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

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. Attacks on these protocols are presented, that, to our knowledge, have not been published before. Finally, an improved version of the 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 [17]. 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 ingenious approach, several protocols and improvements have been proposed ([16, 5]).

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 [1, 13, 19]. Recently, a new approach modelling non-repudiation protocols as games has been proposed by Kremer and Raskin [6].

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

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 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 [17] that has been analysed using three different methods [1, 13, 19] 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 5.2, 6.1 and 6.2. Furthermore, in Section 6.3 we present a straightforward improvement for all three versions of the protocol.

The next section defines fair non-repudiation. Section 3 explains the protocol proposed by Zhou and Gollmann and Section 4 explains our approach to automated protocol analysis. The remaining sections describe possible attacks on three protocol variants and suggest an improvement.

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 [17]. 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.

3 The basic version of the Zhou-Gollmann protocol

In this paper we discuss three versions of a non-repudiation protocol introduced by Zhou and Gollmann in [17, 18, 16]. 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. Variants of the protocols with offline TTP are not discussed in this paper, although these protocols show similar weaknesses.

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. $A \rightarrow B \quad : f_{EOO}, B, L, C, EOO\_C$
2. $B \rightarrow A \quad : f_{EOR}, A, L, C, EOR\_C$
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 5.2 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 Automated security analysis using APA and 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 [9]. It "naturally" emerges from formal language theory [8]. APA are supported by the SH-verification tool (SHVT) that provides components for the complete cycle from formal specification to exhaustive analysis and verification [9].

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 are an 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.

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 $\mathrm{Symkeys_P}$, $\mathrm{Asymkeys_P}$, $\mathrm{State_P}$, and $\mathrm{Goals_P}$ to store the symmetric and

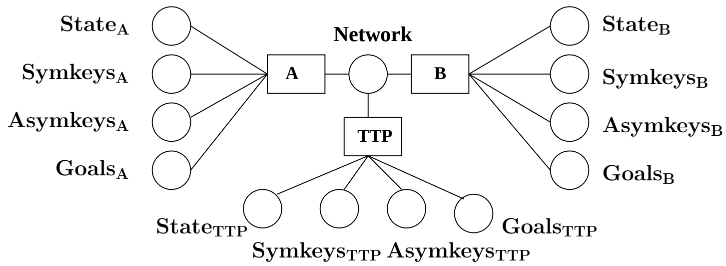

Fig. 1. Structure of the APA model for agents A, B and TTP

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 very basic data types, and the model of cryptography to those algorithms needed to specify the Zhou-Gollmann protocols. 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. $inverse((w, sym, n)) = (w, sym, n)$, $inverse((P, pub, n)) = (P, priv, n)$
3. $verify((P, pub, n), m, sign((P, priv, n), m)) = true$

(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 [4].

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.

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

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 [3].

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 [4].

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 $\mathrm{Goals}_{\mathrm{P}}$. Note that the content of $\mathrm{Goals}_{\mathrm{P}}$ has no influence on 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 $\mathrm{State}_{\mathrm{B}}$, then *EOR_C* for m signed by B is element of $\mathrm{State}_{\mathrm{A}}$ and either a matching *con_K* is in $\mathrm{State}_{\mathrm{A}}$, 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 $\mathrm{State}_{\mathrm{A}}$, then *EOO_C* for m signed by A is element of $\mathrm{State}_{\mathrm{B}}$ and either a matching *con_K* is in $\mathrm{State}_{\mathrm{B}}$, 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 $\mathrm{Goals}_{\mathrm{A}}$ and $\mathrm{Goals}_{\mathrm{B}}$, respectively.

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

State_{A}	$:=$	$\{(\mathbf{B}, \textit{agent}), (\mathbf{TTP}, \textit{server}), (\textit{start}, \mathbf{B}), (m1, m2, \textit{message})\}$
$\text{Asymkeys}_{\text{A}}$	$:=$	$\{(\mathbf{A}, \textit{priv}, (\mathbf{A}, \textit{priv}, 1)), (\mathbf{B}, \textit{pub}, (\mathbf{B}, \textit{pub}, 1)),$ $(\mathbf{TTP}, \textit{pub}, (\mathbf{TTP}, \textit{pub}, 1))\}$
Goals_{A}	$:=$	$\{\textit{NRR}(\mathbf{B})\}$
State_{B}	$:=$	$\{(\mathbf{A}, \textit{agent}), (\mathbf{TTP}, \textit{server}), (\textit{respond}, \mathbf{A})\}$
$\text{Asymkeys}_{\text{B}}$	$:=$	$\{(\mathbf{B}, \textit{priv}, (\mathbf{B}, \textit{priv}, 1)), (\mathbf{A}, \textit{pub}, (\mathbf{A}, \textit{pub}, 1)),$ $(\mathbf{TTP}, \textit{pub}, (\mathbf{TTP}, \textit{pub}, 1))\}$
$\text{State}_{\text{TTP}}$	$:=$	$\{(\mathbf{A}, \textit{agent}), (\mathbf{B}, \textit{agent})\}$
$\text{Asymkeys}_{\text{TTP}}$	$:=$	$\{(\mathbf{TTP}, \textit{priv}, (\mathbf{TTP}, \textit{priv}, 1)), (\mathbf{A}, \textit{pub}, (\mathbf{A}, \textit{pub}, 1)),$ $(\mathbf{B}, \textit{pub}, (\mathbf{B}, \textit{pub}, 1))\}$
Goals_{B}	$:=$	$\{\textit{NRO}(\mathbf{A})\}$

Table 2. The initial state

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

5 Protocol analysis

5.1 APA framework for protocol analysis

The APA specification of a protocol maintains an abstraction level where details like random number generation, number of runs that shall be performed, the actions of agents, etc., are specified on an abstract level. This has to be transfered to an abstraction level where the SHVT can actually perform a state search. The APA framework provides various means to specify concrete analysis scenarios. This includes to specify 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 also acting as A and Bob acting as B), which of the agents may act dishonestly, and other details.

For more details on the analysis APA, see [2].

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 one type of dishonest behaviour, namely the actions of a dishonest agent acting in role A. 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 his initial state (for example the names of other protocol agents, the keys he owns, etc.) and can generate random numbers and new keys.
2. He knows all messages and parts of messages, the names of sender and the intended recipient of messages he receives or maliciously removes from Network.
3. He can generate new messages by applying concatenation or the symbolic functions *encrypt*, *sign* and *hash* to messages he knows. (Note that he can use any message in his knowledge base as a key.)
4. He knows all plaintexts he can generate by deciphering a ciphertext he knows, provided he 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.

5.2 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 [17]). 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 the server add

[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.

these messages 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 (which models the server's delete action).

3. The 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 state in which the security goal *NRO(A)* in Goals_B is not satisfied.

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 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

$$\text{Network} = \{(Bob, Alice, (f_{EOR}, Alice, L, encrypt(K, m2), sign((Bob, priv, 1), (f_{EOO}, Alice, L, encrypt(K, m2))))\}$$

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.

Although this protocol and in particular its fairness has been analysed with two different analysis methods [13, 1], this weakness was not discovered. Bella and Paulson have used the inductive analysis approach based on the theorem prover Isabelle [10] 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.

Schneider's analysis uses CSP. This approach has also been used to analyse authentication and key establishment protocols [12, 11]. 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.

The protocol analysis performed by Zhou and Gollmann themselves [19] uses a modified version of the authentication logic SVO [14]. This logic (like all other authentication logics) cannot express the property of fairness, as stated by the authors of [19], consequently their analysis does not find the protocol weakness.

6 Protocol variants

6.1 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 $L = H(m, K)$ (see [17, 16]). 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:

$$\text{Network} = \{(Alice, Bob, (f_{EOO}, Bob, H(m2, K), encrypt(K, m1), sign((Alice, priv, 1), (f_{EOO}, Bob, H(m2, K), encrypt(K, m1))))\}$$

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 $sS_{Bob}(f_{EOR}, Alice, H(m2, K1), eK(m1))$, 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 $m2$:

$$\text{Network} = \{(Alice, Bob, (f_{EOO}, Bob, H(m2, K), encrypt(K, m2), sign((Alice, priv, 1), (f_{EOO}, Bob, H(m2, K), encrypt(K, m2))))\}$$

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.

6.2 Time-stamping evidence

In [16] 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 security goal: The time stamp T_{con} is supposed to identify 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, then 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 [15] 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.

6.3 Improved fair non-repudiation protocols

Obviously, the problem with all three protocol variants above 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 are 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 Relevance of the attacks

One can easily construct scenarios in which the attacks described in this paper disappear. 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 can not 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 6.2 does not require any of the assumptions above. The attack may occur even if evidence is kept forever by the TTP.

8 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 have shown possible attacks on all three variants. The attacks illustrate the need for more detailed protocol specifications as basis for secure implementations.

The security analyses were carried out using the SH-verification tool [9]. All attacks were found in less than 5 seconds on a PentiumIII 550Mhz computer. No attack was found for the improved version of the protocol proposed in Section 6.3. Here, the tool computed the complete reachable state space for a scenario with three possible protocol runs in 38 minutes. 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, [7] or [11]). 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.

Our current work includes security analyses of more sophisticated non-repudiation protocols with resolve and abort sub-protocols. Results will be published in a forthcoming paper.

References

1. 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.
2. 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.
3. S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and provability, a formal framework. In *Infrastructure Security Conference InfraSec 2002*, volume 2437, page 227, 2002.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. L. C. Paulson. Proving Properties of Security Protocols by Induction. Technical Report 409, Computer Laboratory, University of Cambridg, 1996.
11. B. Roscoe, P. Ryan, S. Schneider, M. Goldsmith, and G. Lowe. *The modelling and Analysis of Security Protocols*. Addison Wesley, 2000.
12. S. Schneider. Verifying authentication protocols with CSP. In *IEEE Computer Security Foundations Workshop*. IEEE, 1997.
13. S. Schneider. Formal Analysis of a non-repudiation Protocol. In *IEEE Computer Security Foundations Workshop*. IEEE, 1998.
14. 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.
15. J. Zhou. *Non-repudiation in Electronic Commerce*. Computer Security Series. Artech House, 2001.
16. 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.
17. 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. IEEE Computer Society Press.

18. J. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 126–132, 1997.
19. 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.