.

Remarkably, one protocol proposed by Zhou and Gollmann [28] that has been analysed using three different methods [2, 24, 30] does not provide fair non-repudiation under reasonable assumptions. We show possible attacks on this protocol and on two of its various versions in Sections 4.3, 5.1 and 5.2. Furthermore, in Section 7 we present a straightforward improvement for all three versions of the protocol. In Section 6 we review two optimistic non-repudiation protocols with resolve and abort sub-protocols and show several attacks on these protocols.

The next section defines the security goals of fair non-repudiation. Section 3 explains the protocol proposed by Zhou and Gollmann and Section 4.1 explains our approach to automated protocol analysis. The remaining sections describe possible attacks on three protocol variants. Finally several attacks on two optimistic non-repudiation protocols are presented and improvements are discussed.

This paper is an extended version of [7] and relates it to more recent results firstly presented in [8].

2 Requirements for fair non-repudiation

This paper concentrates on message transfer protocols with certain security properties: Agent A sends message m to agent 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 the recipient B, or vice versa (fairness).

Non-repudiation of origin and non-repudiation of receipt require evidence of receipt (EOR) and evidence of origin (EOO). All agents participating in the protocol have to agree that these 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 agent seems to have an advantage, for example, if a TTP has transmitted evidence first to one agent and the other agent 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 agent has an advantage over the other agent. This means that if there is an unfair intermediate situation for one agent this agent must be able to reach a fair situation without the help of other untrusted agents. For any agent P we say a protocol execution is finished 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 agents.

In this paper we consider a refined version of the definition of fair non-repudiation by Zhou and Gollmann [28]. 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 *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 in 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 in 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*.*

Optimistic non-repudiation protocols shown in Section 6 are designed to provide the following stronger property. This definition below is similar to the definition of fair non-repudiation with strong fairness and timeliness by Kremer, Markowitch and Zhou [14].

Definition 2 *A message transfer protocol for originator A and recipient B provides* fair non-repudiation with timeliness *if it provides fair non-repudiation and the following security goal is satisfied for all possible messages* 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 malicious parties.*

3 The basic version of the Zhou-Gollmann protocol

We first discuss three versions of a non-repudiation protocol introduced by Zhou and Gollmann in [28, 29, 27]. The purpose of all protocols is to transmit a message from agent A to agent B and to provide evidence for B that the message originated with A while conversely providing evidence for A that B received the message. Thus the protocols shall provide fair non-repudiation as defined above. An online trusted third party TTP is involved in all three protocols. Attacks on optimistic variants of the protocols designed to provide timeliness are discussed in Section 6.

The main idea of all protocols is to split the transmission of the message M into two parts. The first part consists of A sending a commitment $C = eK(M)$ (message M encrypted with key K) and B acknowledging its receipt. Then, A submits the key K and signature sub_K to an on-line trusted third party TTP which makes a signature con_K available that serves both as the second part of the evidence of origin for B and as the second part of evidence of receipt for A. Consequently, evidence of origin EOO and evidence of receipt EOR consists of two parts:

- EOO is composed of EOO_C (A's signature on the commitment C) and con_K (the confirmation of key K by the trusted third party).
- EOR is composed of EOR_C (B's signature on the commitment C) and con_K (the confirmation of key K by the trusted third party).

We adopt the notation from Zhou and Gollmann:

- m, n: concatenation of two messages m and n.
- $H(m)$: a one-way hash function applied to message m.
- $eK(m)$ and $dK(m)$: encryption and decryption of message m with key K.
- $C = eK(m)$: commitment (ciphertext) for message m.
- L: a unique label to link all protocol messages.
- $f_{EOO}, f_{EOR}, f_{SUB}, f_{CON}$: message flags to indicate the purpose of the respective message.
- $sS_A(m)$: principal A's digital signature on message m with A's private signature key S_A. Note that the plaintext is not recoverable from the signature, i.e. for signature verification the plaintext needs to be made available.
- $EOO\_C = sS_A(f_{EOO}, B, L, C)$
- $EOR\_C = sS_B(f_{EOR}, A, L, C)$
- $sub\_K = sS_A(f_{SUB}, B, L, K)$
- $con\_K = sS_{TTP}(f_{CON}, A, B, L, K)$
- $A \rightarrow B : m$: agent A sends message m with agent B being the intended recipient.

- $A \leftrightarrow B : m$: agent A fetches message m from agent B using the "*ftp get*" operation (or by some analogous means).

We first concentrate on the basic version of the protocols which is as follows:

1.	A	$\rightarrow$	B	:	f_{EOO}, B, L, C, EOO_C
2.	B	$\rightarrow$	A	:	f_{EOR}, A, L, C, EOR_C
3.	A	$\rightarrow$	TTP	:	f_{SUB}, B, L, K, sub_K
4.	A	$\leftrightarrow$	TTP	:	$f_{CON}, A, B, L, K, con\_K$
5.	B	$\leftrightarrow$	TTP	:	$f_{CON}, A, B, L, K, con\_K$

A protocol without a TTP puts the agent which is the first to provide all information in a disadvantageous position, since the second agent can just refrain from sending the acknowledgement message. This is avoided by involving TTP: Once A made the key available to TTP, A will always be able to retrieve the remaining evidence con_K.

The authors use the following assumptions:

- All agents are equipped with their own private signature key and the relevant public verification keys.
- B cannot block the message identified by f_{SUB} permanently, thus A will eventually be able to obtain the evidence of receipt.
- The *ftp* communication channel is eventually available, thus also B will eventually be able to obtain K and therefore m and con_K.
- TTP checks that A does not send two different keys K and K' with the same label L and the same agents' names. This is necessary because L serves as identifier for con_K, i.e. TTP will overwrite con_K with con_K' which causes a problem if either A or B have not yet retrieved con_K.
- TTP stores message keys at least until A and B have received con_K.

Additionally, A is required to choose a new label and a new key for each protocol run, but except the above check by TTP no means are provided to guarantee this.

Dispute resolution A dispute can occur if B claims to have received m from A while A denies having sent m, or if A claims having sent m to B while B denies having received m. To resolve such a dispute, the evidence of origin and receipt, respectively, has to be sent to a judge who then checks

- that con_K is TTP's signature on (f_{CON}, A, B, L, K), which means that TTP has indeed made the respective entry because of A's message f_{SUB},
- that EOO_C is A's signature on (f_{EOO}, B, L, C) (that EOR is B's signature on (f_{EOR}, A, L, C), respectively)
- that $m = dK(C)$

The authors conclude that the above protocol provides non-repudiation of origin and receipt and fairness for both agents A and B. However, in section 4.3 we will show that the protocol is unfair for B since it allows A to retrieve evidence of receipt for a message m while B is neither able to retrieve m nor the respective evidence of origin. The scenario in which the attack can occur satisfies all assumptions stated above.

4 Protocol analysis using APA and the SHVT

In this section we introduce our approach for security analysis of cryptographic protocols. We model a system of protocol agents using asynchronous product automata (APA). APA are a universal and very flexible operational description concept for cooperating systems [20]. It "naturally" emerges from formal language theory [19]. APA are supported by the SH-verification tool (SHVT) that provides components for the complete cycle from formal specification to exhaustive analysis and verification [20].

4.1 Specification of cryptographic protocols with APA

An APA can be seen as a family of elementary automata. The set of all possible states of the whole APA is structured as a product set; each state is divided into state components. In the following the set of all possible states is called state set. The state sets of elementary automata consist of components of the state set of the APA. Different elementary automata are "glued" by shared components of their state sets. Elementary automata can "communicate" by changing the content of shared state components.

Protocols can be regarded as cooperating systems, thus APA provide adequate means for protocol formalisation. Figure 1 shows the structure of an asynchronous product automaton modeling a system of three protocol agents A, B and TTP. The boxes are elementary automata and the circles represent their state components.

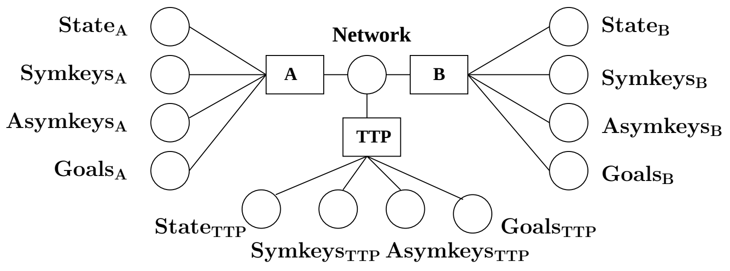

Figure 1: Structure of the APA model for agents A, B and TTP

Each agent P taking part in the protocol is modeled by one elementary automaton P that performs the agent's actions, accompanied by four state components Symkeys_P, Asymkeys_P, State_P, and Goals_P to store the symmetric and asymmetric keys of P, P's local state and the security goals P should reach within the protocol, respectively. The only state component shared between all agents (all elementary automata) is the component Network, which is used for communication. A message is sent by adding it to the content of Network and received by removing it from Network. The neighbourhood relation N (graphically represented by an arc) indicates which state components are included in the state of an elementary automaton and may be changed by a state transition of this automaton. For example, automaton A may change State_A and Network but cannot read or change the state of State_B. The figure shows the structure of the APA. The full specification of the APA includes the state sets (the data types), the transition relations of the elementary automata and the initial state, which we will explain in the following paragraphs.

State sets, messages and cryptography In the present paper we restrict our model to basic data types, and the model of cryptography to those algorithms needed to specify the non-repudiation protocols considered here. For the definition of the domains of the state components as well as for the definition of the set of messages, we need the following basic sets:

$\mathbb{N}$	set of natural numbers
$\mathcal{A}gents$	set of agents' names
$\mathcal{N}once$	set of nonces (numbers that have never been used before)
$\mathcal{C}onstants$	set of constants to determine the agents' states and thus to define state transition relations
$\mathcal{K}eynames$	set of constants to name keys, $\mathcal{A}gents \subseteq \mathcal{K}eynames$
$\mathcal{S}ymflags$	$\{sym, \ldots\}$
$\mathcal{A}symflags$	$\{pub, priv, \ldots\}$
$\mathcal{K}eys$	$\{(w, f, n) \mid w \in \mathcal{K}eynames, f \in \mathcal{S}ymflags \cup \mathcal{A}symflags, n \in \mathbb{N}\}$
$\mathcal{P}redicates$	set of predicates on global states

It is helpful to include the agents' names in the set $\mathcal{K}eynames$ in order to be able to formalise for example the public key of agent P by (P, pub, n) $(n \in \mathbb{N})$. The second component of a key indicates its type. In order to distinguish between different types of keys, more flags (like *pubcipher*, *privcipher*, etc.) can be added to the respective set. The third key component allows to use more than one key with the same name and type. The key K for example in the first run of the Zhou-Gollmann protocol can be formalised by $(K, sym, 1)$, the key of the next run by $(K, sym, 2)$, and so on.

The union of the sets $\mathcal{A}gents$, $\mathcal{N}once$, $\mathcal{C}onstants$, $\mathcal{K}eys$, and $\mathbb{N}$ represents the set of *atomic messages*, based on which we define a set $\mathcal{M}$ of messages in the following way:

1. Every atomic message is element of $\mathcal{M}$.
2. If $m_1, \ldots, m_r \in \mathcal{M}$, then $(m_1, \ldots, m_r) \in \mathcal{M}$.
3. If $k, m \in \mathcal{M}$, then $encrypt(k, m) \in \mathcal{M}$, $decrypt(k, m) \in \mathcal{M}$, $sign(k, m) \in \mathcal{M}$ and $hash(m) \in \mathcal{M}$.

We define the standard functions $elem(k, \ldots)$ and $length$ on tuples $(m_1, \ldots, m_r)$ which return the kth component (or, if $k \geq r$, the rth component) and the number of components of a tuple, respectively.

We model properties of cryptographic algorithms by defining properties of the symbolic functions listed above and by defining predicates. For the analysis of the Zhou-Gollmann protocol we need in particular

1. $decrypt(k, encrypt(k, m)) = m$
2. $inverse((w, sym, n)) = (w, sym, n)$
3. $inverse((P, pub, n)) = (P, priv, n)$
4. $verify((P, pub, n), m, sign((P, priv, n), m)) = true$

(where $k, m \in \mathcal{M}$, $P, w \in \mathcal{K}eynames$, $n \in \mathbb{N}$)

Our general model provides additional symbolic functions for the specification of other cryptographic protocols.

The above properties define for each $m \in \mathcal{M}$ a unique shortest normal form. The set $\mathcal{M}essages$ is the set of all these normal forms of elements $m \in \mathcal{M}$.

Now elements of $\mathcal{M}essages$ constitute the content of State components, while Network contains tuples of $\mathcal{A}gents \times \mathcal{A}gents \times \mathcal{M}essages$, where the first component names the sender of the message and the second component the intended message recipient. In the analysis of a protocol which includes all necessary information (such as to whom to respond) in the messages itself, the first component of the tuple in Network is not evaluated. However, some protocols like the Zhou-Gollmann protocol assume that this information can be retrieved from some lower level transport layer, thus the information has to be provided in addition to the content of the message. An asymmetric key key is stored in $\mathrm{Asymkeys_P}$ using a tuple (Q, f, key), where Q is the name of the agent that P's automaton will use to find the key and $f = pub$ or $f = priv$ is the flag specifying the type of the key. For the formal definition of the data structure of the state components, see [6].

The symbolic functions *encrypt*, *decrypt*, *sign* and *hash* together with the above listed properties model the cryptographic algorithms used in the various versions of the Zhou-Gollmann protocol. For this paper, we assume "perfect encryption", i.e. we assume that keys cannot be guessed, that for generating $encrypt(k, m)$ or $sign(k, m)$, both k and m need to be known, that $encrypt(k, m) = encrypt(k', m')$ and $sign(k, m) = sign(k', m')$ imply $k = k'$ and $m = m'$, and that $hash(m) = hash(m')$ implies $m = m'$.

State transition relation To specify the agents' actions we use so-called *state transition patterns* describing state transitions of the corresponding elementary automaton. Step 2 of the original Zhou-Gollmann protocol where B receives the message f_{EOO} and sends f_{EOR} can be specified as shown in Table 1.

The lines above $\xrightarrow{\mathbf{B}}$ indicate the necessary conditions for automaton B to transform a state transition, the lines behind specify the changes of the state. $\hookrightarrow$ and $\hookleftarrow$ denote that some data is added to and removed from a state component, respectively. B does not perform any other changes within this state transition.

The syntax and semantics of state transition patterns for APA as well as the formal definitions of the state sets is explained in more detail in [5].

A complete specification in the APA framework additionally contains security relevant information. Most important are the security goals the protocol shall reach. The following paragraph explains the formalisation of the security goals defined in Section 2. For more details on a complete protocol specification, we refer the reader to [6].

step 2	Name of the transition pattern
$(A, M, message, plain, L, C,$ $EOO\_C, PKA, SKB)$	Local variables used in the pattern
$M \in$ Network	Variable M is assigned a message in Network.
$A := elem(1, M)$	The variable A is assigned the assumed sender of the message.
$B = elem(2, M)$	B checks that he is the intended recipient.
$(respond, A) \in \mathrm{State_B}$	B can respond to A.
$(A, pub, PKA) \in \mathrm{Asymkeys_B}$	B owns A's public key, the variable PKA is assigned the respective value.
$(B, pub, SKB) \in \mathrm{Asymkeys_B}$	B owns his own private key, the variable SKB is assigned the respective value.
$message := elem(3, M)$	The variable *message* is assigned the data part of M.
$elem(1, message) = f_{EOO}$	B checks that the message contains the f_{EOO} flag.
$elem(2, message) = B$	B checks that he is named in the message.
$L := elem(3, message)$	The variable L is assigned the third message element.
$C := elem(4, message)$	The variable C is assigned the fourth message element.
$plain := (f_{EOO}, B, L, C)$	The variable *plain* is assigned what B assumes to be the plaintext of EOO_C.
$EOO\_C := elem(5, message)$	The variable EOO_C is assigned the signature.
$verify(PKA, plain, EOO\_C) = true$	B verifies EOO_C.
$\xrightarrow{\mathbf{B}}$	state transition is performed by B.
$M \hookleftarrow Network$	The message tuple is removed from Network.
$(f_{EOO}, L, C, expects\_CON,$ $(C, EOO\_C)) \hookrightarrow \mathrm{State_B}$	B stores all relevant data.
$(B, A, (f_{EOR}, A, L, C,$ $sign(SKB, (f_{EOR}, A, L, C)))) \hookrightarrow$ Network	B sends message 2.

Table 1: Detailed specification of step 2 of the Zhou-Gollmann non-repudiation protocol

Security goals In our model, the state components Goals are used to specify security goals. Whenever an agent P performs a state transition after which a specific security goal shall hold from the agent's view, a predicate representing the goal is added to the state of $\mathrm{Goals_P}$. Note that the content of $\mathrm{Goals_P}$ has no influence on the occurence of state transitions.

A protocol is secure (within the scope of our model) if a predicate is true whenever it is element of a Goals component.

In the Zhou-Gollmann protocol, the security goals defined in Definition 1 in Section 2 can now be concretised. As the first goal for example states that at the end of a protocol execution by A, either A owns EOR or B does not own EOO, any state in which B owns EOO must allow A to continue the protocol execution and receive EOR without the help of an untrusted agent. This gives rise to the following definitions:

- For originator A the predicate *NRR(B)* is true if for any message m the following holds:

  If EOO_C for m signed by A and also a matching con_K are elements of $\mathrm{State_B}$, then EOR_C for m signed by B is element of $\mathrm{State_A}$ and either a matching con_K is in $\mathrm{State_A}$, or con_K is made available by TTP and not yet retrieved by A.

- For a recipient B the predicate *NRO(A)* is true if for any message m the following holds:

  If *EOR_C* for m signed by B and a matching *con_K* are elements of State_A, then *EOO_C* for m signed by A is element of State_B and either a matching *con_K* is in State_B, or *con_K* is made available by *TTP* and not yet retrieved by B.

The predicates *NRR(B)* and *NRO(A)* have to be satisfied in all possible states in all protocol runs. They can therefore be included in the initial state of Goals_A and Goals_B, respectively.

Initial state The initial state for the Zhou-Gollmann protocol can now be specified as shown in Table 2.

$$
\begin{array}{lcl}
\text{State}_\text{A} & := & \{(\mathbf{B}, \mathit{agent}), (\mathbf{TTP}, \mathit{server}), (\mathit{start}, \mathbf{B}), (m1, m2, \mathit{message})\} \\
\text{Asymkeys}_\text{A} & := & \{(\mathbf{A}, \mathit{priv}, (\mathbf{A}, \mathit{priv}, 1)), (\mathbf{B}, \mathit{pub}, (\mathbf{B}, \mathit{pub}, 1)), \\
 & & (\mathbf{TTP}, \mathit{pub}, (\mathbf{TTP}, \mathit{pub}, 1))\} \\
\text{Goals}_\text{A} & := & \{\mathit{NRR}(\mathbf{B})\} \\
 & & \\
\text{State}_\text{B} & := & \{(\mathbf{A}, \mathit{agent}), (\mathbf{TTP}, \mathit{server}), (\mathit{respond}, \mathbf{A})\} \\
\text{Asymkeys}_\text{B} & := & \{(\mathbf{B}, \mathit{priv}, (\mathbf{B}, \mathit{priv}, 1)), (\mathbf{A}, \mathit{pub}, (\mathbf{A}, \mathit{pub}, 1)), \\
 & & (\mathbf{TTP}, \mathit{pub}, (\mathbf{TTP}, \mathit{pub}, 1))\} \\
\text{Goals}_\text{B} & := & \{\mathit{NRO}(\mathbf{A})\} \\
 & & \\
\text{State}_\text{TTP} & := & \{(\mathbf{A}, \mathit{agent}), (\mathbf{B}, \mathit{agent})\} \\
\text{Asymkeys}_\text{TTP} & := & \{(\mathbf{TTP}, \mathit{priv}, (\mathbf{TTP}, \mathit{priv}, 1)), (\mathbf{A}, \mathit{pub}, (\mathbf{A}, \mathit{pub}, 1)), \\
 & & (\mathbf{B}, \mathit{pub}, (\mathbf{B}, \mathit{pub}, 1))\}
\end{array}
$$

Table 2: The initial state

The tuple $(m1, m2, \mathit{message})$ represents a reservoir of messages that A can send, the agents' Symkey components as well as Network are empty in the initial state.

4.2 APA framework for protocol analysis

The APA specification of a protocol as described above specifies the behaviour of roles in the protocol. This specification can be used to analyse the correctness of the protocol without any malicious behaviour or interference between protocol runs. For security analysis, the role specification has to be transferred to specific analysis scenarios. The APA framework provides various means to specify concrete analysis scenarios which are then automatically generated from the role specification by the SHVT. The scenarios include the specification of the number and nature of runs that shall be validated (only finitely many runs can be checked), the concrete agents taking part in these runs and their roles (we may want to analyse a scenario with Alice and Bob both acting as A and Bob acting as B), which of the agents may act dishonestly, and other details.

Interpretation of roles Since in the APA specification, A,B, etc. represent roles rather than agents, $(\mathbf{B}, \mathit{agent})$ in A's State component in the initial state does not mean that an agent in role A only knows one single agent in role B. Rather, $(\mathbf{B}, \mathit{agent})$ represents a subset of $\mathcal{A}gents \times \{\mathit{agent}\}$ which contains one element for each agent that can act in role B. Thus, $(\mathbf{B}, \mathit{agent})$ can be viewed as a characteristic element for the possible interpretations. If, for example, Alice and Bob can act in role B, then $(\mathbf{B}, \mathit{agent})$ represents the set $\{\mathit{(Alice,agent)}, \mathit{(Bob,agent)}\}$.

To derive a concrete scenario for the analysis, the characteristic elements of the role specification are replaced by the respective sets. The initial state for a state component is then the result of the union of all these sets.

The analysis APA Every concrete agent P acting honestly (i.e. according to the protocol specification) is modeled by one elementary automaton P and the state components State_P, Symkeys_P, Asymkeys_P, Goals_P and Network in its neighbourhood relation. The elementary automaton P performs state transitions according to the patterns given in the protocol specification. Additional to Network, an analysis APA can contain global state components that can be accessed by all elementary automata. These state components can be used for the generation of random numbers, for the storage of data that can be used to interrupt a protocol run, etc.

The resulting APA together with a specification of analysis details (which are the agents acting, how many runs, etc.) can be used to check that the protocol specification without malicious behaviour results in the desired state transition sequence and that in particular the expected final states of the agents are reached.

For more details on the analysis APA, see [4].

Including dishonest behaviour In order to perform a security analysis, our model includes the explicit specification of dishonest behaviour. For each type of dishonest behaviour, the APA includes one elementary automaton with the respective state components and state transition relations for specifying the concrete actions. (For the protocols discussed in this paper, we consider only two types of dishonest behaviour, namely the actions of a dishonest agent acting either in role A or in role B. However, other protocols may require to differentiate between actions of different dishonest agents.)

The elementary automaton of a dishonest agent can remove all tuples from Network independently from being named as the intended recipient. It can extract the first and second component of the tuples and add them as possible sender and recipient of messages to be sent by itself. (Note that it can use the name of any agent it knows as the sender of its own messages.) Furthermore, it can extract new knowledge from the messages and add this knowledge to the respective state components. A dishonest agent's knowledge can be defined recursively in the following way:

1. A dishonest agent knows all data being part of its initial state (for example the names of other protocol agents, the keys it owns, etc.) and can generate random numbers and new keys.
2. It knows all messages and parts of messages, the names of sender and the intended recipient of messages it receives or maliciously removes from Network.
3. It can generate new messages by applying concatenation or the symbolic functions *encrypt*, *sign* and *hash* to messages it knows. (Note that it can use any message in its knowledge base as a key.)
4. It knows all plaintexts it can generate by deciphering a ciphertext it knows, provided it knows the necessary key as well.

With new messages generated according to the above rules, in every state of the system, the dishonest agent's elementary automaton can add new tuples (*sender,recipient,message*) to Network, *sender* and *recipient* being one of the agents' names it has stored in its State component.

4.3 An attack on the Zhou-Gollmann protocol

We now introduce a concrete scenario for the Zhou-Gollmann protocol. Using the SH verification tool we want to automatically analyse a scenario where Alice (in role A) starts a protocol run with Bob (in role B), and *Server* acts in role TTP. Furthermore we assume that Alice may act dishonestly.

The following details of the analysis scenario are of particular interest:

1. In order to model the assumptions made by Zhou and Gollmann that B cannot block the f_{SUB} message permanently and that A and B are eventually able to retrieve the f_{CON} message, we restrict dishonest behaviour in that we do not allow Alice to remove messages from Network containing an intended recipient other than herself.
2. Zhou and Gollmann do not require the TTP to provide messages f_{CON} forever (see [28]). A reasonable assumption is that these messages are available at least until A and B each have retrieved their message, and that the TTP then may delete them [1]. We simply model this by letting Server add these messages

[1]We assume that the TTP "magically" knows who has received *con_K*. In practical implementations either the TTP needs some confirmation of receipt by A and B before deleting the stored key or a time limit may have to be set.

to Network. Since messages can only be removed from Network from the agent named as the intended recipient (assumption 1), these messages stay in Network until Alice and Bob remove them. Therefore, Server can delete f_{CON} immediately after sending.

3. Server's check that the same label L may not be used together with two different keys can easily be modeled by storing a tuple (A, B, L, K) in the Server's State component for each protocol run. Thus messages f_{SUB} that contain L and K already being part of one tuple can be rejected.

4. We assume that all signatures are checked by the recipients. Thus messages with incorrect signatures will be rejected. In consequence we restrict Alice's behaviour further in allowing her only to send messages with correct signatures. Other than that, Alice may send anything she knows as L, K and $eK(m)$.

We then want to analyse whether there is a reachable state in which the security goal *NRO(A)* in Goals_B is not satisfied.

Indeed, the SHVT finds such a state. Starting with the initial state, the SHVT computes all reachable states until it finds a state in which Alice owns EOR_C and con_K for a particular message, while Bob is not able to get con_K for this message. The SHVT outputs the state indicating a successful attack. Now one can let the SHVT compute a path from the initial state to this attack state, showing how the attack works. In the following, these steps are explained.

In the first step, Alice generates a new label L and key K, stores these data in her State component for later use, and starts a first protocol run with message $m1$. For the rest of the protocol run, Alice acts honestly and the protocol run proceeds according to the protocol description. At the end of the protocol run, Alice owns in her state component $\text{State}_\text{Alice}$ both Bob's *EOR_C* and Server's *con_K* for $m1, L$ and K:

$$\begin{aligned}\text{State}_\text{Alice} \;=\; & \{\ldots, (m1, m2, \mathit{messages}), (\ldots, K, L, \ldots),\\ & (\ldots, \mathit{sign}((\mathit{Bob}, \mathit{priv}, 1), (f_{EOR}, \mathit{Alice}, L, \mathit{encrypt}(K, m1))),\\ & \mathit{sign}((\mathit{Server}, \mathit{priv}, 1), (f_{CON}, \mathit{Alice}, \mathit{Bob}, L, K)))\}\end{aligned}$$

Alice can now start the next protocol run. Among the possibilities to do this is one state transition in which she chooses $m2$ as the next message to send, but does not generate a new label and a new key. Instead, she uses L and K she has stored to use in an attack. Thus the state component Network contains the following data:

$$\begin{aligned}\text{Network} \;=\; & \{(\mathit{Alice}, \mathit{Bob}, (f_{EOO}, \mathit{Bob}, L, \mathit{encrypt}(K, m2),\\ & \mathit{sign}((\mathit{Alice}, \mathit{priv}, 1), (f_{EOO}, \mathit{Bob}, L, \mathit{encrypt}(K, m2)))))\}\end{aligned}$$

Bob may still own an *EOO* tuple for L, K and $m1$ in State_Bob and may therefore be able to decrypt the ciphertext $\mathit{encrypt}(K, m2)$. However, the protocol specification does not require him to check this and Bob has no reason to try any old keys on the new message. The assumption that Bob stores all proofs he ever received is quite unrealistic in any case, thus Bob may have deleted the particular key and *EOO*. Consequently, Bob answers with the f_{EOR} message including *EOR_C* for $m2$, which results in

$$\begin{aligned}\text{Network} \;=\; & \{(\mathit{Bob}, \mathit{Alice}, (f_{EOR}, \mathit{Alice}, L, \mathit{encrypt}(K, m2),\\ & \mathit{sign}((\mathit{Bob}, \mathit{priv}, 1), (f_{EOO}, \mathit{Alice}, L, \mathit{encrypt}(K, m2)))))\}\end{aligned}$$

Now, since Alice still owns *con_K* for L and K, she owns a valid proof of Bob having received message $m2$ that will be accepted by any judge, and stops the protocol run. Bob on the other hand does neither own the key K nor the Server's signature (or does not know that he owns these data). Thus, security goal *NRO(A)* is not satisfied and Bob in fact will never be able to retrieve $m2$. This shows that while much care has been taken to assure fairness of the protocol for A, the protocol does not provide fairness for B.

4.4 Other analysis approaches

Although this protocol and in particular its fairness has been analysed with two different analysis methods [24, 2], this weakness was not discovered. Bella and Paulson have used the inductive analysis approach based on the theorem prover Isabelle [21] which had been developed for the analysis of authentication and key establishment protocols. They cannot find the above attack, because in their model the only malicious action of the protocol agents consists of abandoning protocol sessions before termination. This is not a realistic attack scenario for non-repudiation protocols. In addition, in inductive protocol specifications as proposed by Paulson, et al., agents cannot "forget" data. Although agents can explicitly memorize data using the *notes* event, there is no event type to remove this data from an agent's memory. In principle it is possible to extend inductive protocol specification in order to include deletion of data from an agent's memory. However, it is an open question whether security proofs remain feasible in this extended model. The major difficulty of such an extension is, that monotinic properties of inductive specifications are lost. Therefore, we expect security proofs to be more difficult.

A different analysis by Schneider uses CSP [24]. This approach has also been used to analyse authentication and key establishment protocols [23, 22]. The scenario in which the security proofs are carried out is more realistic than the scenario of Bella and Paulson. The behaviour of the originator A and recipient B is not restricted. They can execute all possible malicious actions. Even so, the attack does not exist in the scenario because of the rather unrealistic assumption that evidence CON_K and the keys remain available for download at the TTP forever. Similar to Paulson's approach, Schneider restricts the model to monotonic growth of the agents' memory content. Again, abandoning this restriction might complicate security proofs.

It remains as an interesting open question whether security proofs for non-repudiation protocols using a theorem prover are feasible if realistic analysis scenarios are used.

The protocol analysis performed by Zhou and Gollmann themselves [30] uses a modified version of the authentication logic SVO [25]. This logic (like all other authentication logics) cannot express the property of fairness, as stated by the authors of [30], consequently their analysis does not find the protocol weakness.

5 Variants of the Zhou-Gollmann protocol

5.1 Unique labels

Obviously, a critical point of the protocol is how to choose the label L. In a different version of the protocol, Zhou and Gollmann suggest to use $L = H(m, K)$ [28, 27]. This guarantees that whenever a new message is sent, the message will be accompanied by a new label, even if the same key K is used. The actions performed by the agents are essentially the same with the exception of label generation and that the label check performed by TTP is now obsolete. In a dispute, the judge will additionally check that $L = H(dK(C), K)$.

Unfortunately the change of label generation does not avoid the unfairness of the protocol. We again model a scenario with dishonest Alice acting in role A and Bob acting in role B. Since the hash values are checked by a judge we model Alice by requiring to use the hash function for label generation, but we allow Alice to use anything she knows as parameters. Thus, Alice initiates the protocol with a message including $H(m2, K)$ and $eK(m1)$ and the respective signature:

$$\begin{aligned}\text{Network} \quad = \quad & \{(\mathit{Alice},\mathit{Bob},(f_{EOO},\mathit{Bob},H(m2,K),\mathit{encrypt}(K,m1),\\ & \mathit{sign}((\mathit{Alice},\mathit{priv},1),(f_{EOO},\mathit{Bob},H(m2,K),\mathit{encrypt}(K,m1))))\}\end{aligned}$$

Bob removes the message tuple from Network and proceeds according to the description in Section 3. The protocol run ends with each Alice and Bob owning $sS_{Server}(f_{CON}, Alice, Bob, H(m2, K), K)$. Additionally, Alice owns $sS_{Bob}(f_{EOR},\ Alice, H(m2, K), eK(m1))$, and Bob owns Alice's respective signature. However, these do not present a valid proof for Bob that Alice has sent $m1$ nor a proof for Alice that Bob has received $m1$, as a judge would find that the label used in the signatures is not equal to $H(m1, K)$. Alice knows this (after all, she generated the label), but Bob may not know it if he is not required to make the respective check after the last protocol step.

Now Alice can start a second protocol run in which she uses the same label $H(m2, K)$ and key K, but this time indeed sends the enciphered message $m2$:

$$\begin{aligned}\text{Network} \quad = \quad & \{(\mathit{Alice},\mathit{Bob},(f_{EOO},\mathit{Bob},H(m2,K),\mathit{encrypt}(K,m2),\\ & \mathit{sign}((\mathit{Alice},\mathit{priv},1),(f_{EOO},\mathit{Bob},H(m2,K),\mathit{encrypt}(K,m2))))\}\end{aligned}$$

Bob answers by sending his f_{EOR} message. Now, Alice owns a valid EOR_C of Bob for $m2$ and, from the first protocol run, con_K for $H(m2, K)$ and K. Hence Alice owns a valid proof that Bob received message $m2$, while again Bob has no chance to ever retrieve $m2$ or the Server's signature. Alice now stops the protocol run.

5.2 Time-stamping evidence

In [27] Zhou et al. propose to use time stamps generated by TTP in evidence con_K to identify when the key and thus the message was made available. The protocol remains the same except for the additional time stamp T_{con} in $con\_K = sS_{TTP}(f_{CON}, A, B, L, T_{con}, K)$.

In addition to fair non-repudiation (Definition 1) this protocol shall satisfy another goal: The time stamp T_{con} is supposed to identify the time when the message is made available to B.

To model non-repudiation with time stamps we have introduced a discrete time model to our framework. An additional elementary automaton $Time$ increases a natural number in a state component T. Assumptions about the agents' behaviour relative to time can be modelled by the behaviour of the automaton $Time$. In the non-repudiation protocol example, we assume that when B expects to retrieve con_K and con_K is already made available by TTP then B retrieves con_K within the actual time slot. In this case T is only increased by $Time$ after B has retrieved con_K.

In this scenario the security goal for B is that whenever B retrieves con_K, the time stamp signed by the TTP in con_K must be the actual time contained in state component T.

In the scenario where Bob (in role B) acts honestly as described above, Alice (in role A) can execute protocol steps 3 and 4 without previous execution of steps 1 or 2 and receive a time-stamped con_K with a specific T_{con}. Alice can even collect several different time stamps for the same message. Later, Alice starts the protocol as usual with step 1 at time $T > T_{con}$. Bob responds with EOR_C. As step 3 and 4 have already happened, Alice terminates the protocol run after step 2 and owns a valid evidence of receipt for time $T_{con} < T$, although the message was not available for Bob before time T.

In [26] where a more elaborated version of this protocol is introduced, Zhou has pointed out that by sending sub_K before receiving EOR_C, Alice enables Bob to get the message and evidence of origin without providing any evidence of receipt. However, as Bob cannot know that Alice has already submitted sub_K to TTP he has no reason to retrieve it from the TTP, and if he retrieves it, he only gets evidence of origin containing an old time stamp.

6 Attacks on optimistic fair non-repudiation protocols

In this section we show different attacks on two optimistic non-repudiation protocols that are supposed to provide fairness and respect timeliness in accordance with Definitions 1 and 2. These attacks have first been published by the authors together with Holger Vogt [8]. One protocol was proposed by Kremer and Markowitch [12, 14] and has been analyzed in [11]. Several other protocols [17, 18, 13] build on this protocol. The second protocol is a very similar one proposed by Zhou, Deng and Bao [27, 33], which has already been analyzed and improved in [3]. Both protocols (which we will call KM protocol and ZDB protocol, respectively, in the remainder of this paper) use an offline TTP, i.e. a TTP which 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 [28]. 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 Definitions 1 and 2 and both realizations are based on the same ideas.

6.1 The ZDB Protocol

For the description of the two protocols we use the same notation as in Section 3, and the following ones:

- $f_1, f_2, \ldots, f_8$: message flags to indicate the purpose of the respective message.
- $eP_{TTP}(K)$ encryption of key K with TTP's public key.
- $EOO\_C = sS_A(f_1, B, L, C)$: evidence of origin of C.
- $EOR\_C = sS_B(f_3, A, L, EOO\_C)$: evidence or receipt of C.
- $EOO\_K = sS_A(f_3, B, L, K)$: evidence of origin of K.

- $\mathit{EOR\_K} = sS_B(f_4, A, L, \mathit{EOO\_K})$: evidence or receipt of K.
- $\mathit{sub\_K} = sS_A(f_5, B, L, K, TTP, \mathit{EOO\_C})$: evidence of submission of K to the TTP.
- $\mathit{con\_K} = sS_{TTP}(f_6, A, B, L, K)$: evidence of confirmation of K by the TTP.
- $\mathit{abort} = sS_{TTP}(f_8, A, B, L)$: evidence of abortion. being the intended recipient.

The protocol consists of one main *exchange* protocol and two sub-protocols *abort* and *resolve*. The *exchange* sub-protocol is as follows:

```
1.  $A \rightarrow B : f_1, f_5, B, L, C, TTP, eP_{TTP}(K), \mathit{EOO\_C}, sub_K$
2. IF $B$ gives up THEN quit ELSE
    $B \rightarrow A : f_2, A, L, \mathit{EOR\_C}$
3. IF $A$ gives up THEN abort ELSE
    $A \rightarrow B : f_3, B, L, K, \mathit{EOO\_K}$
4. IF $B$ gives up THEN resolve ELSE
    $B \rightarrow A : f_4, A, L, \mathit{EOR\_K}$
5. IF $A$ gives up THEN resolve
```

The *abort* sub-protocol which can only be performed by A is as follows:

```
1.  $A \rightarrow TTP : f_7, B, L, sS_A(f_7, B, L)$
2. IF resolved THEN
      $TTP \rightarrow A : f_2, f_6, A, B, L, K, \mathit{con\_K}, \mathit{EOR\_C}$
    ELSE
      $TTP \rightarrow A : f_8, A, B, L, \mathit{abort}$
```

The *resolve* sub-protocol is as follows, where the initiator U is either A or B:

```
1.  $U \rightarrow TTP : f_2, f_5, A, B, L, TTP, eP_{TTP}(K), \mathit{sub\_K}, \mathit{EOO\_C}, \mathit{EOR\_C}$
2. IF aborted THEN
      $TTP \rightarrow U : f_8, A, B, L, \mathit{abort}$
    ELSE
      $TTP \rightarrow U : f_2, f_6, A, B, L, K, \mathit{con\_K}, \mathit{EOR\_C}$
```

6.2 The KM-Protocol

- $f_{EOO}, f_{EOR}, f_{Sub}, \ldots$: message flags to indicate the purpose of the respective message.
- $EOO = sS_A(f_{EOO}, B, TTP, L, H(C))$: evidence of origin of C.
- $EOR = sS_B(f_{EOR}, A, TTP, L, H(C))$: evidence or receipt of C.
- $Sub = sS_A(f_{Sub}, B, L, eP_{TTP}(K))$: evidence of submission of K to the TTP.
- $\mathit{EOO\_K} = sS_A(f_{EOO_K}, B, L, K)$: evidence of origin of K.
- $\mathit{EOR\_K} = sS_B(f_{EOO_R}, A, L, K)$: evidence or receipt of K.
- $Rec_X = sS_X(f_{Rec_X}, Y, L)$: recovery request.
- $\mathit{con\_K} = sS_{TTP}(f_{Con_K}, A, B, L, K)$: evidence of confirmation of K by the TTP.
- $\mathit{abort} = sS_A(f_{Abort}, B, L)$: abort request.
- $Con_a = sS_{TTP}(f_{Con_a}, A, B, L)$: evidence of abort confirmation.

The protocol also consists of one main *exchange* protocol and two sub-protocols *abort* and *resolve*. The *exchange* protocol is as follows:

1. $A \rightarrow B : f_{EOO}, f_{Sub}, B, TTP, L, C, eP_{TTP}(K), EOO, sub$
2. IF B gives up THEN quit ELSE
   $B \rightarrow A : f_{EOR}, A, TTP, L, EOR$
3. IF A gives up THEN abort ELSE
   $A \rightarrow B : f_{EOO_K}, B, L, K, EOO\_K$
4. IF B gives up THEN resolve ELSE
   $B \rightarrow A : f_{EOR_K}, A, L, EOR\_K$
5. IF A gives up THEN resolve

The *abort* sub-protocol also can only be performed by A, but involves B as well:

1. $A \rightarrow TTP : f_{Abort}, L, B, Abort$
2. IF resolved OR aborted THEN stop ELSE
   $TTP \rightarrow A : f_{Con_a}, A, B, L, Con_a$
   $TTP \rightarrow B : f_{Con_a}, A, B, L, Con_a$

The *resolve* sub-protocol is as follows, where X and Y are either A or B:

1. $X \rightarrow TTP : f_{Rec_X}, f_{Sub}, Y, L, H(C), eP_{TTP}(K), Rec_X, Sub, EOR, EOO$
2. IF aborted OR recovered THEN stop ELSE
   $TTP \rightarrow A : f_{Con_K}, A, B, L, K, con\_K, EOR$
   $TTP \rightarrow B : f_{Con_K}, A, B, L, K, con\_K$

6.3 Attacks on the ZDB-protocol and the KM-protocol

In this section we first describe the general idea of the two protocols. Then we explain the differences and possible attacks.

- 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 and receipt for C are exchanged (EOO_C and EOR_C, essentially the respective signatures on L and C), A sends K and EOO_K to B (evidence of origin for K, essentially A's signature on L and C). B can then decrypt M and returns evidence of receipt for K (EOR_K, essentially B's signature on L and C).
- Two sub-protocols *abort* and *resolve* involving a trusted third party TTP shall provide fairness and timeliness.
- The *abort* sub-protocol can be invoked by A at any stage of the protocol. If no *resolve* has happened before, the TTP confirms the *abort* and a future *resolve* is prevented for this combination of A, B and label L.
- Missing evidence of origin or receipt of key K and the missing key itself can be obtained by either party by using the *resolve* sub-protocol. As the first message of the main protocol includes K encrypted with the public key of the TTP ($eP_{TTP}(K)$), 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.

6.3.1 Reuse of $eP_{TTP}(K)$ in a different context

The first message of the main parts of both protocols essentially consists of L, C, the respective EOO_C, $eP_{TTP}(K)$, and a signature sub_K by A which can be used by B in the *resolve* sub-protocol. 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 in the ZDB protocol. In the KM protocol, sub_K contains only items sent as parts of the first message, in particular it contains $eP_{TTP}(K)$, while in the ZDB protocol the key K itself is signed. Thus in the KM protocol B can produce a valid sub_K for a different context, i.e. with the same $eP_{TTP}(K)$ but different parties and label. By using this sub_K in a *resolve* sub-protocol after having received EOO_C, B gains K and A does not receive any evidence of receipt. Thus, the protocol is unfair for A. This attack is not possible in the ZDB protocol because knowledge of K is necessary to generate a valid sub_K.

6.3.2 Sending wrong sub_K or EK

On the other hand, signing the plaintext K has the drawback that B is not able to check the validity of sub_K. This enables a different attack on the ZDB protocol published by Boyd and Kearney [3] which is not possible in the KM protocol. By sending an invalid sub_K A can prevent the termination of B. Even the improved version proposed in [3] is susceptible to a similar attack where A sends a wrong $eP_{TTP}(K)$. This again prevents B from terminating. In [31] 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.

6.3.3 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 Section 5. 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 Section 4. 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 sub-protocol.

- 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 that 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.

- There is no obvious reason for B to store evidence from past protocol runs, especially if the protocol run was not completed because of a wrong label.

- In the second protocol run A can send wrong $eP_{TTP}(K)$ and sub_K to prevent a successful resolve by B.

This attack may also be successful against related protocols proposed in [27, 33, 3, 31].

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.

7 Improving fair non-repudiation protocols

The attack on the Zhou-Gollmann protocol and its variants is very easy to prevent. We show one approach in Section 7.1. Improving the ZDB- and MK-protocols is more complicated, as different attacks are based on diverse protocol weaknesses and subtle differences between the protocols. Generic design principles have been presented in [8]. These design principles include recommendations on the construction of labels, on the context of messages and on the details of specification of TTP behaviour. Based on these principles a new optimistic fair non-repudiation protocol is presented in [8] that is immune to all attacks shown in this paper.

7.1 Improving the Zhou-Gollmann protocol

Obviously, the problem with all three protocol variants of the Zhou-Gollmann protocol is that B cannot control the connection between the different parts of the evidence. We suggest to improve the protocol by letting B introduce his own label L_B when receiving the first message of the protocol and including this label in all subsequent messages. Thus, the new specification of EOR_C, sub_K and con_K is as follows:

- $EOR\_C = sS_B(f_{EOR}, A, L, L_B, C)$
- $sub\_K = sS_A(f_{SUB}, B, L, L_B, K)$
- $con\_K = sS_{TTP}(f_{CON}, A, B, L, L_B, K)$

All three attacks are prevented because A cannot reuse con_K for a different message and A is not able to get a valid con_K before step 2 was executed as A cannot guess L_B.

7.2 The new optimistic fair non-repudiation protocol

The main differences of the protocol presented in [8] to the KM and ZDB protocol are as follows:

- The protocol uses a label $L = H(A, B, TTP, H(C), H(K)$ which is verifiable by all parties, as C and $H(K)$ are contained in the first message of the main protocol.
- All signatures include this label, thus linking all messages of one protocol execution.
- $eP_{TTP}(f_K, L, K)$ instead of only $eP_{TTP}(K)$ is used, so the TTP can check that this is the key K used in the protocol execution identified by L.

In consequence, the problems of the various protocols discussed in this paper are avoided.

8 Relevance of the attacks

One can easily construct scenarios in which the attacks described in this paper are not possible. However, the requirements for these scenarios are not very intuitive. The following observations motivate the choice of our analysis scenario:

1. After the protocol is finished there should be no need for the TTP to keep evidence available for retrieval by protocol agents. Fairness of future protocol runs must not rely on evidence from past protocol runs stored at the TTP, unless the protocol specification explicitly mentions respective actions.

   It is obvious that the TTP cannot delete evidence con_K before both Alice and Bob have retrieved it, as in this case the protocols cannot be fair. On the other hand, apart from dispute resolution, no further protocol steps are carried out after Alice and Bob have received their pieces of evidence. As the TTP does not participate in the dispute resolution there is no obvious need to store any evidence after completion of the protocol.

   Although any real-world TTP may store a permanent log of all transactions this log cannot prevent any of the attacks. Data in the log is not available for further protocol executions, and subsequent investigation cannot detect any misbehaviour of Alice because the TTP is not involved in the second (malicious) protocol run.

2. Any agent must keep evidence as long as necessary to resolve disputes about the particular protocol run. However, at some point the agent will consider the respective transaction completed, thus from then on the evidence is no longer relevant. Thus it is not reasonable to require that evidence has to be kept forever to be used in future protocol runs.

The time attack in Section 5.2 does not require any of the assumptions above. The attack may occur even if evidence is kept forever by the TTP.

9 Conclusions

In this paper we have demonstrated our method for security analysis of cryptographic protocols using three variants of a non-repudiation protocol proposed by Zhou and Gollmann and two optimistic non-repudiation protocols designed to provide timeliness. We have shown possible attacks on all these protocols. The attacks illustrate the need for more detailed protocol specifications as basis for security analysis and secure implementations.

The security analyses were carried out using the SH-verification tool [20]. No attack was found for the improved version of the protocol proposed in Section 7. Although no attacks where found, other attacks may exist in scenarios we have not checked.

Our methods do not provide proofs of security, but are similar to model checking analysis approaches where a finite state space is searched for insecure states (see, for example, [16] or [22]). Compared to other approaches, our methods are both very flexible and minimal with respect to implicit assumptions (we use "perfect encryption" and assume that no unauthorised access to agents' memory is possible) and use more detailed protocol specifications. We expect that the attacks can also be found using other formal analysis methods if the specification of the protocol is not too abstract and if the attacks are not hidden by implicit assumptions.

The examples show that although a protocol has been carefully studied and proven to be secure there may still be unknown attacks. Consequently, security proofs have to be treated with care. Such proofs could be based on improper explicit or implicit assumptions. It remains as an interesting open question whether security proofs for non-repudiation protocols using a theorem prover are feasible in realistic analysis scenarios.

References

[1] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In Tsutomu Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 6–17, Zürich, Switzerland, April 1997. ACM Press.

[2] G. Bella and L. Paulson. Mechanical proofs about a non-repudiation protocol. In *Proceedings of 14th International Conference on Theorem Proving in Higher Order Logic*, Lecture Notes in Computer Science, pages 91–104. Springer Verlag, 2001.

[3] Colin Boyd and Peter Kearney. Exploring fair exchange protocols using specification animation. In *Information Security – ISW 2000*, volume 1975 of *Lecture Notes in Computer Science*, pages 209–223, Wollongong, Australia, December 2000. Springer-Verlag.

[4] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Role based specification and security analysis of cryptographic protocols using asynchronous product automata. GMD Report 151, GMD – Forschungszentrum Informationstechnik GmbH, 2001.

[5] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and provability, a formal framework. In *Infrastructure Security Conference InfraSec 2002*, volume 2437 of *Lecture Notes in Computer Science*, pages 227–245. Springer Verlag, 2002.

[6] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Role based specification and security analysis of cryptographic protocols using asynchronous product automata. In *DEXA 2002 International Workshop on Trust and Privacy in Digital Business*. IEEE, 2002.

[7] S. Gürgens and C. Rudolph. Security analysis of (un-) fair non-repudiation protocols. In *Formal Aspects of Security (FASec '02)*, Lecture Notes in Computer Science. BCS-FACS, Springer Verlag, 2002.

[8] S. Gürgens, C. Rudolph, and H.Vogt. On the security of fair non-repudiation protocols. In W. Mao and C. Boyd, editors, *6th International Conference on Information Security – ISC 2003*, Lecture Notes in Computer Science. Springer Verlag, 2003.

[9] Sigrid Gürgens and Carsten Rudolph. Security analysis of (un-) fair non-repudiation protocols. In *Formal Aspects of Security 2002*, 18–20 December 2002. 18 pages.

[10] K. Kim, S. Park, and J. Baek. Improving fairness and privacy of zhou-gollmann's fair non-repudiation protocol. In *Proceedings of 1999 ICPP Workshop on Security*, pages 140–145, 1999.

[11] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *Proceedings of 12th International Conference on Concurrency Theory*, Lecture Notes in Computer Science, pages 551–565. Springer Verlag, 2001.

[12] Steve Kremer and Olivier Markowitch. Optimistic non-repudiable information exchange. In J. Biemond, editor, *21st Symposium on Information Theory in the Benelux*, pages 139–146, Wassenaar, The Netherlands, May 2000. Werkgemeenschap Informatieen Communicatietheorie, Enschede.

[13] Steve Kremer and Olivier Markowitch. Selective receipt in certified e-mail. In *Progress in Cryptology – INDOCRYPT 2001*, number 2247 in Lecture Notes in Computer Science, pages 136–148, Chennai, India, 16–20 December 2001. Springer-Verlag.

[14] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An intensive survey of fair non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, November 2002.

[15] Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *CONCUR 2001 – Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 551–565, Aalborg, Denmark, August 2001. Springer-Verlag.

[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *Second International Workshop, TACAS '96*, volume 1055 of *LNCS*, pages 147–166. SV, 1996.

[17] Olivier Markowitch and Steve Kremer. A multi-party optimistic non-repudiation protocol. In *Information Security and Cryptology – ICISC 2000*, volume 2015 of *Lecture Notes in Computer Science*, pages 109–122, Seoul, Korea, December 2000. Springer-Verlag.

[18] Olivier Markowitch and Steve Kremer. An optimistic non-repudiation protocol with transparent trusted third party. In *Information Security – ISC 2001*, volume 2200 of *Lecture Notes in Computer Science*, pages 363–378, Malaga, Spain, October 2001. Springer-Verlag.

[19] P. Ochsenschläger, J. Repp, and R. Rieke. Abstraction and composition – a verification method for co-operating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459, 2000.

[20] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche. The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The Int. Journal of Formal Methods*, 11:1–24, 1999.

[21] L. C. Paulson. Proving Properties of Security Protocols by Induction. Technical Report 409, Computer Laboratory, University of Cambridg, 1996.

[22] B. Roscoe, P. Ryan, S. Schneider, M. Goldsmith, and G. Lowe. *The modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

[23] S. Schneider. Verifying authentication protocols with CSP. In *IEEE Computer Security Foundations Workshop*. IEEE, 1997.

[24] S. Schneider. Formal Analysis of a non-repudiation Protocol. In *IEEE Computer Security Foundations Workshop*. IEEE, 1998.

[25] P.F. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *IEEE Symposium on Security and Privacy*, pages 14–28, May 1994.

[26] J. Zhou. *Non-repudiation in Electronic Commerce*. Computer Security Series. Artech House, 2001.

[27] J. Zhou, R. Deng, and F. Bao. Evolution of fair non-repudiation with TTP. In *Proceedings of 1999 Australasian Conference on Information Security and Privacy ACISP*, Lecture Notes in Computer Science, pages 258–269. Springer Verlag, 1999.

[28] J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 55–61, Oakland, CA, 1996. IEEE Computer Society Press.

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

[30] J. Zhou and D. Gollmann. Towards verification of non-repudiation protocols. In *Proceedings of 1998 International Refinement Workshop and Formal Methods Pacific*, pages 370–380, 1998.

[31] Jianying Zhou. Achieving fair non-repudiation in electronic transactions. *Journal of Organizational Computing and Electronic Commerce*, 11(4):253–267, December 2001.

[32] Jianying Zhou, Robert Deng, and Feng Bao. Evolution of fair non-repudiation with TTP. In *Information Security and Privacy – ACISP '99*, volume 1587 of *Lecture Notes in Computer Science*, pages 258–269, Wollongong, Australia, 7–9 April 1999. Springer-Verlag.

[33] Jianying Zhou, Robert Deng, and Feng Bao. Some remarks on a fair exchange protocol. In *Public Key Cryptography – PKC 2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 46–57, Melbourne, Australia, January 2000. Springer-Verlag.

[34] Jianying Zhou and Dieter Gollmann. A fair non-repudiation protocol. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 55–61, Oakland, CA, May 1996. IEEE Computer Society Press.