Formal Analysis of FPH Contract Signing Protocol Using Colored Petri Nets

Magdalena Payeras-Capellà, Macià Mut-Puigserver, Andreu Pere Isern-Deyà, Josep L. Ferrer-Gomila, and Llorenç Huguet-Rotger

Departament de Matemàtiques i Informàtica, Universitat de les Illes Balears
{mpayeras,macia.mut,andreupere.isern,jlferrer,l.huguet}@uib.es

Abstract. An electronic contract signing protocol is a fair exchange protocol where the parties exchange their signature on a contract. Some contract signing protocols have been presented, and usually they come with an informal analysis. In this paper we use Colored Petri Nets to formally verify the fairness and the resistance to five previously described attacks of FPH contract signing protocol. We have modeled the protocol and the roles of the signers, a trusted third party, malicious signers as well as the role of an intruder. We have proven that the protocol is resistant to typical attacks. However, we have detected three cases where the protocol generates contradictory evidences. Finally, we have explained which should be the behavior of an arbiter to allow the resolution of these conflicting situations.

Keywords: contract signing protocol, Coloured Petri Nets, formal verification.

1 Introduction

Contract signing procedures, certified electronic mail or electronic purchases are good examples of fair exchange protocols. A fair exchange of values always provides an equal treatment to all users, and, at the end of the execution of the exchange, all parties have the element that wished to obtain, or the exchange has not been solved successfully (in this case, nobody has its expected element). These protocols make use of non-repudiation services, so they have to produce evidences to guarantee non-repudiation services. In case of dispute an arbiter has to be able to evaluate the evidences and take a decision in favor of one party without any ambiguity. Contract signing protocols allow the signature of a previously accorded contract by two or more signers. The fair exchange protocol ensures that at the end of the exchange all the signers have the signed contract or none of them have it. Fair exchange protocols often use Trusted Third Parties (TTPs) helping users to successfully realize the exchange. Several electronic contract signing protocols have been presented, with TTPs involved in different degrees. Among them there are a few proposals where the exchange can be finished in only three steps. Micali's protocol [1] and FPH protocol [2] are both efficient protocols with 3 messages in the exchange protocol. These protocols differ in the resolution protocol as well as in the elements exchanged in the three steps. However, they have another common aspects like the use of an off-line TTP, called optimistic approach. This concept of *optimistic protocol* was introduced in [3] by Asokan et al. In an optimistic fair exchange protocol the TTP only intervenes in case of problems to guarantee the fairness of the exchange.

Y. Chen et al. (Eds.): SecureComm 2009, LNICST 19, pp. 101–120, 2009.

Bao et Al. described [4] three attacks to Micali's protocol and proposed an improved protocol. Recently, Sornkhom and Permpoontanalarp [5] have applied a formal method to analyze the security of Micali's protocol by using Colored Petri Nets (hereinafter CPNs). This method allows the demonstration of the vulnerability of Micali's protocol to the three attacks described by Bao. Additionally, the method has been used to find two new attacks to Micali's protocol.

In this paper we have created a new model for the formal analysis of FPH protocol, similar to that used by Sornkhom and Permpoontanalarp but adapted to the features of the present analysis. Once the protocol is modeled, we can formally prove the behavior of the protocol in case of malicious users. Our first goal is to prove the fairness of this protocol; first we will do that in case of malicious signers, and then we have modeled a malicious intruder.

We have organized the paper as follows, in Section 2 we summarize FPH protocol with its security characteristics. Section 3 includes the description of the simulation model using CPNs. Section 4 presents the analysis of the protocol and the results obtained in different execution scenarios. Finally, section 5 includes the conclusions and describes future applications of the simulation model.

2 FPH Contract Signing Protocol

2.1 Ideal Features of a Contract Signing Protocol

Practical solutions for contract signing require of the existence and possible involvement of a TTP. To obtain efficiency, three objectives are usually pursued:

- To reduce the involvement of the TTP.
- To reduce the number of messages to be exchanged.
- Possible implication of the TTP should not require expensive operations, neither the storage of high volume of information.

The first objective has been achieved in some proposals. They are the optimistic solutions [3,6,7,8,9] and the TTP are not involved in every protocol run. Regarding the number of messages to be exchanged, [6] states that three is the minimum number of messages for a contract signing protocol. Protocols for contract signing have to provide evidence to parties to prove, at the end of the exchange, if the contract is signed and the terms of the contract. Some additional properties have to be achieved in optimistic protocols [7,9]:

- *Effectiveness*: if the parties behave correctly the TTP will not be involved;
- *Fairness*: no party will be in advantageous situation at any stage of a protocol run;
- *Timeliness*: parties can decide when to finish a protocol run;
- *Non-repudiation*: parties can not deny their actions;
- *Verifiability of the third party*: if the TTP misbehaves, all damaged parties will be able to prove it.

In this section we describe the FPH protocol that will be formally evaluated in next sections. This protocol achieves the previous requirements.

2.2 Description of FPH Contract Signing Protocol

It is assumed that both (*A*)lice and (*B*)ob have already agreed on a plaintext contract *C* before the exchange. Then they sign the contract using the protocol. The channel used among the signers is an unreliable channel, so it cannot be assumed that the messages sent through this channel arrive to their recipient. The channel between a signer and the TTP is a resilient channel, that is, the messages will eventually arrive to their recipient but the time of the arrival cannot be predicted. The originator, *A*, and the recipient, *B*, will exchange non-repudiation evidence directly. Only in case they cannot get the expected items from the other party, the TTP will be invoked, by initiating *cancel* or *finish sub-protocols*. The notation and elements used in the protocol description are in Table 1 while the *exchange sub-protocol* is described in Table 2.

Table 1. Elements

X, Y	Concatenation of two messages *X* and *Y*
H(X)	Collision-resistant one-way hash function of message *X*
$S_i(X)$	Digital signature on message *X* with the private key, or signing key, of *i* (using some hash function, H(), to create a digest of *X*)
i → j: X	*i* sends message *X* to *j*
M={*A,B,C*}	Message containing the contract to be signed, *C*, the originator, *A*(lice), and the recipient, *B*(ob)
$h_A = S_A(M)$	Signature of *A* on the contract *M*
$h_B = S_B(M)$	Signature of *B* on the contract *M*
$ACK_A = S_A(h_B)$	Signature of *A* on h_B; acknowledgement that *A* knows that the contract is signed, and is part of the necessary evidence for *B*
$ACK_T = S_T(h_B)$	Signature of the TTP on h_B; this is an equivalent acknowledgement to which *A* should have sent
$h_{AT} = S_A[H(M), h_A]$	Evidence that *A* has requested TTP's intervention
$h_{BT} = S_B[H(M), h_A, h_B]$	Evidence that *B* has requested TTP's intervention
$h'_B = S_T(h_B)$	Signature of the TTP on h_B to prove its intervention

Table 2. Exchange sub-protocol

1. *A* → *B*:	M, h_A
2. *B* → *A*:	h_B
3. *A* → *B*:	ACK_A

If the protocol run is completed, the originator *A* will hold non-repudiation (NR) evidence, h_B, and the recipient *B* will hold non-repudiation evidence, h_A and ACK_A. So the protocol meets the effectiveness requirement. If it is not the case, *A* or *B*, or both, need to rectify the unfair situation by initiating the *cancel* or *finish sub-protocol*, respectively, so that the situation returns to a fair position.

If *A* "says" (*A* could be trying to cheat or being in a wrong conception of the exchange state) that she has not received message 2 from *B*, *A* may initiate the *cancel sub-protocol* (Table 3).

Table 3. Cancel sub-protocol

	1'. $A \rightarrow T$:	H(M), h_A, h_{AT}
IF (*finished=true*)	2'. T:	retrieves h_B
	3'. $T \rightarrow A$:	h_B, h'_B
ELSE	2". $T \rightarrow A$:	$S_T(\text{"}cancelled\text{"}, h_A)$
	3". T:	Stores *cancelled=true*

In the *cancel sub-protocol*, the TTP will verify the correctness of the information given by *A*. If it is not the case, the TTP will send an error message to *A*. Otherwise, it will proceed in one of two possible ways. If the variable *finished* is *true*, it means that *B* had previously contacted with the TTP (see paragraph below), and the TTP had given the NR token to *B*, ACK_T. Now it has to give the NR token to *A*. So, it retrieves this stored NR token, h_B, and sends it to *A*, and a token to prove its intervention, h'_B. But if *B* had not previously contacted with the TTP, the TTP will send a message to *A* to cancel the transaction, and it will store this information (*cancelled = true*) in order to satisfy future petitions from *B*. Whatever case, now, we are again in a fair situation.

Table 4. Finish sub-protocol

	2'. $B \rightarrow T$:	H(M), h_A, h_B, h_{BT}
IF (*cancelled=true*)	3'. $T \rightarrow B$:	$S_T(\text{"}cancelled\text{"}, h_B)$
ELSE	3". $T \rightarrow B$:	ACK_T
	4". T:	stores *finished=true* and h_B

If *B* "says" that he has not received message 3, *B* may initiate the *finish sub-protocol* (Table 4). In the *finish sub-protocol*, the TTP will verify the correctness of the information given by *B*. If it is not the case the TTP will send an error message to *B*. Otherwise, it will proceed in one of two possible ways. If the variable *cancelled* is *true*, it means that *A* had previously contacted with the TTP (see paragraph above). The TTP had given a message to *A* to cancel the transaction, and now it has to send a similar message to *B*. Otherways, the TTP will send the NR token, ACK_T, to *B*. In this case the TTP will store the NR token, h_B, and will assign the value *true* to the *finished* variable, in order to satisfy future petitions from *A*. Again, whatever case, now, we are in a fair situation.

As a conclusion, the protocol is fair and we have not made timing assumptions (the protocol is asynchronous).

2.3 Informal Analysis of Fairness and Non-repudiation of FPH Protocol

After a protocol run is completed (with or without the participation of the TTP), disputes can arise between participants. We can face with two possible types of disputes: repudiation of *A* (*B* claims that the contract is *signed*) and repudiation of *B* (*A* claims that the contract is *signed*).

An external arbiter (not part of the protocol) has to evaluate the evidence held and brought by the parties to resolve these two types of disputes. As a result, the arbiter will

determine who says the truth. The arbiter has to know who is the originator and who is the recipient; remember that the contract, *M*, contains this information.

In case of repudiation of *A*, *B* is claiming that he received the signature on the contract *M* from *A*. He has to provide the following information to an arbiter: *M*, h_A and ACK_A or ACK_T. The arbiter will check if h_A is A's signature on M, and if it is positive the arbiter will assume that A had sent her signature to B. Then, the arbiter will check if ACK_A is A's signature on h_B, or it will check if ACK_T is TTP's signature on h_B. If this verification is positive, the arbiter will assume that either A or the TTP had sent an acknowledgement to B. Therefore, the arbiter will side with B. Otherwise, if one or both of the previous checks fails, the arbiter will reject B's demand. If the evidence held by B proves he is right, and A holds a message like $PR_T[H(\text{"}cancelled\text{"}, h_A)]$, it means that the TTP or A had acted improperly.

In case of repudiation of B, A is claiming that B had signed the contract M. She has to provide the following information to an arbiter: M and h_B. The arbiter will check if h_B is B's signature on M, and if it is positive the arbiter will assume that B had received M and h_A, and that he is committed to obtain the acknowledgement, ACK_A or ACK_T. If the previous verification fails, the arbiter will reject A's demand. If the verification is positive, the arbiter should interrogate B. If B contributes a *cancel* message, it means that *B* contacted with the TTP, and the TTP observed that *A* had already executed the *cancel sub-protocol*. For this reason the TTP sent the *cancel* message to *B*. Now it is demonstrated that *A* has tried to cheat. Therefore, the arbiter will reject *A*'s demand, and the arbiter will side with *B*. If *B* cannot contribute the *cancel* message, the arbiter will side with *A*.

As a conclusion, the protocol meets the non-repudiation requirement. Moreover, the protocol also fulfils the property of verifiability of the TTP [2]. This informal analysis doesn't cover all the possible situations derived of the execution of the protocol. It will be completed with a formal verification of the protocol (included in Section 4) resulting from the use of the model based on Petri Nets described in Section 3.

3 Description of the Model Used for the Formal Analysis of Fair Exchange Protocols

3.1 Colored Petri Nets

CPN (Colored Petri Nets) is a discrete-event modeling language combining Petri Nets with a programming language called standard ML [10]. Petri Nets are capable to provide the interaction between processes and the programming language is used for the definition and manipulation of the data types. So, CPN can be used as a formal method to analyze distributed systems and communication protocols. A CPN model is an executable model representing the states of the system and the transitions that can cause a change of the state of the system. CPN contains four kinds of components:

- *Places*. They represent the system state at a given time. The places change from the activation of the transitions.
- *Transitions*. They are the actions which implies a state change.
- *Arcs*. They are the links between places and transitions.

– *Color sets*. The tokens that move through the states and transitions have a value, called color.

The global system state, after firing an event, is called *marking*. So, a *marking* is like a photo of the state of the system after each event. One of the tools that implement CPN is CPNTools [10]. This is the tool we have used in this work. When the model is designed, we can submit a simulation process in order to generate the state space. The state space is the set of markings between initial and final event. Therefore, we extract a complete definition of the system behavior along its execution.

3.2 General Assumptions and Methodology

In order to use Petri Nets to model the protocol, a number of general assumptions are made:

– Each party in the model has a unique identifier.
– Each party already knows the public keys of the others.
– Cryptographic algorithms used in the model are secure.
– The messages sent between the TTP and any party will always be delivered to the intended destination without modification (resilient channel).

The methodology followed to analyze the fairness of the protocol is:

– Build the model
  – Declare color sets (*colsets*) to represent messages and elements in the protocol.
  – Create top-level net to model the parties.
  – Create entity-level net to model the behavior of each party.
  – Create process-level net for each entity-level.
  – Declare functions and variables that will be used in the model.
– Generate the state space
  – Set up initial marking for each party.
  – Generate the state space of the model using CPNTools.
– Create query functions to search for attack states.
– Extract attack scenarios using paths between states if attacks are found.

3.3 Description of the Model

In our model, based on Sornkhom and Permpoontanalarp's model [5], we have four key parties: Alice (*A*), Bob (*B*), Intruder (*I*) and TTP. While the TTP is strictly honest, the other parties can take the role of a malicious party. *A* and *B*, in their malicious role (A_m and B_m respectively), can stop the exchange or they can contact to the TTP in many different steps and this way, they could try to cheat the other party. *I* is a malicious party who can acts as an observer, like a man in the middle, and moreover he can deploy many other tasks: *drop*, *store*, *forward* or *modify* messages in transit sent by any party involved in the exchange.

In order to model the *drop* and *stop* events made by malicious parties (e.g. A_m, B_m or *I*), the model has a mechanism to inform about these events to the other involved

parties. When an event occurs, a message is immediately sent by the party who drops the message or stops the exchange to the other parties involved. This assumption helps us to avoid the use of a timeout on each party. When an event message is received, the party could act contacting the TTP or maybe stopping the exchange depending of which is the current protocol step.

Another important consideration is: messages between the TTP and any other party of the model will always be delivered to the intended destination without any modification.

With the provided data, we are able to build an scenario that can be used to model the protocol using different attack sessions, where each session can involve an initiator (e.g. A or A_m) and a responder (e.g. B or B_m). Note that I and TTP are implicitly present in every session trace. So, we can deploy four sessions: (A, B), (A_m, B), (A, B_m) and (A_m, B_m), where (X, Y) denotes which party is the initiator (X) and the responder (Y). In this paper, we won't consider parallel session attacks where malicious parties can be involved in multiple and concurrent sessions, and this task will be deployed in further works.

The architecture of the model can be divided into three big blocks, using a top-down technique: top, entity and process levels. All messages sent by any party are a combination of source, destination and a protocol message as a payload.

The top level scheme (Fig. 1) shows basic interaction with all parties involved in the protocol and the message flow between these parties. In the top level we can see the contents of each party's database, which contains the protocol messages sent and received by each party. Finally, we can see and control the content of the session. The

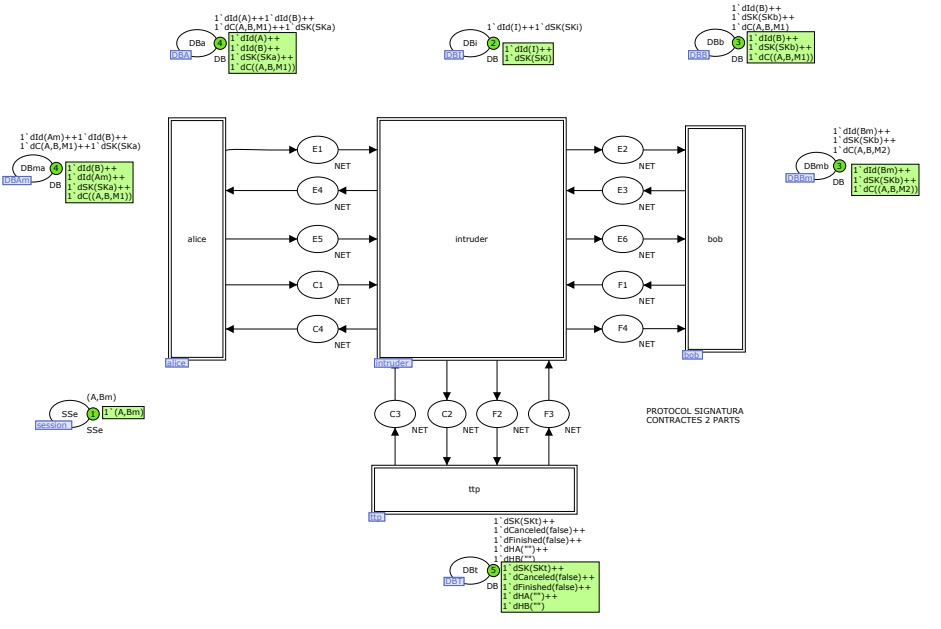

Fig. 1. Top level scheme

variable controls that will be used to distinguish the roles of the parties involved in the protocol execution (e.g. honest or malicious role). Moreover, in Fig. 1 we can see as the messages always are intercepted by *I* in their transit between parties.

The entity level shows us a more detailed model of the protocol and denotes all the steps each party can execute. In Fig. 2 we can see the entity level of *A* and her two roles. Transitions TA_1 to TA_4 are the transitions corresponding to her honest role, and TAm_1 to TAm_4 are the transitions of the malicious role. The first transitions of *A*, TA_1 and TAm_1, are to generate the first protocol message and send it to *B*. The transitions TA_2 and TAm_2 are to receive and verify the second message sent by *B* and send to *B* the third message. TA_3 and TAm_3 have the responsibility to contact the TTP using the *cancellation sub-protocol*, and the last transitions TA_4 and TAm_4 are to receive the response from the TTP. Note that the selection of the transitions that will be executed is done by the session configuration which tells if the party is honest or malicious.

B's entity level, as it is shown in Fig. 3, like *A*'s entity level, implements the honest (TB_1 to TB_3) and malicious (TBm_1 to TBm_3) roles of *B*. TB_1 and TBm_1 are to receive and verify the first message of the protocol and they also send the second message, while TB_2 and TBm_2 are to receive and verify the third protocol message and, if it is needed, these transitions are able to contact the TTP. At last, TB_3 and TBm_3 are to receive the response from the TTP.

The process level implements all the actions deployed by the users and specifies how the relations between the entities are. The actions deployed by each process are atomic, e.g. only one process can be executed at the same time. This can be done by a unique token, which is shared between all parties of the model. It is captured by each party when a process starts, and it will be released when the process ends. Moreover, each process level is controlled by a session flow control mechanism. This mechanism

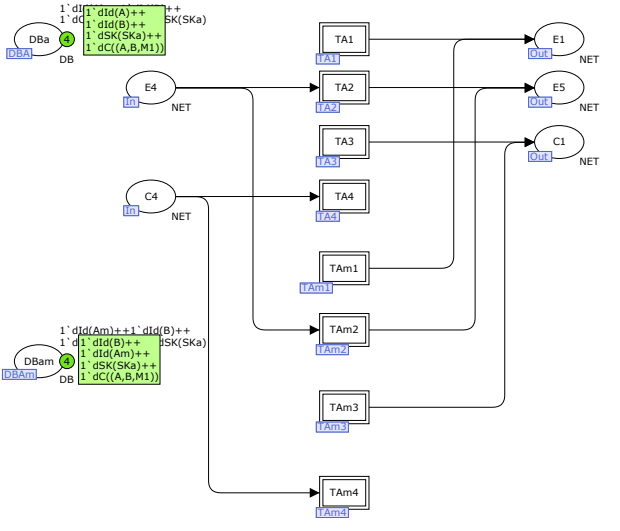

Fig. 2. *A*'s entity level

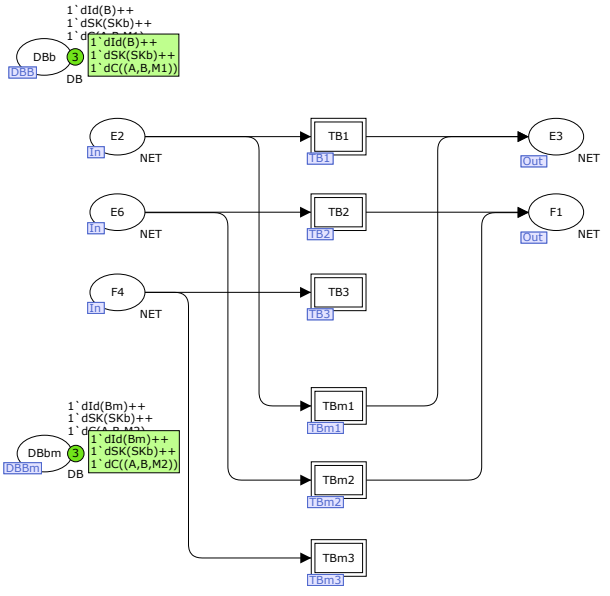

Fig. 3. *B*'s entity level

is defined like a token which passes through parties and at every step, they change it contents. This token controls the order in that actions will be done. For example, it controls that a message generation should be executed after the verification step.

3.4 Query Functions

In order to extract attack scenarios from state spaces we have developed a set of query functions, such that of Fig. 4 to find special contents in each party database. The main function is *SearchCommitsTerminalNodes(ack,id)*, where *ack* is the element or commit we would like to search in the database of the *id* party. This function returns a list of markings which fulfill some conditions. The function is build around the use of standard query function *PredNodes(p1,p2,p3)*. The first parameter is another custom query function named *SearchCommits(ack,id)*, where *ack* and *id* have the same use as in the previous query. This function is capable to take up the contents of the desired database *id* and tell us if the *ack* is in the database. The second parameter is to choose only markings which are leaf markings, e.g. terminal markings, which are markings that contain a complete execution of the protocol. The last parameter, *NoLimit*, tells the query should walk all markings and return all results.

The main query can be used to analyze the fairness property. In order to do this, we apply the query function against the parties involved in the exchange, depending of the session, to search the desired commit. The function will return a list of terminal markings. The analysis of this list will tell us if the exchange is fair or not.

```
fun SearchCommits( ack:DB, id:Id ) : Node list
  = PredAllNodes(
      fn n =>
      let
          val dba = Mark.Top'DBa 1 n
          val dbb = Mark.Top'DBb 1 n
          val dbi = Mark.Top'DBi 1 n
          val dbam = Mark.Top'DBma 1 n
          val dbbm = Mark.Top'DBmb 1 n
      in
          if (id=A) then
             cf(  ack ,  dba ) > 0
          else if (id=B) then
             cf( ack , dbb ) > 0
          else if (id=Am) then
             cf( ack, dbam ) > 0
          else if (id=Bm) then
             cf( ack, dbbm ) > 0
          else (* id=I *)
             cf( ack , dbi ) > 0
      end
)

fun SearchCommitsTerminalNodes( ack:DB, id:Id ) : Node list
    = PredNodes ( (SearchCommits(ack,id)) ,
                  fn n => (Terminal n) andalso (FullyProcessed n),
                  NoLimit)
```

Fig. 4. Search query functions developed in order to search commits into the party's databases

4 Formal Analysis of FPH Contract Signing Protocol

4.1 Evaluation of the Vulnerability to Previously Defined Attacks

Until today, several attacks to contract signing protocols have been described. Bao et Al. [4] found three attacks to Micali's ECS1 protocol (Table 5). Later Sornkhom and Permpoontanalarp [5] found two new attacks to the same protocol. The consequence of these attacks is the loss of fairness. For this reason, we have used the model based on CPN described in last section to evaluate the resistance of FPH protocol to all these attacks.

Micali's ECS1 protocol (Table 5) and FPH protocol are similar, so we will use the same notation to describe them. Moreover, we will use $E_X(Y)$ to denote the encryption using the public key of *X* of the message *Y*. *A* is committed to the contract, *C*, as an initiator if *B* has both $S_A(C, Z)$ and *M* where $Z = E_{TTP}(A, B, M)$ and *M* is a random. On the other hand, *B* is committed to *C* as a responder if *A* has both $S_B(C, Z)$ and $S_B(Z)$.

Now we are going to describe the five attacks to Micali's protocol and apply them to FPH protocol, then we will use the model to prove both the fairness and its resistance to these attacks.

Table 5. Micali's ECS1 protocol definition

	$A \rightarrow B$:	$S_A(C, Z)$
	$B \rightarrow A$:	$S_B(C, Z), S_B(Z)$
IF (Both signatures are valid)	$A \rightarrow B$:	M
IF (B receives valid M such that $Z = E_{TTP}(A, B, M)$)		The exchange is completed
ELSE	$B \rightarrow$ TTP:	$A, B, Z, S_B(C, Z), S_B(Z)$
	TTP $\rightarrow A$:	$S_B(C, Z), S_B(Z)$
	TTP $\rightarrow B$:	M

Bao's First Attack. A is a malicious initiator and sends a false element in step 1. In Micali's protocol this attack (Table 6) can be done if *A* sends a false Z where $Z = E_{TTP}(A, B, M)$. In this case, *A* can always obtain *B*'s commitment but *B* will not have *A*'s commitment. This attack is possible because *B* cannot verify the elements received in step 1.

Table 6. Bao's First Attack Trace

A → B:	$S_A(C, Z)$ where $Z = E_{TTP}(A, B, M)$
B → A:	$S_B(C, Z), S_B(Z)$
A → B:	Nothing
B → TTP:	$A, B, Z, S_B(C, Z), S_B(Z)$
TTP → A:	Nothing
TTP → B:	Nothing

In order to detect the attack on the model, we have generated a session with A_m (*A* acting maliciously) and *B*, as we can see on Fig. 5. In this attack, A_m builds a false contract M_2 and she sets an arbitrary initiator (*X*) and arbitrary responder (*Y*). The first query searches h_A element in A_m's database, finding four cases, corresponding to markings 20, 21, 22 and 37. The second query searches the same element, h_A, in *B*'s database and as we can see, *B* never has this element. This is because the verification stage fails and *B* never stores the received message. The two last queries search the response of the TTP into A_m's database, and we can see that A_m only receives a *cancel* message (marking 37) and she never obtains the NR evidence from the TTP.

Then, FPH protocol is not vulnerable against Bao's *first attack*, because *B* verifies the elements received in step 1 and in case of *attack 1* he doesn't send the message of step 2. Then *A* will not send message 3. If *A* tries to contact the TTP, the TTP will send a cancellation proof and stores *cancelled=true*. *B* will not contact the TTP because he doesn't have any valid element from *A*.

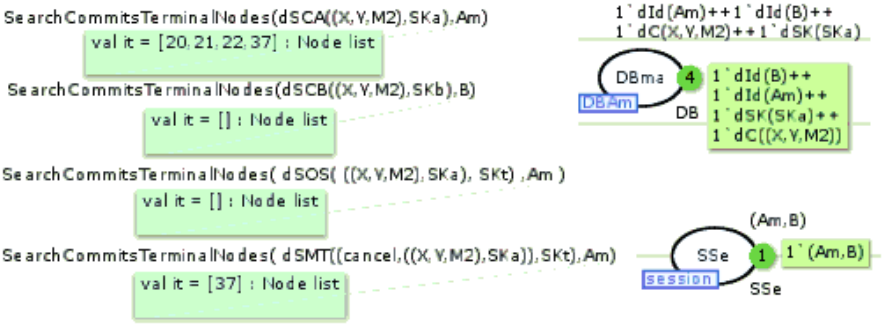

Fig. 5. First attack query results, A_m database contents and session configuration

Bao's Second Attack. *A* conspires with another initiator *A'* and changes her identity in step 1. In Micali's protocol this attack (Table 7) can be done if *A* conspires with *A'* and sends a false *Z* where $Z = E_{TTP}(A', B, M)$. In this case, malicious *A* can always obtain *B*'s commitment on a contract between *B* and *A'*, but *B* will not have anything. This attack is possible because *B* cannot verify the identity of *A* in the element received in step 1.

Table 7. Bao's Second Attack Trace

A → B:	$S_A(C, Z)$ where $Z = E_{TTP}(A', B, M)$
B → A:	$S_B(C, Z), S_B(Z)$
A → B:	Nothing
B → TTP:	$A, B, Z, S_B(C, Z), S_B(Z)$
TTP → A:	Nothing
TTP → B:	Nothing

The *second attack* can be detected in the model using the same session configuration (A_m, B) as the *first attack*, but using a different contract. In this case, we have built a false contract with a confabulated initiator (*X*), the initial receiver (*B*) and the previously accorded plain contract (M_1). As we can see in Fig. 6, the query results are the same as in the *first attack*, the second function never returns any result because *B* never builds message 2. Then, if we search the TTP's response on A_m's database, we can see A_m never obtains the NR and she only could have a cancellation proof.

So, FPH protocol is not vulnerable against this attack. *B* verifies the elements received in step 1 and in case of *attack 2* he doesn't send the message of step 2, as in *attack 1*. Then *A* will not send message 3 and the exchange will be stopped and *A* will not obtain *B*'s commitment. If *A* tries to conclude the exchange contacting the TTP, she will receive a cancellation proof. On the other side, *B* will not contact the TTP because he doesn't want to finish the exchange because he knows that the element sent in step 1 is false and, moreover, he hasn't sent any element.

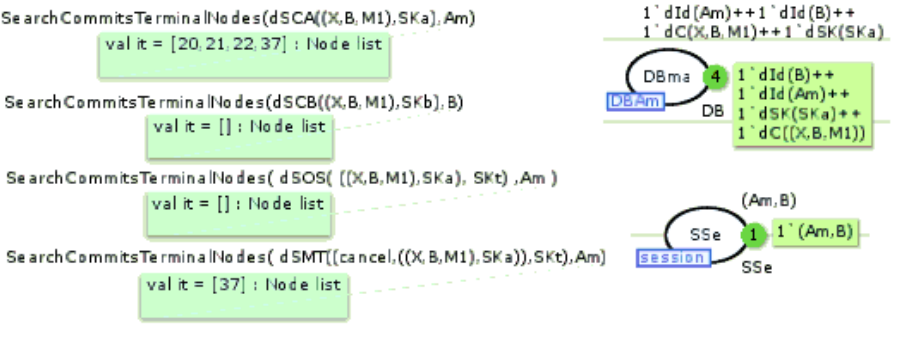

Fig. 6. Second attack query results, A_m database contents and session configuration

Table 8. Bao's Third Attack Trace

A → B:	$S_A(C, Z)$ where $Z = E_{TTP}(A', B, M)$
B → TTP:	$Z, S_B(C', Z), S_B(Z)$ for a false contract C'
TTP → A:	$S_B(C', Z), S_B(Z)$
TTP → B:	M

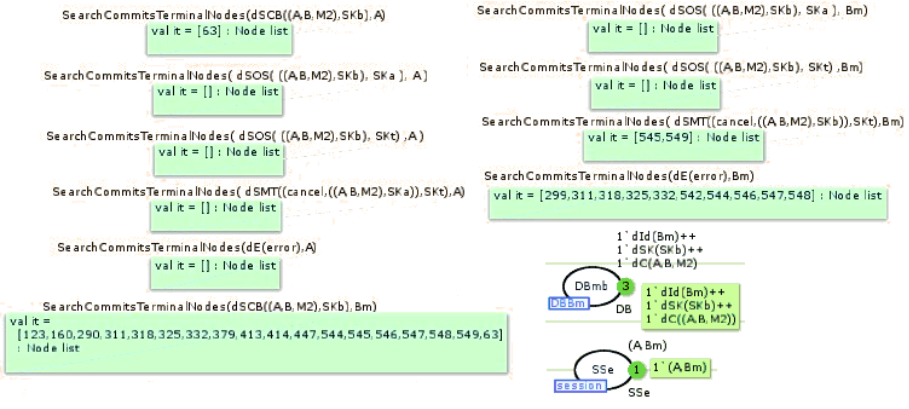

Fig. 7. Third attack query results, B_m database contents and session configuration

Bao's Third Attack. Malicious *B* contacts the TTP and requests the resolution with a false contract. In Micali's protocol this attack (Table 8) can be done if *B*, after the reception of a valid message in step 1, contacts the TTP to start the resolution of the exchange. In this request *B* includes a fake contract. In this case, malicious *B* always gets *A*'s commitment on the original contract, but *A* obtains *B*'s commitment on the false contract (selected by *B*). This attack is possible because *A* cannot request the resolution of the exchange and obtains from the TTP the elements resulting of the resolution started by *B*.

The *third attack* can be verified with the model using a session configuration where *A* is the honest initiator and B_m is the malicious responder, e.g. (A, B_m). B_m builds a contract containing a false plain text (M_2) but using the real initiator and responder. As we can see in Fig. 7, when B_m receives the first message, he changes its contents by setting a different plain contract (M_2). Then, we have searched if a false h'_B sent by B_m is into *A*'s database and, effectively, it is in marking 63. Although *A* stores the message, she verifies it and she decides it is wrong and she doesn't generate the third message. Then *A* can contact the TTP, but she would ask for the original real contract using the *cancellation sub-protocol* and the TTP will send a cancellation proof to *A*. Finally, we can search the TTP's responses in B_m's database and we can see that he never obtains the alternative proof. Moreover, he can only obtain the cancellation proof and an error message because the TTP's verification fails.

In FPH protocol, however, when *A* receives a false $h'_B = S_B(M', A, B)$ in step 2, she detects the attack, stops the exchange and contacts the TTP. If *B* has contacted the TTP in first place and the request contained a false h_B, the TPP has been able to detect

that h_A and h_B are not related with the same contract. Then, when *A* sends a resolution request, the TTP will send her a cancellation proof, so the contract will not be signed. If *A* contacts the TTP in first place, she will obtain a cancellation proof.

Fourth Attack. An Attacker eavesdrops B's commitment. The *fourth attack* was described in [5] and it is possible because Micali's protocol has an incomplete definition on *B*'s commitment. The message $(S_B(C, Z), S_B(Z))$ is the evidence to prove that *B* has committed himself to contract *C* with any initiator. The evidence is not linked to the initiator, so anybody who has it can claim to be an initiator of the contract committed by *B*.

The *fourth attack* can be detected using a session between two honest parties, *A* and *B*. As we can see in Fig. 8, the databases of *A* and *B* contain the previously committed contract. In this case, we would search states where an intruder, *I*, eavesdrops messages. So, in the first query we will find one state where *I* changes the initiator of the contract. This message is found on *B*'s database and finally, using the third query, we can prove how *B* never builds his commit, h_B, over the wrong contract with *I* as initiator.

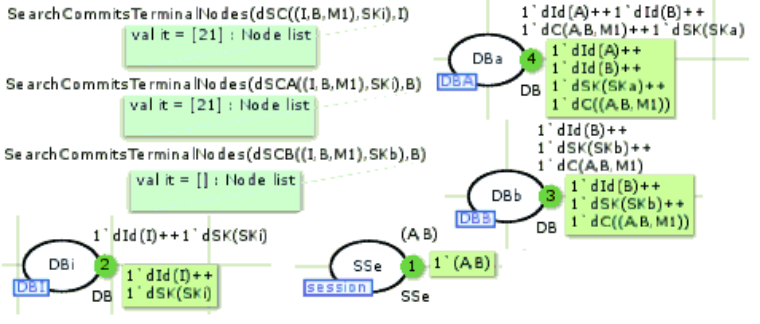

Fig. 8. Fourth attack query results, parties' database contents and session configuration

In contrast to Micali's protocol, FPH protocol has linked *B*'s commitment to the contract. The evidence is the message $h_B = S_B(M)$, however holding this evidence is not enough for anyone to prove that *B* has committed himself to contract *C*. Because FPH protocol specifies that *M* contains the contract to be signed, C, and it indicates who is the originator, *A*, and who is the recipient, *B*. Thus, FPH protocol is resistant to this attack.

Fifth Attack. Swapping the initiator and the responder role. In the *fifth attack* (Table 9) described in [5] a malicious *A* can get *B*'s commitment on a contract between *B* as an initiator and any conspired party A_r as a responder. But *B* will not get anything. In order to perform the attack, *A* involves *B* in the protocol so as to exchange the commitments on a contract. But *A* build a fake item *Z* with the identity of *B* as the initiator and a conspiring party A_r as the responder: $Z = E_{TTP}(B, A_r, M)$. Finally *A* will give $S_B(C, Z)$ and *M* to A_r. The TTP can't send anything to *A* and *B* because item *Z* doesn't fulfill the protocol specifications. Now, A_r can successfully claim *B*'s commitment on the contract as an initiator and *B* doesn't have any kind of evidence.

Table 9. Fifth Attack Trace

A → B:	$S_A(C, Z)$ where $Z = E_{TTP}(B, A_r, M)$
B → A:	$S_B(C, Z), S_B(Z)$
A → B:	Nothing
B → TTP:	$A, B, Z, S_B(C, Z), S_B(Z)$
TTP → A:	Nothing
TTP → B:	Nothing

The *last attack* reconstruction (Fig. 9) uses a session composed by A_m and B. In this case, A_m changes the contents of the contract, swapping party's roles but she uses the previously committed contract (M_1). The application of the query functions against the model (Fig. 9) is the same as in the *first* and *second attacks*. The step 1 searches in A_m's database the first element h'_A and then the second query searches the second message into *B*'s database. As it is shown, *B* does not build it. Finally, the third and fourth queries try to search responses sent by the TTP into A_m database. As we can see, she will only obtain a cancellation proof (marking 37).

However, in FPH protocol, as we have already explained, *B* verifies the item received at the step 1 of the protocol. Thus, if *A* has made improper changes in the message, *B* will detect it. Then, he will not continue and he will not send the message of step 2. Therefore, the attack described here will not be successful.

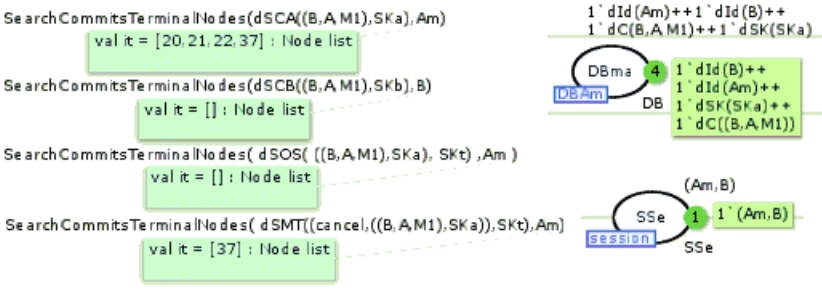

Fig. 9. Fifth attack query results, A_m database contents and session configuration

4.2 Fairness Analysis

In this section we will describe some conflicting situations in FPH protocol where the signers have contradictory evidences (see Section 2.3). The evidences generated by this protocol are not transferable, and an arbiter must contact both signers to solve a dispute, know the final state of the exchange and guarantee non repudiation. This property has been described in [11] and is called weak fairness. In this formal analysis of the fairness of the protocol we will prove that the arbiter can solve all kinds of conflicting situations derived from the execution of the protocol.

In [2], we described a conflicting situation where *A* can obtain NR evidence from *B* (h_B) and a *cancel* message from *T*, while *B* obtain NR evidence from *A* (h_A, ACK_A). *A* can do it, for instance, invoking the *cancel sub-protocol* after the end of the *exchange*

sub-protocol. It seems that *A* can affirm that the contract is signed or is not signed (*cancelled*), depending on her usefulness, while *B* possesses NR evidence that will prove that the contract is signed. We have detected this situation in the formal analysis and we have called it *case 1*.

Moreover, thanks to our model we have discovered two more conflicting situations. The first one (we will call it *case 2*) is produced when a malicious *A* invokes the *cancel sub-protocol* after the end of the *exchange sub-protocol* (as in *case 1*) and then a malicious *B* executes it, too. It seems that *A* and *B* can state that the contract is signed or is not signed (*cancelled*), depending on her usefulness.

The last conflicting case we have detected (*case 3*) is achieved when the exchange is stopped after the step 2. In this case *A* has the NR evidence from *B* while *B* does not have the NR evidence from *A*. Both parties can contact the TTP. If *B* contacts in first place the TTP will send him the NR evidence and the contract will be signed. Instead, if *A* contacts in first place, the TTP will cancel the exchange and then *A* would have NR evidence from *B* (h_B) and a *cancel* message from *T*, while *B* obtains the *cancel* message.

The three conflicting situations are found in the model deploying a session composed by a malicious initiator or both a malicious initiator and a malicious responder, e. g. (A_m, B_m). This way, all possible behaviors of both parties are contemplated. Using the already known query functions, as shown in Fig. 10, we have searched into each party's database the desired commits. In this case, we have searched the second and third messages of the *exchange sub-protocol* and all the responses received from the TTP.

If we study the list of markings obtained from each query, we can build Table 10 with the three cases previously described. For each case, we denote the state of the contract (signed as S and cancelled as C) and either if A_m or B_m have contacted,

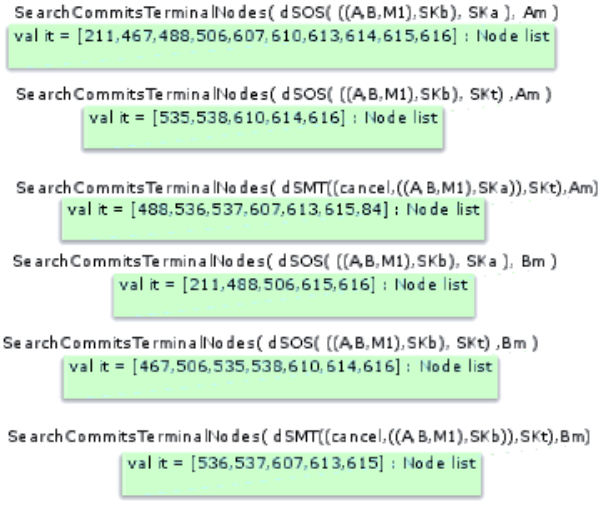

Fig. 10. Query functions results over the model with (A_m, B_m) session

Table 10. List of the markings corresponding to the three cases with contradictory evidences

Case	Marking	A_m has NR	B_m has NR	A_m contacts TTP	B_m contacts TTP
1	488	S & C	S	Yes (M)	No
2	615	S & C	S & C	Yes (M)	Yes (M)
3	607	S & C	C	Yes (M)	Yes (H)
3	613	S & C	C	Yes (M)	Yes (H)

Table 11. Scenarios without contradictory evidences

Marking	A_m has NR	B_m has NR	A_m contacts TTP	B_m contacts TTP
84	C	Nothing	Yes (H)	No
211	S	S	No	No
467	S	S	No	Yes
506	S	S & S (by TTP)	No	Yes
535	S (by TTP)	S (by TTP)	Yes (H)	Yes (H)
536	C	C	Yes (H)	Yes (H)
537	C	C	Yes (H)	Yes (H)
538	S (by TTP)	S (by TTP)	Yes (H)	Yes (H)
610	S & S (by TTP)	S (by TTP)	Yes (M)	Yes (H)
614	S & S (by TTP)	S (by TTP)	Yes (M)	Yes (H)
616	S & S (by TTP)	S & S (by TTP)	Yes (M)	Yes (M)

maliciously (M) or honestly (H), the TTP. As shown in Table 10, using the model we have located the three cases where A_m and B_m have contradictory evidences although we have detected four possible scenarios, because *case 3* could appear twice.

As we can see, *case 1* happens on marking 488, when A_m obtains the NR evidence from B_m (so *A* has evidence that the contract is *signed*), but she contacts the TTP in order to cancel the exchange. This is a malicious behavior, because A_m shouldn't contact the TTP to cancel an exchange that is already finished. The TTP sends A_m the *cancel* message and then, A_m could affirm that the contract is *signed* or *cancelled*. B_m receives the NR from A_m and he doesn't need to contact the TTP.

In *case 2*, corresponding to marking 615, the *exchange sub-protocol* ends successfully for each party, but A_m contacts the TTP, after the transfer of NR evidence to B_m, in order to obtain a *cancel* message. Once B_m receives the NR evidence from A_m, she also contacts the TTP and he obtains a *cancel* message. So, A_m and B_m have a malicious behavior because they contact the TTP when they shouldn't.

Case 3 is detected twice on the model. In both scenarios, A_m obtains the NR from B_m but B_m never receives the third message. In each scenario, A_m executes maliciously the *cancellation sub-protocol* and she receives a *cancel* message from the TTP. In the first scenario, corresponding to marking 607, A_m decides to maliciously stop the exchange and she doesn't send the third message to B_m. In the other hand, marking 613 is the result of a *drop* event of the third message by an intruder, *I*. From the point of view of B_m, both scenarios are the same, and he will contact to the TTP in order to resolve the situation obtaining a *cancel* message for each scenario.

In addition to the conflicting cases, there are other cases detected by the model where there aren't contradictory evidences but, in some cases, each party could have repeated proofs because they may contact the TTP when the protocol is successfully ended. Table 11 displays the scenarios without contradictory evidences.

The most interesting cases displayed on Table 11 are markings 84 and 616. In marking 84, B_m has nothing from A_m because an intruder *I* has executed a drop event on the first message. B_m cannot execute the *finalization sub-protocol* because he doesn't have any valid element from A_m. In the other hand, A_m resolves the contract executing the *cancellation sub-protocol* obtaining a *cancel* message from the TTP. The second marking, 616, is the case where A_m and B_m act maliciously contacting the TTP when the *exchange sub-protocol* ends successfully. It is similar to case 615, but this time, B_m contacts in first place the TTP, obtaining the corresponding NR evidence. Then, if A_m tries to cancel, the TTP sends a NR evidence that states that the contract is *signed*.

In order to solve these conflicting situations an arbiter must always contact both parties, and in case of contradictory evidences, we have established that he must act as follows:

- *Case 1*: *A* can state that the contract is *signed* or *cancelled*, but *B* possesses NR evidence that will prove that the contract is *signed*. If *A* tries to use the *cancel* message she will be proving she is a cheating party, so the arbiter will side with *B*.
- *Case 2*: As in *case 1*, an arbiter will contact both parties in case of contradictory evidences. If *B* shows NR evidence that will prove that the contract is *signed*, the arbiter can state that the contract is *signed*, and if *B* shows that the contract is *cancelled* the arbiter will state that the contract is *cancelled*. This way, due to the fact that *A* is always the first cheating party, if the arbiter sides always with *B*, the protocol will discourage *A* to act fraudulently.
- *Case 3*: Once again, *A* has acted fraudulently, and if the arbiter sides with *B* he will state that the contract is *cancelled*.

As a conclusion, we have detected the previously defined conflicting situation and we have discovered two additional cases. All the cases are due to the fraudulent behavior of *A*. To solve these situations, an arbiter must contact both parties and in case of conflict he must always side with *B*. This way, the protocol will be fair in all cases and moreover the fraudulent behavior of the parties is discouraged.

5 Conclusions and Future Work

In this paper we have formally analyzed, using a formal method (Petri Nets), an efficient contract signing protocol, FPH protocol [2], known as one of the solutions involving only three messages, as Micali's protocol. But, while Micali's protocol has been flawed (three attacks were found by Bao et.al. and two more attacks were found by Sornkhom and Perpoomtanalarp), FPH protocol is not vulnerable to any of these attacks due to its features. We have evaluated FPH protocol using a model that assumes that all the signers can be dishonest and an intruder can also attack the exchange, and we have proven the resistance to all these attacks using the model.

We have evaluated all the possible situations involving malicious users and intruders, and in all cases the exchange ends in a fair situation. Moreover, we have also detected that there are three cases in where, although the exchange is fair, one of the signers (or both) can have contradictory evidences. For these reason, although the exchange is fair, we cannot say that the proofs generated by the protocol are transferable, because both parties have to be interrogated by an arbiter to know the final state of the exchange. Finally, we have created a set of rules to determine the role of the arbiter in order to achieve fairness even when contradictory evidences are presented.

With the model created to evaluate the vulnerability of the protocol to previously described attacks and prove the fairness of the protocol we will be able, in a near future, to formally analyze other properties of the protocol, such as the verifiability of the TTP and also try to model more complex protocols such as a multiparty contract signing protocol. Moreover, we will adapt the model to work with new attack scenarios, like confabulated attacks using data from two different signature sessions. In parallel, we will work in the improvement of the model in order to include more control over the intruder's behavior and some other enhancements.

Acknowledgement

This work is partially supported by MEC and FEDER under projects: "Seguridad en la Contratación Electrónica basada en Servicios Web" (CICYT TSI2007-62986) and ARES "Grupo de Investigación Avanzada en Seguridad y Privacidad de la Información" (Consolider - Ingenio CSD2007-004). We would like to thank Yongyuth Permpoontanalarp for his useful comments and support during the development of this work.

References

1. Micali, S.: Simple and Fast Optimistic Protocols for Fair Electronic Exchange. In: Proceedings of 21st Symposium on Principles of Distributed Computing, pp. 12–19 (2003)
2. Ferrer-Gomila, J., Payeras-Capellà, M., Huguet-Rotger, L.: Efficient Optimistic N-Party Contract Signing Protocol. In: Davida, G.I., Frankel, Y. (eds.) ISC 2001. LNCS, vol. 2200, pp. 394–407. Springer, Heidelberg (2001)
3. Asokan, N., Shunter, M., Waidner, M.: Optimistic Protocols for Fair Exchange. In: 4th ACM Conference on Computer and Communications Security, pp. 7–17 (1997)
4. Bao, F., Wang, G., Zhou, J., Zhu, Z.: Analysis and Improvement of Micali's Fair Contract Signing Protocol. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 176–187. Springer, Heidelberg (2004)
5. Sornkhom, P., Permpoontanalarp, Y.: Security analysis of micali's fair contract signing protocol by using coloured petri nets. In: 9th ACIS Int. Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 329–334 (2008)
6. Ferrer-Gomila, J.L., Payeras-Capellà, M.M., Huguet-Rotger, L.: Optimality in asynchronous contract signing protocols. In: Katsikas, S.K., López, J., Pernul, G. (eds.) TrustBus 2004. LNCS, vol. 3184, pp. 200–208. Springer, Heidelberg (2004)
7. Asokan, N., Shoup, V., Waidner, M.: Asynchronous Protocols for Optimistic Fair Exchange. In: IEEE Symposium on Research in Security and Privacy, pp. 86–99 (1998)
8. Garay, J.A., Jakobsson, M., MacKenzie, P.: Abuse-free optimistic contract signing. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 449. Springer, Heidelberg (1999)

9. Zhou, J., Deng, R., Bao, F.: Some remarks on a fair exchange protocol. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 46–57. Springer, Heidelberg (2000)
10. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. Intenationals Journal on Software Tools for Technology Transfer, 213–254 (2007)
11. Kremer, S., Markowitch, O., Zhou, J.: An Intensive Survey of Fair Non-Repudiation Protocols. Computer Communications 25, 1606–1621 (2002)