

Towards Verification of Timed Non-repudiation Protocols

Kun Wei and James Heather

Department of Computing, University of Surrey, Guildford,
Surrey GU2 7XH, UK

{k.wei, j.heather}@surrey.ac.uk

Abstract. Fairness of non-repudiation is naturally expressed as a liveness specification, as in [Sch98]; to formalize this idea, we apply the process algebra CSP to analyze the well-known Zhou-Gollmann protocol. We here model and verify a variant of the ZG protocol that includes a deadline (timestamp) for completion of the protocol, after which an agent can no longer initiate the recovery protocol with the TTP to get hold of the non-repudiation evidence. The verification itself is performed by the FDR model-checker.

1 Introduction

Security protocols are often complex because they represent concurrent systems in which various entities can run independently and simultaneously. Consequently, constructing proofs of correctness by hand can be arduous and error-prone.

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 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 shows in [Sch98] 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 [KMZ02], classified according to whether or not the protocol uses a trusted third party (TTP). The first kind of approach providing fairness in exchange protocols is based on either a *gradual exchange* [Ted83] or *probabilistic protocol* [MR99]. 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 this paper, we will verify fairness of the timed Zhou-Gollmann protocol [ZG97] with an off-line TTP—that is, a TTP that is involved in the protocol only when parties are in dispute. To model such a protocol, we build a model of all of the entities involved in the network: a spy, a TTP, an honest party and so on. The factor of time is also considered in such a protocol; for example, it is reasonable that the responder should know when the evidence is available from the TTP, so that it does not have to poll the server at regular intervals, causing unnecessary network traffic.

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.

Fairness in the Zhou-Gollman protocol relies on the assumption that the communication channels between a TTP and all parties are *resilient*. A resilient channel may delay a message for a finite, unknown amount of time, but will

eventually deliver it to its destination. Communication between parties, however, goes across *unreliable* channels that allow a message to be lost, delayed, or even delivered to the wrong destination.

The paper is organised as follows: the CSP notation is briefly introduced, and the timed Zhou-Gollmann protocol is described. We give details of the CSP modelling for every entity involved in a run, and its associated FDR encoding. Finally, we discuss the implications of the successful verification, and talk about future work.

2 CSP Notation

CSP is an event-orientated language for describing concurrent systems and their interactions. A security protocol is a concurrent system in which a series of messages are exchanged among the various parties involved. CSP is therefore well suited to the modelling and analysis of security protocols.

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. 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 [Ros98, Sch99].

Stop is a stable deadlocked process that never performs any events. The process $c \rightarrow P$ behaves like P after performing the event c . A 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 external choice $P_1 \square P_2$ 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 $P_1 \square P_2$, but the choice in this case is non-deterministic.

The process $P_1 \parallel_A \parallel_B P_2$ is the process where all events in the intersection of A and B must be synchronized, and other events within A and B can be performed independently by P_1 and P_2 respectively. An interleaving $P_1 \parallel P_2$ executes each part entirely independently and is equivalent with $P_1 \parallel_{\emptyset} P_2$.

The process $P \setminus A$ will pass through the same events as P , but events in the set A become be invisible. The renamed process $P[a \leftarrow b]$ means that the event a is completely replaced by b in the process P . In addition, processes may also be described recursively whenever such descriptions are well defined.

A trace is defined to be a sequence of finite 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 in FDR is done by means of determining whether one process refines another. 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:

$$P \sqsubseteq_F Q \equiv \text{traces}(P) \supseteq \text{traces}(Q) \wedge \text{failures}(P) \supseteq \text{failures}(Q)$$

For the properties we are considering, if P meets the properties we are verifying, then Q also meets them if Q refines P .

3 The Timed Zhou-Gollmann Protocol

Zhou and Gollmann present a basic fair non-repudiation protocol using a lightweight TTP in [ZG96], which supports non-repudiation of origin and non-repudiation of receipt as well as fairness. They then propose an improved protocol in [ZG97], with an off-line TTP that is more efficient in environments in which the two parties usually play fair in a protocol run, and want to resort to the TTP only when they are in dispute. In addition, it is possible (and, indeed, desirable) to include a timeout in the protocol, so that the responder will know at what point he will be able to recover evidence from the TTP.

The main idea of all Zhou-Gollmann protocols 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, Alice then sends the message key encrypted with her secret key to Bob or the TTP. Finally Alice and Bob may get the evidence or confirmation messages 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 defined by A .
- C : commitment (ciphertext) for message M encrypted with K .
- L : a unique label used to identify a particular protocol run.
- $f_{NRO}, f_{NRR}, f_{EEO}, f_{EOR}, f_{SUB}, f_{CON}$: flags indicating the purpose of a signed message.
- T : the deadline by which the TTP must have been asked to make the evidence available to the public.
- s_i : an asymmetric key used to generate i 's digital signature.

After cutting down the plaintext part, the simplified protocol can be divided into a main protocol and a recovery protocol. In the normal case, the sender Alice and the responder Bob will exchange messages and non-repudiation evidence directly, described as follows:

1. $A \rightarrow B : s_A(f_{NRO}, B, L, T, C)$
2. $B \rightarrow A : s_B(f_{NRR}, A, L, T, C)$

3. $A \rightarrow B : s_A(f_{EOO}, B, L, K)$
4. $B \rightarrow A : s_B(f_{EOR}, A, L, K)$

And if Alice does not get message 4 from Bob after sending message 3, she then launches the recovery protocol to get the associated evidence from the TTP.

1. $A \rightarrow TTP : s_A(f_{SUB}, B, L, T, K)$
2. $B \leftrightarrow TTP : s_T(f_{CON}, A, B, L, T, K)$
3. $A \leftrightarrow TTP : s_T(f_{CON}, A, B, L, T, K)$

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 $C = K(M)$, along with a chosen deadline T (which is to be interpreted according to the TTP's clock); 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. Bob can also refuse to respond to Alice if he is not satisfied with the deadline T .

After she has got a response, Alice directly sends the encrypted message key K to Bob, and Bob then sends the associated evidence back again. The protocol is now successfully completed if no dispute occurs; however, if Alice does not get her evidence at step 3 of the main protocol, she can launch the recovery protocol and submit a message to the TTP to retrieve the evidence. The TTP will check the deadline T first to determine whether or not to accept the request. If the request comes in before the deadline, the TTP will generate the evidence and make it available to Alice and Bob. The advantage of this deadline is that if Bob does not receive message 3 from Alice, he does not have to poll the TTP indefinitely to see if Alice has initiated the recovery protocol and thus made the key and the evidence available to him. He can simply wait until time T and then poll the TTP. If Alice has already initiated the recovery protocol then he will be able to get the key K and the non-repudiation evidence; if she has not done so then he will not be able to get the key or the evidence, but he will know that Alice cannot get the non-repudiation evidence either, since the deadline has now passed.

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 be 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.

Although Bob in the execution of the protocol can be temporarily in an advantageous position, Alice and Bob should be in a fair position at the end of the protocol. The introduction of the deadline T does in principle compromise the fairness of the protocol; for instance, Alice may not get the evidence from Bob at step 4 in the main protocol, but the submission of Alice's request to initiate the recovery protocol may be so severely delayed that the deadline has passed by the time it arrives and the TTP refuses to respond to it. Alice will in this

instance not get all the required evidence, even though Bob has obtained his. As suggested in [ZG97], Alice has to choose T to be large enough that this issue will not arise in practice.

4 CSP Modelling

Fairness says that if either A or B has got full evidence, the other party cannot be prevented from retrieving the evidence indefinitely. 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 need to model a special intruder party. However, 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.

In our model, we directly formalize the outcome of the TTP's test for whether the deadline has passed, without modelling specific values of T ; in other words, we model the deadline T as a boolean variable. When the TTP judges whether T has expired, the outcome will be either true or false, and the TTP will accordingly either accept or refuse the request. The deadline test can be modelled within the TTP using internal choice.

4.1 Data Types

The above description of the protocol indicates that the message space contains flags, labels, various keys, names of parties, text messages, the deadline and combinations of these. Encryption, as is typical in these situations, are treated symbolically.

Like other model checkers, FDR can only verify systems with a reasonable number of states. Therefore, we assume that only two parties are communicating, and we restrict the number of possible messages of each data type.

```
datatype fact = Sq.Seq(fact) |
  SK.(fact,fact) | Encrypt.(fact,fact) |
  Alice | Bob | TTP |
  pkA | pkB | pkT | skA | skB | skT |
  fNRO | fNRR | fEOO | fEOR | fSUB | fCON |
  La | Lb | Ka | Kb | T | AtoB | BtoA
```

where the type `fact` is a collection of all constants, and it can be used to represent any message appearing in the protocol.

We also define some sets, functions and definitions to represent legitimate messages, symbolic encryption and mapping of labels, keys and messages with the identities of parties.

We assume that no party is able to forge other parties' digital signatures; that is, parties never release their private keys. In our scenario, we will treat A as a dishonest party, or a spy, and B is 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.

4.2 Defining Honest Parties

We now represent the behaviour of an honest party in the timed ZG protocol. The protocol specification assumes that the channel between parties is unreliable, whereas the channel between the TTP and parties is resilient. We define, as follows, the transmission of messages using CSP channels.

```
channel trans,rec:agents.agents.Umessages
channel send,get:allagents.allagents.Rmessages
channel evidence:agents.messages
```

where `trans` and `rec` are for unreliable channels, `send` and `get` are for resilient channels; the channel `evidence` represents announcement of parties' obtained evidence; `Umessages` and `Rmessages` include messages in unreliable channels and resilient channels respectively.

A party can act either as a sender or as a responder; once its labels have run out, it acts only as a responder.

```
User(id,ls) =  ls! =<> & Send(id,ls)
              [] Resp(id,ls)
```

When acting as a sender, the party chooses the facts from its own knowledge to construct and transmit the messages in turn. In order to keep the size of all parties' message spaces fairly small, the parties A and B have only one value for labels, message keys and plaintext, but A may get some of information from B such as the message key K_b during the execution of the protocol and use it in later runs.

We integrate all behaviour of a party in the main protocol and the recovery protocol into one process. After A sends the message to B at step 3 in the main protocol, she may wait for a response from B and finish the protocol, or initiate the recovery protocol to retrieve the evidence from the TTP.

```
Send(id,ls) = |~|a:diff(agents,{id})@ (|~|l:label(id)@
                                         (|~|k:symkeys(id)@ (|~|m:text(id)@
trans.id.a.ske(sk(id),Sq.<fNRO,a,l,T,encrypt(k,m)>) ->
rec.id.a.ske(sk(a),Sq.<fNRR,id,l,T,encrypt(k,m)>) ->
trans.id.a.ske(sk(id),Sq.<fE00,a,l,k>) ->
```

```

((rec.id.a.ske(sk(a),Sq.<fEOR,id,l,k>) -> User(id,tail(ls)))
 []
 (send.id.TTP.ske(sk(id),Sq.<fSUB,a,l,T,k>) ->
 get.id.TTP.ske(skT,Sq.<fCON,id,a,l,T,k>->User(id,tail(ls))))))

```

The responder process performs the protocol from the opposite perspective. Note that we assume the responder can refuse to accept messages including its own labels, since the labels are usually generated associated with the plaintexts and the message keys; therefore, it is reasonable to suppose that the receiver is vigilant enough to spot such abuses.

```

Resp(id,ls) = []a:diff(agents,{id})@ ([l:diff(labels,label(id))
 @([k:symmetrickey@([m:plaintext@
 rec.id.a.ske(sk(a),Sq.<fNRO,id,l,T,encrypt(k,m)>)->
 trans.id.a.ske(sk(id),Sq.<fNRR,a,l,T,encrypt(k,m)>)->
 ((rec.id.a.ske(sk(a),Sq.<fE00,id,l,k>) ->
 trans.id.a.ske(sk(id),Sq.<fEOR,a,l,k>)-> User(id,ls))
 []
 (get.id.TTP.ske(skT,Sq.<fCON,a,id,l,T,k>) ->User(id,ls))))))

```

The responder does accept any commitment because it does not know what the commitment means until the end of the run. In addition, A may not send message 3 to B at all; B must thus be able to check whether the evidence is available from the TTP.

For the purpose of verification, we define a process `Show(id)` to show the evidence that a party has obtained. This process may show the evidence to the network as long as the relevant party has got the evidence. Finally, a well-behaved party is described as:

```
Party(id,ls) = User(id,ls) [|{|rec,get|}|] Show(id)
```

4.3 Creating a Spy

In the modelling of the non-repudiation protocol, we do not define a special party, a spy, as different from the legitimate parties. On the contrary, 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. Our spy model roughly corresponds to Roscoe's lazy spy model [Ros98], but slightly modified to suit our case. We here represent some key parts of the model; more details may be found in [Ros98].

A spy first has a set of deductive rules; for example, if it knows all members of a sequence, then it can build the sequence. A deduction is a pair (X, f) where X is a finite set of facts and f is an individual fact. Thus, anyone in possession of X can construct f as well. In our spy model, three types of deduction are built based on constructing and extracting sequences, symmetric-key encryption and public-key encryption.

The spy has an initial basic knowledge, such as public keys, labels and so on, and can further close up such basic facts by means of the `Close` function to construct a number of legitimate messages before the start of the protocol. The full initial knowledge of the spy is constructed by closing up the initial basic knowledge under deduction rules. In this case we chose Alice as a spy, what she initially knows may then be represented as follows:

```
IK= {Alice,Bob,TTP,pkA,pkB,pkT,skA,T,
     fNRO,fNRR,fE00,fE0R,fSUB,fCON,La,Ka,At0B}
Known = Close(IK)
```

In order to restrict the state space to a manageable size, we define a new set of deductions whose conclusion is something that the spy does not know yet, but that it will learn. In other words, the spy can never deduce anything it already knows. Additionally, to reduce the size of state space further and to ensure efficient compilation by the model checker, we define a parallel network which has one process for every fact inside the spy's `LearnableFacts`.

```
ignorantof(f) = member(f, messages)& learn.f -> knows(f)
  [] infer?t:{(X,f')|(X,f')<-Deductions,f==f'}->knows(f)

knows(f) = member(f,messages)&say.f -> knows(f)
  [] member(f,messages)&learn.f->knows(f)
  [] infer?t:{(X,f')|(X,f')<-Deductions,member(f,X)}->knows(f)
```

where `Deductions` is a collection of all possible deductive rules only for learnable facts.

Finally, the spy is then constructed by putting all these processes in parallel, hiding the inferences, and applying the *chase* operator¹.

```
Spy = chase((|f:LearnableFacts@[AlphaL(f)]ignorantof(f))
           \{|infer|}) ||| SayKnown
```

where `SayKnown` makes the spy say or learn legitimate messages in its `Known` facts.

To make the spy useful in a real network, we rename it so that it may communicate with other parties. Also, we provide the spy with the capability to show its evidence.

```
RenSpy(id) =((Spy[|say.f<-trans.a.b.f,learn.f<-rec.a.b.f|
                 a.b.f<-Ucomm,a==id]
             [|say.f<-send.a.b.f,learn.f<-get.a.b.f|
                 a.b.f<-Rcomm,a==id]))
           [|{|rec,get|}] Show(id)
```

where `Ucomm` and `Rcomm` are used to reduce unnecessary states; for example, `Rcomm` may restrict that one of agents must be the TTP and `f` must be the messages circulating in the resilient channel.

¹ The *chase* operator is designed specifically for this purpose; the reader is invited to consult [For97] for more information.

4.4 TTP and Medium

The trusted third party is supposed 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. The TTP also refuses to respond the parties whenever the deadline T has expired. The test for expiry of the deadline T is modelled by an internal choice in CSP. It is therefore modelled as follows:

```
Tnot(m)=send?a:agents!TTP!m->(Tnot(m) |~| Tknows(Gen(m)))
Tknows(S)=get?m:S->Tknows(S) []idle-> Tknows(S)
```

```
TrustTP = (|||m:mess_SUB@ Tnot(m))
```

where, obviously, the TTP will not confirm the party's submission after T in the $Tnot(m)$; if the TTP accepts it, the message will go into the process $Tknows$ where the evidence will be available to both parties. Note that we implement the possibility of delays in the resilient channels by introducing an action *idle* in the $Tknows(S)$. When the TTP receives a message, it then can hold the message in a finite amount of time, but will send it out eventually. The TTP only accepts messages with the label f_{SUB} . Also, we define a function $Gen(m)$ to transform submitted messages to confirmed messages for involved parties.

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. Since the resilient channel has been modelled in the definition of the TTP, the model of the medium here is defined only for the unreliable channel:

```
Hears(m) = member(m,Umessages)&
           trans?a?b:diff(agents,{a})!m -> Middle(m)
Middle(m) = idle -> Middle(m)
           []lost -> Hears(m)
           []rec?a?b:diff(agents,{a})!m -> Hears(m)
```

```
Medium = |||m:Umessages@Hears(m)
```

The medium is modelled exactly in terms of its description in the protocol. We define two channels *idle* and *lost* to represent messages being delayed or lost.

4.5 Specification and Verification

The two parties and the TTP transmit messages via unreliable channels and resilient channels in the medium as shown in Figure 1. It would be desirable to allow more potential protocol participants, since the protocol is expected to be

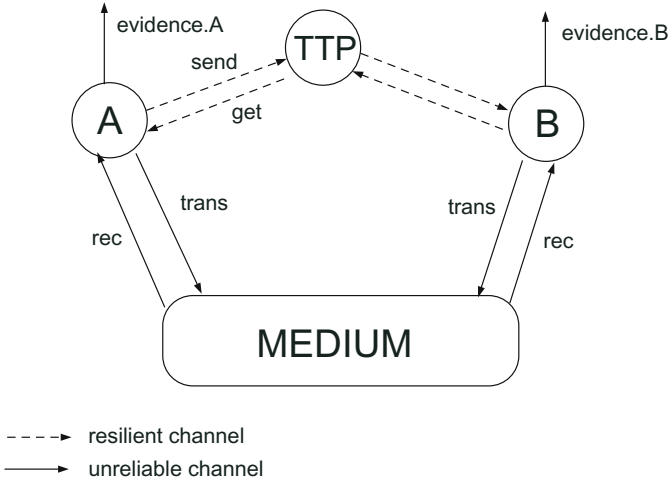


Fig. 1. Network for a non-repudiation protocol

correct even in the presence of other parties of the network. However, a bigger network would quickly give rise to state space explosion.

The entire network is the parallel combination of these components:

```
Network = ((RenSpy(Alice) ||| Party(Bob,<Lb>)
            [|{|send,get|}|] TrustTP)
            [|{|trans,rec|}|] Medium
```

We can then test for attacks on the protocol by checking whether this network satisfies a specification encapsulating the fairness property.

Fairness is naturally specified by Schneider [Sch98] in the stable failures model of CSP. The essence of his 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. We have slightly changed the above specification to meet the timed Zhou-Gollmann non-repudiation protocol, and we give here two specifications according to the different role of B.

First, we deal with the case where B acts as a responder. In the normal case, if A has got evidence of receipt then B must be in a position to obtain evidence of origin.

$$\begin{aligned}
 FAIR1(tr, X) \hat{=} & \text{evidence.A.s}_B(f_{EOR}, A, La, T, Ka) \text{ in } tr \\
 & \wedge \text{evidence.A.s}_B(f_{NRR}, A, La, C) \text{ in } tr \\
 \Rightarrow & \\
 & (\text{evidence.B.s}_A(f_{NRO}, B, La, Ka) \notin X \\
 & \wedge \text{evidence.B.s}_A(f_{EOO}, B, La, T, Ka) \notin X)
 \end{aligned}$$

When a dispute arises, the specification is defined as follows:

$$\begin{aligned}
FAIR2(tr, X) &\hat{=} \text{evidence}.A.s_T(f_{CON}, A, B, La, T, Ka) \text{ in } tr \\
&\quad \wedge \text{evidence}.A.s_B(f_{NRR}, A, La, T, C) \text{ in } tr \\
&\Rightarrow \\
&\text{get}.B.TTP.s_T(f_{CON}, A, B, La, T, Ka) \notin X \vee \\
&(\text{evidence}.B.s_A(f_{NRO}, B, La, T, Ka) \notin X \\
&\quad \wedge \text{evidence}.B.s_T(f_{CON}, A, B, La, T, Ka) \notin X)
\end{aligned}$$

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

Secondly, we deal with the case in which B acts as a sender. For this case, the specification is different from the above one, since a sender is in a weaker position in the protocol. If no dispute arises:

$$\begin{aligned}
FAIR3(tr, X) &\hat{=} \text{evidence}.A.s_B(f_{EOO}.A.Lb.Kb) \text{ in } tr \\
&\quad \wedge \text{evidence}.A.s_B(f_{NRO}.A.Lb.T.C) \text{ in } tr \\
&\Rightarrow \\
&\text{send}.B.TTP.s_B(f_{SUB}.B.Lb.T.Kb) \text{ in } tr \vee \\
&\text{send}.B.TTP.s_B(f_{SUB}.B.Lb.T.Kb) \notin X
\end{aligned}$$

Because of the unreliable channel between A and B, B may not obtain the evidence, but he can not be prevented from initiating the recovery protocol. Furthermore, if B has launched the recovery protocol, he then must be able to get the evidence from the TTP.

$$\begin{aligned}
FAIR4(tr, X) &\hat{=} \text{send}.B.TTP.s_B(f_{SUB}, A, Lb, T, Kb) \text{ in } tr \\
&\Rightarrow \\
&\text{get}.B.TTP.s_T(f_{CON}, B, A, Lb, T, Kb) \notin X
\end{aligned}$$

To meet the fairness property of the timed Zhou-Gollmann protocol, the process **Network** must satisfy the *FAIR1–FAIR4* in the stable failures model of CSP.

The formal verification shows there is no more fault in the timed ZG protocol under the assumptions described in this paper, other than the compromise caused by the introduction of the deadline T . As the designers say, the deadline T may result in the sender not getting the full evidence. In practice, the sender simply has to choose T big enough and send K to the responder only when it has sufficient time to launch the recovery protocol. In addition, the responder can be temporarily in an advantageous position, but both of them will be in a fair position at the end of the protocol run.

5 Discussion and Future Work

In this paper, we have modelled and analyzed the timed Zhou-Gollmann non-repudiation protocol. Fairness, an important property in a non-repudiation protocol, 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. In the CSP modelling, fairness is naturally described as a liveness property in the stable failures model.

Although the introduction of the deadline T makes the protocol closer to reality, it compromises the fairness of the parties. There are also two minor hidden issues: one is that the responder can be temporarily in a advantageous position, the other is that the sender may initiate the recovery protocol even when it has got the evidence. The evidence will mean the same to a judge regardless of whether it has been obtained through the main protocol or the recovery protocol, but it might be considered problematic that it is easy for the responder to prove that the initiator asked the TTP to intervene in the protocol execution. In the context of electronic commerce, it may result in bad publicity if it is known that the parties had to resort to the trusted third party to get the required evidence.

Some related work can be found in the literature concerning verification of non-repudiation protocols using different approaches. Zhou et al. in [ZG98] firstly use ‘BAN-like’ belief logic to check only safety properties of the non-repudiation protocols. Schneider [Sch98] 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. Shmatikov and Mitchell in [SM01] verify fairness as a monotonic property using Mur ϕ ; 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 [KR01] 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.

We have shown that the combination of CSP and FDR is an excellent tool to verify non-repudiation protocols. 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 extend our current model to cover this issue in future work.

We still have some distance to go towards our aim of proving fairness of the protocol in its full generality, with an unbounded number of participants and atomic messages. Evans in [Eva03] gives a useful start on this issue by using rank functions and a theorem prover, PVS, to verify safety properties. This approach allows one to deal with networks with an infinite number of states and even a infinite number of parties. In the future, we will investigate this approach and apply it in the analysis of liveness properties of non-repudiation protocols.

References

- [Eva03] Neil Evans. *Investigating Security through Proof*. PhD thesis, Royal Holloway, University of London, 2003.
- [For97] Formal Systems (Europe) Ltd. Failures-Divergence Refinement—FDR 2 user manual, 1997. Available from Formal Systems’ web site at <http://www.formal.demon.co.uk/FDR2.html>.

- [KMZ02] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An intensive survey of non-repudiation protocols. Technical Report 473, 2002.
- [KR01] Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Lecture Notes in Computer Science*, 2154, 2001.
- [MR99] Olivier Markowitch and Yves Roggeman. Probabilistic non-repudiation without trusted third party. In *Second Workshop on Security in Communication Network 99*, 1999.
- [Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1998.
- [Sch98] Steve A. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, 1998.
- [Sch99] Steve A. Schneider. *Concurrent and real-time systems: the CSP approach*. John Wiley & Sons, 1999.
- [SM01] Vitaly Shmatikov and John 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.
- [Ted83] Tom Tedrick. How to exchange half a bit. In *CRYPTO*, pages 147–151, 1983.
- [ZG96] Jianying Zhou and Dieter 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.
- [ZG97] J. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
- [ZG98] 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, September 1998.