A theorem-proving approach to verification of fair non-repudiation protocols

Kun Wei and James Heather

Department of Computing, University of Surrey, Guildford, Surrey GU2 7XH, UK
Email: {k.wei,j.heather}@surrey.ac.uk

Abstract. We use a PVS embedding of the stable failures model of CSP to verify non-repudiation protocols, allowing us to prove the correctness of properties that are difficult to analyze in full generality with a model checker. The PVS formalization comprises a semantic embedding of CSP and a collection of theorems and proof rules for reasoning about non-repudiation properties. The well-known Zhou-Gollmann protocol is analyzed within this framework.

1 Introduction

Over the past decade, formal methods have been remarkably successful in their application to the analysis of security protocols. For example, the combination of CSP and FDR [9, 7, 11] has proved to be an excellent tool for modelling and verifying safety properties such as authentication and confidentiality. However, non-repudiation properties have not yet been mastered to the same degree since they must often be expressed as liveness properties and the vast bulk of work to date has been concerned only with safety properties.

Schneider has shown in [12] how to extend the CSP approach to analyze non-repudiation protocols. His proofs of correctness, based on the traces and the stable failures models of CSP as well as on rank functions, are constructed by hand. For safety properties, one usually assumes that one honest party wishes to communicate with another honest party, and one asks whether a dishonest intruder can disrupt the communications so as to effect breach of security. When considering non-repudiation, however, we are concerned with protecting one honest party against possible cheating by his or her interlocutor. Thus a non-repudiation protocol enables parties such as a sender Alice and a responder Bob to send and receive messages, and provides them with evidence so that neither of them can deny having sent or received these messages when they later resort to a judge for resolving a dispute.

There are two basic types of non-repudiation: *Non-repudiation of Origin (NRO)* provides Bob with evidence of origin that unambiguously shows that Alice has previously sent a particular message, and *Non-repudiation of Receipt (NRR)* provides Alice with evidence of receipt that unambiguously shows that Bob has received the message. Unforgeable digital signatures are usually the mechanism by which NRO and NRR can be obtained.

However, a major problem often arises: there may come a point during the run at which either Alice or Bob reaches an advantageous position; for example, Alice may have collected all the evidence she needs before Bob has collected his, and Alice may then deliberately abandon the protocol to keep her advantageous position. Usually we will want to ensure that the protocol is *fair*.

- *Fairness* guarantees that neither Alice nor Bob can reach a point where he or she has obtained non-repudiation evidence, but where the other party is prevented from retrieving any required evidence that has not already been obtained.

Obviously fairness is the most difficult property to achieve in the design of such protocols, and several different solutions have been proposed. Two kinds of approach are discussed in [5], classified according to whether or not the protocol uses a third trusted party (TTP). The first kind of approach providing fairness in exchange protocols is based on either a *gradual exchange* [15] or *probabilistic protocol* [8]. Without the involvement of a TTP, a sender Alice gradually releases messages to a responder Bob over many rounds of a protocol, with the number of rounds chosen by Alice and unknown to Bob. Bob is supposed to respond for every message, and any failure to respond may cause Alice to stop the protocol. However, such protocols require that all parties have the same computational power, and a large number of messages must be exchanged. The other kind of approach uses a TTP to handle some of the evidence. Many fair non-repudiation protocols use the TTP as a delivery authority to establish and transmit some key evidence. The efficiency of such protocols depends on how much a TTP is involved in the communication, since heavy involvement of the TTP may become a bottleneck of communication and computation.

In the CSP model, fairness is naturally described as a liveness property. It is impossible for fairness to guarantee that both Alice and Bob can collect the required evidence simultaneously, since we are dealing with an asynchronous network, but it does guarantee that either of them must be able to access the evidence as long as the other party has obtained it. In this paper, we will verify the correctness of fairness of the Zhou-Gollmann protocol [18] using the process algebra CSP and its semantics embedding in a PVS theorem prover.

We have shown how to go some way towards verifying fairness of the ZG protocol using CSP and the FDR model checker in a recent paper [17]. However, in order to keep the problem tractable, it was necessary there to model a system with very small numbers of parties and messages; it was not possible to use the model-checking approach to prove fairness of the protocol in its full generality, or even for a system of a moderate size.

Evans and Schneider [3, 12] give a useful start on this issue by using rank functions and the PVS embedding of the traces model of CSP to verify safety properties such as *NRO* and *NRR*. Our model, to some degree extending their work, can verify not only safety specifications, but also liveness specifications by means of embedding the stable failures model of CSP into the PVS theorem prover; in consequence, we can prove fairness, which cannot be tackled within their model.

In the remainder of this paper, we give a brief introduction to the CSP notation and to the embedding of CSP in PVS. We present the ZG protocol and its analysis, and then show the formalization of the protocol modelling and of its verification in PVS. Finally, we discuss and compare with other verification approaches for non-repudiation protocols.

2 CSP notation

In CSP, a system can be considered as a process that might be hierarchically composed of many smaller processes. An individual process can be combined with events or other processes by operators such as prefixing, choice, parallel composition, and so on. There are four semantic models available—traces, stable failures, failures/divergences, and failures/divergences/infinite traces—and which one is chosen depends on what properties of the system one is trying to analyze. For safety properties, the traces model of CSP is enough. In this paper, we use the stable failures model of CSP to verify fairness in the ZG protocol. We will briefly illustrate the CSP language and the semantic models; for a fuller introduction, the reader is referred to [10, 13].

Stop is a stable deadlocked process that never performs any events. The process $c \rightarrow P$ behaves like P after performing the event c. An event like c may be compounded; for example, one often used patten of events is $c.i.j.m$ consisting of a channel c, a sender i, a receiver j and a message m. The output $c!x \rightarrow P$ describes a process which is initially willing to output x along channel c, and then behave as P. The input $c?x : T \rightarrow P(x)$ denotes that it is initially ready to accept any value x of type T along channel c, and subsequently behave as $P(x)$.

The external choice $P_1 \,\Box\, P_2$ may behave either like P_1 or like P_2, depending on what events its environment initially offers it. The traces of internal choice $P_1 \sqcap P_2$ are the same as those of $P_1 \,\Box\, P_2$, but the choice in this case is non-deterministic.

The interface parallel $P_1 \underset{A}{\parallel} P_2$ is the process where all events in the set A must be synchronized, and other events can be performed independently by P_1 and P_2 respectively. An interleaving $P_1 \,|||\, P_2$ executes each part entirely independently and is equivalent with $P_1 \underset{\emptyset}{\parallel} P_2$.

A trace is defined to be a finite sequence of events. A refusal set is a set of events from which a process can fail to accept anything no matter how long it is offered; *refusals*(P/t) is the set of P's refusals after the trace t; then (t, X) is a failure in which X denotes *refusals*(P/t). If the trace t can make no internal progress, this failure is called a *stable failure*.

Liveness is concerned with behaviour that a process is guaranteed to make available, and can be inferred from stable failures; for example, if, for a fixed trace t, we have $a \notin X$ for all stable failures of P of the form (t, X), then a must be available after P has performed t.

Verification of property specifications is done by means of determining whether one process satisfies a specification. In the stable failures model, this equates to

checking whether the traces and failures of one process are subsets of the traces and failures of the other:

$$P \textbf{ sat } S \Leftrightarrow \forall\, tr \in \mathit{traces}(P) \bullet S(tr) \wedge \forall (tr, X) \in \mathit{failures}(P) \bullet S(tr, X)$$

where $\mathit{traces}(P)$ collects all traces of the process P and $\mathit{failures}(P)$ denotes the set of stable failures of P.

For the properties we are considering, if S satisfies the property specification we are verifying, then P also holds such a property.

3 Embedding CSP semantics in PVS

Full details of the embedding of the stable failures model of CSP in PVS for mechanizing proofs is presented in [16].

In the analysis of security protocols, we usually do not require modelling of successful termination; therefore, we here use a simplified version of the embedding of CSP in order to reduce the complexity of verification to some extent.

The stable failures model is represented by pairs (T, F) in which T is a set of traces that forms the semantics of a process in the traces model. The classic formalization of traces is to simply consider traces as lists of events. PVS has provided a predefined abstract datatype `list`. Thus, the type `trace` is defined as follows:

```
trace: TYPE = list[E]
```

where `E` is a parameter to denote the events appearing in the lists.

Processes in the stable failures model consist of pairs (T, F) that satisfy various conditions, which can be found in [16], or in [10, 13]. Some of CSP's main operators are listed in Table 1 in Appendix, with the standard CSP syntax and PVS's syntax. We also use the relation '`<=`' to denote satisfaction; for example, P **sat** S is represented as `P<=S` in PVS.

Recursive processes in CSP are defined in terms of equational definitions. We here formalize such processes by using the 'μ-calculus' theory to compute the least fixed point of a monotonic function[1]. We also have proved a general fixed point induction theorem to verify recursive processes. In order for fixed points to be useful, we have extended the least fixed point theory to represent mutually recursive processes and to prove whether a function has a unique fixed point.

In addition, we have proved a number of algebraic laws which are essential in the verification of properties of processes; these laws can also help us to verify the consistency of the CSP semantics. For more detailed explanations of the embedding of CSP in PVS, readers are advised to consult [16].

[1] A monotonic function in this context is a function F such that if $Q \sqsubseteq P$ then $F(Q) \sqsubseteq F(P)$.

4 The Zhou-Gollmann protocol

Zhou and Gollmann present a basic fair non-repudiation protocol using a lightweight TTP in [18], which supports non-repudiation of origin and non-repudiation of receipt as well as fairness. The main idea of the Zhou-Gollmann protocol is that a sender Alice delivers the ciphertext and the message key to Bob separately; the ciphertext is sent from the originator Alice to the recipient Bob, and Alice then sends the message key encrypted with her secret key to the TTP. Finally Alice and Bob may get their evidence from the TTP to establish the required non-repudiation. The notation below is used in the protocol description.

- M: message to be sent from A to B.
- K: symmetric key chosen by A.
- C: commitment (ciphertext) for message M encrypted with K.
- L: a unique label used to identify a particular protocol run.
- $f_{EOO}, f_{EOR}, f_{SUB}, f_{CON}$: flags indicating the purpose of a signed message.
- s_i: an asymmetric key used to generate i's digital signature.

After cutting down the plaintext part, the simplified protocol can be described as follows:

1. $A \rightarrow B: \ s_A(f_{EOO}, B, L, C)$
2. $B \rightarrow A: \ s_B(f_{EOR}, A, L, C)$
3. $A \rightarrow TTP: \ s_A(f_{SUB}, B, L, K)$
4. $B \leftrightarrow TTP: \ s_T(f_{CON}, A, B, L, K)$
5. $A \leftrightarrow TTP: \ s_T(f_{CON}, A, B, L, K)$

We briefly examine the protocol step by step to see how it works. Firstly, Alice composes a message including a flag, a unique label L, the receiver's name B and a ciphertext $C = K(M)$; Alice then signs the message with her private key and sends it to Bob. Secondly, Bob collects the message as one piece of evidence in which the label L identifies the run of the protocol, and then Bob responds with his signed message to provide A with evidence that B really has received C in this run. After she has got a response, Alice submits the encrypted message key K to the TTP. The TTP then decrypts it to get the K, generates the associated evidence encrypted with its private key and makes it available to Alice and Bob. Finally, Alice and Bob can fetch the evidence respectively.

The guarantee of fairness of such a protocol comes from an assumption that the channels between TTP and the parties are *resilient*; that is, messages may be delayed, but will eventually arrive in a finite amount of time. However, the channels between Alice and Bob can be unreliable; that is, the medium may delay, lose or misdirect messages.

5 CSP modelling

Schneider in [12] gives an excellent overview of how to extend the CSP approach to analyze non-repudiation protocols. Because of the absence of mechanizing

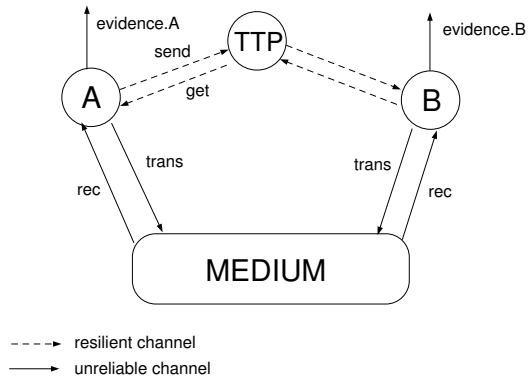

Fig. 1. Network for the ZG protocol

support, however, his proof of correctness has to be constructed by hand. Our version, whilst based on his analysis, is machine-assisted and makes various changes to the model in order to bring it closer a real-world implementation of the protocol.

Fairness says that if either A or B has managed to retrieve full evidence, the other party cannot be prevented indefinitely from retrieving the evidence. We cannot assert for verifying fairness that once A has obtained the evidence then B must have obtained the evidence as well, because there may be a delay between A's reception and B's reception. However, we can ensure that the evidence must be available to B, or that a specific action must be about to happen to enable B to get the evidence in the future.

To check a protocol like this one with CSP, we have to build models of the parties, the TTP and the medium and see how they can interfere with each other. Since the protocol is used to protect parties that do not trust each other, we do not adopt the traditional Dolev-Yao model [2], which provides a special intruder; instead, in our model, one of two communicating parties is an intruder. Fairness is only guaranteed to the party who runs in accordance with the protocol; for example, if A releases the symmetric key K before B responds, A will certainly place herself in a disadvantageous position.

It is also important, for a fully general proof, to allow other agents to participate on the network, since the protocol is expected to be correct no matter how many parties there are. We use a similar structure to that given by Schneider; the structure is shown in Figure 1.

The transmission of messages between parties is modelled by a CSP channel *trans*: the event $trans.i.j.m$ denotes that party i transmits a message m to party j. Similarly, the receipt of messages is modelled as a channel *rec*: the event $rec.i.j.m$ means that i receives a message m from j. The medium plays a role of the unreliable channel, whereas the the resilient channel is removed out of the medium and is used to directly link parties and the TTP. The communication in

the resilient channel is modelled as *send* and *get*. In addition, the parties have an *evidence* channel which they use to announce the evidence.

The different channels (one unreliable, one resilient) effectively partition the message space. Messages with the flags f_{EOO} and f_{EOR} can appear only on the unreliable channel; the messages with f_{SUB} and f_{CON} can occur only in the *send* and *get* channels. This enables us to determine which kind of channel carries a particular message when a party is announcing the evidence on the *evidence* channel.

The entire network is the parallel combination of these components:

$$NETWORK = ((Party_A \;|||\; Party_B) \underset{\{send,get\}}{\parallel} TTP) \underset{\{trans,rec\}}{\parallel} Medium$$

In our scenario, we will treat A as a dishonest party, or a spy, and B as an honest party who always performs in accordance with the protocol; A and B may behave either as a sender or as a responder. A and B may run the protocol many times, and A may make use of the information deduced from B's messages to initiate a new run. We now consider how to model the behaviour of the various components.

5.1 Defining honest parties

The basic assumption underlying the definition is that the party B is not able to release his private key, reuse a label, or lose evidence he has already got. In our model, the party B can act either as a sender or as a responder. When acting as a sender, the party B running the protocol will then be described as follows:

$$\begin{aligned} SEND = \; & trans.B.A.(s_B(f_{EOO}.A.L.C)) \rightarrow rec.B.A.(s_A(f_{EOR}.B.L.C)) \\ & \rightarrow send.B.TTP.(s_B(f_{SUB}.A.L.K)) \\ & \rightarrow get.B.TTP.(s_T(f_{CON}.A.B.L.K)) \rightarrow Party_B \end{aligned}$$

The responder process performs the protocol from the opposite perspective. B acting as a responder is described as follows:

$$\begin{aligned} RESP = \; & rec.B.A.(s_A(f_{EOO}.B.L.C)) \rightarrow trans.B.A.(s_B(f_{EOR}.A.L.C)) \\ & \rightarrow get.B.TTP.(s_T(f_{CON}.A.B.L.K)) \rightarrow Party_B \end{aligned}$$

In order to allow a party to declare the evidence as soon as he has obtained it, we provide the *SHOW* process described as follows:

$$\begin{aligned} SHOW_B(S) = \; & rec.B.A?m \rightarrow SHOW_B(S \cup \{m\}) \\ & \Box \; get.B.TTP?m \rightarrow SHOW_B(S \cup \{m\}) \\ & \Box \; evidence.B!m : S \rightarrow SHOW_B(S) \end{aligned}$$

Finally, the whole process representing the honest party B is

$$Party_B = (SEND \;\Box\; RESP) \underset{\{rec,get\}}{\parallel} SHOW_B(\emptyset)$$

5.2 Creating a spy

In our modelling of the non-repudiation protocol, we do not define a special party, a spy, as different from the legitimate parties. We assume that one of two communicating parties is a spy who may be able to deduce something of value from the messages it has received. The non-repudiation protocol is supposed to provide fairness for an honest party even if the other party is a spy.

The behaviour of a spy on the network is therefore described by the CSP process $Party_A$:

$$\begin{aligned} Party_A(S) = \; & trans.A.B!m : S \rightarrow Party_A(S) \\ & \Box\ rec.A.B?m \rightarrow Party_A(Close(S \cup \{m\})) \\ & \Box\ send.A.TTP!m : S \rightarrow Party_A(S) \\ & \Box\ get.A.TTP?m \rightarrow Party_A(Close(S \cup \{m\})) \\ & \Box\ evidence.A!m : S \rightarrow Party_A(S) \end{aligned}$$

The spy is able to transmit anything over the network that can be deduced from the messages she has already learnt. She is also able to receive anything transmitted over the network.

The spy has an initial basic knowledge, such as public keys, labels and so on, and can build a number of legitimate messages before the start of the protocol. The $Close(S)$ function returns the set S closed up under these deduction rules. We here allow the spy to have three types of deduction based on constructing and extracting sequences, symmetric-key encryption and public-key encryption. For example, if she knows $\{K(M), K\}$, the spy can deduce M. However, the spy is supposed not to know other parties' private keys since they will never transmit these keys on the network.

5.3 Medium and TTP

The medium provides two types of message delivery service: one is an unreliable channel where messages might be lost, delayed and sent to any address; another one is a resilient channel where messages might be delayed, but will eventually arrive, and also be guaranteed not to arrive at the wrong address. The medium here is defined only for the unreliable channel, since the resilient channel will be integrated into the definition of the TTP.

$$\begin{aligned} Medium(S) = \; & (\ trans?i?j?m \rightarrow Medium(S \cup \{m\}) \\ & \quad \Box\ rec?i?j!m : S \rightarrow Medium(S \backslash \{m\})\ ) \sqcap idle \rightarrow Medium(S) \end{aligned}$$

Note that the medium can deliver a message to the wrong destination, which means that it may lose messages in some sense. The *idle* channel may cause messages to be delayed at random.

The trusted third party is expected to act in accordance with its role in the protocol; that is, the TTP accepts signed messages, generates new evidence and

makes them available to associated parties. It is therefore modelled as follows:

$$\begin{aligned} TTP(S) = \ & (\ send?i.TTP?m \rightarrow TTP(S \cup Gen(\{(i,m)\}))) \\ & \Box\ get!m : S \rightarrow TTP(S)\ ) \sqcap idle \rightarrow TTP(S) \end{aligned}$$

The TTP also plays the role of the resilient channel, along with the *idle* channel that causes delays to message delivery. The $Gen(\{(i,m)\}))$ generates two copies of the evidence, for example, in this case they are $A.TTP.s_T(f_{CON}.A.B.L.K)$ and $B.TTP.s_T(f_{CON}.A.B.L.K)$, so that only involved parties are able to have access to the evidence. It is important to note the underlying assumption hidden in this definition: the TTP always stores evidence it has generated and never discards it.

5.4 Specification

We here concentrate on fairness of the ZG protocol since Evans and Schneider [3, 12] have provided rigorous verification of *Non-repudiation of Origin* and *Non-repudiation of receipt* in PVS by using the embedding of the traces model of CSP and rank functions.

Fairness is naturally expressed in the stable failures model of CSP. The essence of the idea is that if one of the two parties has obtained full evidence, then the other party either is already in possession of it or is able to access it. Since fairness is guaranteed only to a party who performs completely in accordance with the protocol, we here give only two specifications according to the different roles of B.

First, we deal with the case where B acts as a responder; that is, if A has proof of receipt, then B must be in a position to obtain proof of origin. Thus the formal specification is given as follows:

$$\begin{aligned} FAIR1(tr, X) \mathrel{\widehat{=}} & \ evidence.A.s_B(f_{EOR}.A.L.C) \in tr \\ & \wedge\ evidence.A.s_T(f_{CON}.A.B.L.K) \in tr \\ \Rightarrow & \ evidence.B.s_A(f_{EOO}.B.L.C) \notin X \\ & \wedge\ (get.B.TTP.s_T(f_{CON}.A.B.L.K) \notin X \\ & \quad \vee\ evidence.B.s_T(f_{CON}.A.B.L.K) \notin X) \end{aligned}$$

and the requirement on the system is that

$$NETWORK \ \ \textbf{sat} \ \ FAIR1(tr, X)$$

The above specification states that if A holds full evidence, then B must either be able to get the evidence or have already obtained the evidence.

Secondly, we deal with the case in which B acts as a sender; that is, if A has proof of origin, then B must be in a position to obtain proof of receipt. It is

therefore modelled as follows:

$$\begin{aligned} FAIR2(tr, X) \mathrel{\widehat{=}} & \ evidence.A.s_B(f_{EOO}.A.L.C) \in tr \\ & \wedge \ evidence.A.s_T(f_{CON}.A.B.L.K) \in tr \\ \Rightarrow & \ evidence.B.s_A(f_{EOR}.B.L.C) \notin X \\ & \wedge \ (get.B.TTP.s_T(f_{CON}.A.B.L.K) \notin X \\ & \quad \vee \ evidence.B.s_T(f_{CON}.A.B.L.K) \notin X) \end{aligned}$$

and the specification is:

$$NETWORK \ \ \textbf{sat} \ \ FAIR2(tr, X)$$

We now need to verify the two assertions above by translating the specifications into PVS notation.

6 The fairness model in PVS

The fairness property requires that if A has obtained full evidence, then B either is already in possession of it or is able to access it; for example, in the $FAIR1$ specification, if A has got $s_B(f_{EOR}.A.L.C)$ and $s_T(f_{CON}.A.B.L.K)$, then either both messages $s_A(f_{EOO}.B.L.C)$ and $s_T(f_{CON}.A.B.L.K)$ must have been appeared in the trace of B or the event $get.B.TTP.s_T(f_{CON}.A.B.L.K)$ cannot be prevented from appearing in the trace of B.

In order to establish such properties, it is useful to construct some general properties. We here use the assertion that the network satisfies the $FAIR1$ specification as an example to show how to prove the fairness property in PVS.

The fairness property is concerned with the fact that certain events should occur only under particular circumstances. We here first introduce an important specification inspired by the similar one in Schneider's rank function theory as follows:

$$R \ \text{precedes} \ T \mathrel{\widehat{=}} sigma(tr \upharpoonright T) = T \Rightarrow sigma(tr \upharpoonright R) = R$$

which states that if all events from the set T occur in a trace, then the events from R must appear earlier in the trace. The definition `pcd(R,T)` is given in Figure 2 in Appendix where `proj` corresponding to $\upharpoonright$ is the projection operation and `sigma(t)` collects all events of a trace t. A number of proof rules shown in Figure 2 can be used to make various CSP processes meet the above specification. In addition, we also define a specification `no(R)` to represent that any event of the set R does not appear in the traces.

The benefit of defining such a specification is that it can assist us to prove a group of lemmas which establish a chain of 'precedes'.

Lemma 1.

$$\begin{aligned} NETWORK \ \ \textbf{\textit{sat}} \ & evidence.A.s_B(f_{EOR}.A.L.C) \in tr \\ & \Rightarrow rec.A.B.s_B(f_{EOR}.A.L.C) \in tr \end{aligned}$$

Proof. In combination with the rules listed in Figure 2 in Appendix, it is straightforward to prove $rec.A.B.EOR$ precedes $evidence.A.EOR$ in the $Party_A$ process, which is formalized as a recursive process in PVS. The $NETWORK$ process then holds the property since the interleaving of $Party_A$ and $Party_B$ meets it according to the rule `pcd_interleave`, then the parallel combination of such an interleaving with TTP and $Medium$ also meets the lemma in terms of the rule `pcd_parallel1`.

Lemma 2.

$$NETWORK \ \textbf{\textit{sat}} \ rec.A.B.s_B(f_{EOR}.A.L.C) \in tr \Rightarrow trans.B.A.s_B(f_{EOR}.A.L.C) \in tr$$

Proof. In line with the rule `pcd_parallel`, the $NETWORK$ inherits this property from the $Medium$ process where $trans.B.A.EOR$ always precedes $rec.A.B.EOR$ which is included in the interface of the $Medium$ with other components .

Lemma 3.

$$NETWORK \ \textbf{\textit{sat}} \ trans.B.A.s_B(f_{EOR}.A.L.C) \in tr \Rightarrow rec.B.A.s_A(f_{EOO}.B.L.C) \in tr$$

Proof. In $Party_B$, $rec.B.A.EOO$ obviously occurs earlier than $trans.B.A.EOR$ since B performs completely in accordance with the protocol, so $NETWROK$ satisfies the property too no matter how other components behave.

Lemma 4.

$$NETWORK \ \textbf{\textit{sat}} \ rec.B.A.s_A(f_{EOO}.B.L.C) \in tr \Rightarrow evidence.B.s_A(f_{EOO}.B.L.C) \notin X$$

Proof. This lemma does not use 'precedes'; however, it can be easily proved in PVS using the general fixed point induction theorem. According to the definition of the $Party_B$, he would like to announce the evidence as long as he gets it.

Therefore, Lemmas 1–4 together with the rule `pcd_transitive` establish the following lemma, which is the first part of the $FAIR1$ specification.

Lemma 5.

$$NETWORK \ \textbf{\textit{sat}} \ evidence.A.s_B(f_{EOR}.A.L.C) \in tr \Rightarrow evidence.B.s_A(f_{EOO}.B.L.C) \notin X$$

To finish the proof of the fairness property, we then introduce the second specification as follows:

$$T \text{ is unpreventable after } R \mathrel{\widehat{=}} sigma(tr \restriction R) = R \Rightarrow sigma(tr \restriction T) = T \vee X \cap T = \emptyset$$

which states that if all events from the set R occur in a trace, then all events from T either have appeared earlier in the trace or are not included in the refusal set X after this trace; the function *sigma* converts a trace into a set. Such a specification corresponding to the second part of the $FAIR1$ specification is here used to prove that $evidence.B.CON$ is unpreventable after A has got full evidence. The definition of `upt(T,R)` and some important rules are given in Figure 3 in the appendix.

Since both parties always get the evidence before they announce it on the *evidence* channel, the lemma we really want to reach here is described as follows:

Lemma 6.

$$\begin{aligned} NETWORK \ \textbf{\textit{sat}} \ & rec.A.B.s_B(f_{EOR}.A.L.C) \in tr \\ & \wedge \ get.A.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ \Rightarrow \ & get.B.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ & \vee \ get.B.TTP.s_T(f_{CON}.A.B.L.K) \notin X \end{aligned}$$

On face of it, the assertion is too tricky to be reached simply by applying the rules listed in Figure 3. The solution is to rewrite the $NETWORK$ process so that it matches with the rule `upt_parallel`.

$$NETWORK = (Party_A \ ||| \ Party_B) \underset{\{trans,rec,send,get\}}{\parallel} (Medium \ ||| \ TTP)$$

Obviously, this new definition is equivalent to the original one, but makes the proof become rather easier when combined with the following lemmas.

Lemma 7.

$$\begin{aligned} Medium \ ||| \ TTP \ \textbf{\textit{sat}} \ & get.A.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ \Rightarrow \ & get.B.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ & \vee \ get.B.TTP.s_T(f_{CON}.A.B.L.K) \notin X \end{aligned}$$

Proof. Once the TTP has received a legitimate submission, the evidence is always available to the parties involved. Hence, $get.B.TTP.CON$ is always unpreventable after $get.A.TTP.CON$ has occurred.

Lemma 8.

$$\begin{aligned} Party_A \ ||| \ Party_B \ \textbf{\textit{sat}} \ & trans.B.A.s_B(f_{EOO}.A.L.C) \in tr \\ \Rightarrow \ & get.B.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ & \vee \ get.B.TTP.s_T(f_{CON}.A.B.L.K) \notin X \end{aligned}$$

Proof. The $Party_B$ has to perform $get.B.TTP.CON$ after he responds to A with $s_B(f_{EOO}.A.L.C)$, so that $get.B.TTP.CON$ is unpreventable.

Lemma 9.

$$\begin{aligned} NETWORK \ \textbf{\textit{sat}} \ & trans.B.A.s_B(f_{EOR}.A.L.C) \in tr \\ & \wedge \ get.A.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ \Rightarrow \ & get.B.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ & \vee \ get.B.TTP.s_T(f_{CON}.A.B.L.K) \notin X \end{aligned}$$

Proof. This is proved using the rule `upt_parallel` with Lemma 7 and Lemma 8.

Also, from the proof of the first part of the *FAIR*1 specification, we may prove the fact that $trans.B.A.EOR$ precedes $rec.A.B.EOR$, so that Lemma 6 can be proved using the rule `upt_pcd_transitive` with Lemma 9. Using this rule again, we can prove the second part of the *FAIR*1 specification that is described as follows:

Lemma 10.

$$\begin{aligned} NETWORK \ \textbf{\textit{sat}} \ & evidence.A.s_B(f_{EOR}.A.L.C) \in tr \\ & \wedge \ evidence.A.s_T(f_{CON}.A.B.L.K) \in tr \\ \Rightarrow \ & get.B.TTP.s_T(f_{CON}.A.B.L.K) \in tr \\ & \vee \ get.B.TTP.s_T(f_{CON}.A.B.L.K) \notin X \end{aligned}$$

This completes the proof; we have formalized all this in PVS. The proof of the *FAIR*2 specification has also been formally completed in a similar way, making use of the above lemmas.

7 Discussion

In this paper, we have modelled and verified the Zhou-Gollmann non-repudiation protocol with respect to correctness of fairness, which requires that neither of two parties can establish evidence of origin or evidence of receipt while still preventing the other party from obtaining such evidence. Proving fairness in a theorem prover can help us to extract some general understanding of how to design such a kind of protocol. For instance, from the Lemma 7 and Lemma 8, we can clearly see that least two factors should be considered: one is that the TTP should always make the evidence available to the parties involved when it has received a legitimate submission; the other is that B cannot be prevented from accessing the evidence when he has responded to the first part of evidence from A.

Although the Zhou-Gollmann protocol is rather simple, our formal verification shows that it does provide strong fairness under the assumptions described in this paper. However, there is an attack [4] if we slightly change our assumptions. Suppose that we allow the TTP and B to lose the evidence, and A first completes a protocol run with B and possesses $s_B(f_{EOR}.A.L.C)$ and $s_T(f_{CON}.A.B.L.K)$; a couple of weeks later, A then uses the same symmetric

key and label but sends a new plaintext message to initiate a new run; A ends the run by obtaining the new EOR and presenting it with $s_T(f_{CON}.A.B.L.K)$ to a judge; A might be lucky enough to discover that TTP and B have discarded their old evidence.

Incorporating this assumption into our model would mean that the correctness of the fairness property cannot be proved because of the presence of the hidden attacks. For example, Lemma 4 would not hold any more if B could lose the evidence. In investigating why these lemmas can no longer be proven, it is likely that one would uncover the attack.

We here give a suggestion that all evidence should have incorporated into it a tag provided by the TTP, such as timestamp, so as to make it clear when the two parts of the evidence match each other.

Some related work can be found in the literature concerning verification of non-repudiation protocols using different approaches. Zhou et al. in [19] firstly use 'BAN-like' belief logic to check only safety properties of the non-repudiation protocols. Schneider [12] gives an excellent overview of the CSP modelling and proves the correctness of properties using stable failures and rank functions; however, the proofs are constructed by hand. Evans [3] extends Schneider's work to prove safety properties such as NRO and NRR in the PVS theorem prover. Shmatikov and Mitchell in [14] verify fairness as a monotonic property using Murφ; that is, if fairness is broken at one point of the protocol, the protocol will remain unfair. This approach also cannot deal with liveness properties. Kremer and Raskin [6] use the finite state model checker MOCHA to verify non-repudiation and fair exchange protocols. This approach, which is rather different from ours here, can also cope with liveness properties as well as safety properties. However, they have modelled networks in which A and B can engage in only one run of the protocol. In addition, Abadi and Blanchet [1] formalize and verify the key security properties of a cryptographic protocol for certified email that has many commonalities with the Zhou-Gollmann protocol. Most of verification work is done with an automatic protocol verifier, however such a tool has not been verified and finished.

We have proved the fairness property of the Zhou-Gollmann protocol in its full generality in the case of two parties that are able to perform multiple runs and act in different roles, along with an unbounded number of atomic messages. Admittedly, verifying a system like this requires considerable work. However, PVS is a deductive system in which all completed proofs can be used in later proofs. In the course of constructing this proof, we have amassed many lemmas and theorems that will make proving properties of similar systems substantially less time-consuming, both for us and for others.

We aim to extend our model to deal with protocols that involve more than two parties. We also wish to cover timeliness; that is, we wish to verify that all honest parties can reach a point where they can stop the protocol while preserving fairness. We will develop our current model to cover this issue in future work.

References

1. M. Abadi and B. Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. In R. Cousot, editor, *Static Analysis, 10th International Symposium (SAS'03)*, volume 2694 of *Lecture Notes on Computer Science*, pages 316–335, San Diego, California, June 2003. Springer Verlag.
2. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.
3. N. Evans. *Investigating Security through Proof.* PhD thesis, Royal Holloway, University of London, 2003.
4. S. Gürgens and C. Rudolph. Security analysis of (un-) fair non-repudiation protocols. In A. E. Abdallah, P. Ryan, and S. Schneider, editors, *FASec*, volume 2629 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2002.
5. S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of non-repudiation protocols. Technical Report 473, 2002.
6. S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Lecture Notes in Computer Science*, 2154, 2001.
7. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
8. O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *Second Workshop on Security in Communication Network 99*, 1999.
9. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. *Proceedings of 8th IEEE Computer Security Foundations Workshop*, 1995.
10. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall International, 1998.
11. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling And Analysis Of Security Protocols.* Addison Wesley, July 2000.
12. S. A. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, 1998.
13. S. A. Schneider. *Concurrent and real-time systems: the CSP approach.* John Wiley & Sons, 1999.
14. V. Shmatikov and J. C. Mitchell. Analysis of abuse-free contract signing. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, pages 174–191, London, UK, 2001. Springer-Verlag.
15. T. Tedrick. How to exchange half a bit. In *CRYPTO*, pages 147–151, 1983.
16. K. Wei and J. Heather. Embedding the stable failures model of CSP in PVS. In J. Romijn, G. Smith, and J. van de Pol, editors, *IFM*, volume 3771 of *Lecture Notes in Computer Science*, pages 246–265. Springer, 2005.
17. K. Wei and J. Heather. Towards verification of timed non-repudiation protocols. In R. Gorrieri, F. Martinelli, P. Ryan, and S. Schneider, editors, *Proceedings of FAST'05*, volume 3866 of *Lecture Notes in Computer Science*, pages 224–257. Springer-Verlag, 2005.
18. J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 55–61, Oakland, CA, 1996. IEEE Computer Society Press.
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, Canberra, Australia, Sept. 1998.

Appendix

Operation	**CSP**	**PVS**
Stop	*Stop*	`Stop`
Prefix	$a \rightarrow P$	`a >> P`
External choice	$P_1 \Box P_2$	`P1 \/ P2`
Internal choice	$P_1 \sqcap P_2$	`P1 /\ P2`
Interface parallel	$P_1 \underset{A}{\parallel} P_2$	`Par(A)(P1,P2)`
Interleave	$P_1 \mid\mid\mid P_2$	`P1 // P2`

Table 1. CSP syntax

```
property_pcd[E:TYPE]:THEORY

a: VAR E
t: VAR trace[E]
R,T,X,A: VAR set[E]
P,Q: VAR process[E]

pcd(R,T):[set[trace[E],set[[trace[E],set[E]]]]
   =({t|sigma(proj(t,T))=T IMPLIES sigma(proj(t,R)=R},{(t,X)|true})

no(R): [set[trace[E]],set[[trace[E],set[E]]]]
             = ({ t | proj(t,R)=null }, {(t,X) | proj(t,R)=null} )

pcd_stop: LEMMA nonempty?(T) IMPLIES Stop <= pcd(R,T)
pcd_prefix: LEMMA P <= pcd(R,T) AND NOT T(a) IMPLIES a>>P <= pcd(R,T)
pcd_extchoice: LEMMA P<=pcd(R,T) AND Q<=pcd(R,T)
                                  IMPLIES P\/Q <= pcd(R,T)
pcd_parallel: LEMMA P <= pcd(R,T) AND subset?(T,A)
                                  IMPLIES Par(A)(P,Q) <= pcd(R,T)
pcd_parallel1: LEMMA P <= pcd(R,T) AND Q <= no(T)
                                  IMPLIES Par(A)(P,Q) <= pcd(R,T)
pcd_interleave: LEMMA P <= pcd(R,T) AND Q <= no(T)
                                  IMPLIES P//Q <= pcd(R,T)
pcd_transitive: LEMMA P<=pcd(R,X) AND P<=pcd(X,T)
                                  IMPLIES P <= pcd(R,T)

END property_pcd
```

Fig. 2. Proof rules for precedes

```
property_upt[E:TYPE]:THEORY

a: VAR E
t: VAR trace[E]
R,T,X,A: VAR set[E]
P,Q: VAR process[E]

upt(T,R):[set[trace[E],set[[trace[E],set[E]]]]
         =({t|true},{(t,X)| sigma(proj(t,R))=R IMPLIES
                            (sigma(proj(t,T))=T OR disjoint?(T,X))})

upt_stop: LEMMA nonempty?(R) IMPLIES Stop <= upt(T,R)

upt_prefix: LEMMA P <= upt(T,R) AND NOT R(a) AND nonempty?(R)
                                          IMPLIES  a >> P <= upt(T,R)
upt_extchoice: LEMMA P<=upt(T,R) AND Q<=upt(T,R)
                                          IMPLIES P\/Q <= upt(T,R)
upt_parallel: LEMMA P<=upt(T,R) AND Q<=upt(T,X) AND
                            subset?(union(R,X),A) AND subset?(T,A)
                          IMPLIES Par(A)(P,Q) <= upt(T,union(R,X))

upt_interleave: LEMMA P<=upt(T,R) AND Q<=no(R) IMPLIES P//Q <= upt(T,R)

upt_pcd_transitive: LEMMA P <= upt(T,X) AND P <= pcd(X,R)
                                                  IMPLIES P <= upt(T,R)

END property_upt
```

Fig. 3. Proof rules for unpreventable