Noname manuscript No.
(will be inserted by the editor)

Game-Based Verification of Contract Signing Protocols with Minimal Messages

Ying Zhang · Chenyi Zhang · Jun Pang · Sjouke Mauw

Received: date / Accepted: date

Abstract A multi-party contract signing (MPCS) protocol is used for a group of signers to sign a digital contract over a network. We analyse the protocols of Mauw, Radomirović and Torabi Dashti (MRT), using the finite-state model checker Mocha. Mocha allows for the specification of properties in alternating-time temporal logic (ATL) with game semantics, and the model checking problem for ATL requires the computation of winning strategies. This gives us an intuitive interpretation of the verification problem of crucial properties of MPCS protocols. MRT protocols can be generated from minimal message sequences, depending on the number of signers. We discover an attack on fairness in a published MRT protocol with three signers and a general attack on abuse-freeness for all MRT protocols. For both attacks, we present solutions. The abuse-freeness attack leads us to a revision of the methodology to construct an MRT protocol. Following this revised methodology, we design a number of MRT protocols using minimal message sequences for three and four signers, all of which have been successfully model checked in Mocha.

Keywords: Contract signing, alternating-time temporal logic, model checking, fairness, abuse-freeness.

Y. Zhang
Computer Science and Communications
University of Luxembourg, Luxembourg &
School of Computer Science and Technology
Shandong University, China

C. Zhang
School of Information Technology and Electrical Engineering
University of Queensland, Australia
E-mail: chenyi@uq.edu.au

J. Pang (corresponding author)
Computer Science and Communications
University of Luxembourg, Luxembourg
E-mail: jun.pang@uni.lu
Phone: +352 4666445625
Fax: +352 4666445500

S. Mauw
Computer Science and Communications &
Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg, Luxembourg
E-mail: sjouke.mauw@uni.lu

1 Introduction

The goal of a multi-party contract signing (MPCS) protocol is to allow a number of parties to sign a digital contract over a network. Such a protocol is designed as to ensure that no party is able to withhold his signature after having received another party's signature. A simple way to achieve this is to introduce a trusted third party (T). The trusted third party simply collects signed contracts from all signers, verifies the signatures, and then distributes them back to the signers. A major drawback of this approach is that the trusted third party easily becomes a bottleneck and overwhelmed by the communications, if it has to be involved in a huge number of protocol executions. This problem can be tackled by the introduction of, so-called, *optimistic* multi-party contract signing protocols [5]. The idea is that involvement of the trusted third party is only required if something goes wrong, e.g. if one of the parties tries to cheat or if a non-recoverable network error occurs. If all parties and the communication network behave correctly, which can be considered the *optimistic* case, the protocol terminates successfully without intervention of the trusted third party.

MPCS protocols are supposed to satisfy three properties: fairness, abuse-freeness and timeliness. *Fairness* means that each signer who sends out his signature has a means to receive all the other signers' signatures.

Abuse-freeness guarantees that no signer can prove to an outside observer that he is able to determine the result of the protocol. *Timeliness* ensures that each signer has the capability to end infinite waiting.

Several optimistic contract signing protocols have been proposed, most of which only focus on the special case of two parties [6, 17]. In 1999, Garay and Mackenzie proposed the first optimistic contract signing protocol [18] for multiple parties, which we call the GM protocol. Chadha, Kremer and Scedrov found a flaw in the GM protocol for $n \geq 4$, where n is the number of signers. They revised the GM protocol by modifying one of its sub-protocols and proposed the new protocol [9] in 2004 (which we call the CKS protocol). Mukhamedov and Ryan later showed that the CKS protocol fails to satisfy the fairness property for $n \geq 5$ by giving a so-called *abort-chaining attack*. They proposed a fixed protocol [24] in 2008 based on the CKS protocol. Mukhamedov and Ryan proved that their protocol satisfies fairness and claimed that it satisfies abuse-freeness and timeliness as well. They also gave a formal analysis of fairness in the NuSMV model checker for five signers.

Using the notion of abort-chaining attacks, Mauw, Radomirović and Torabi Dashti analysed the message complexity of MPCS protocols [23]. Their results made it feasible to construct MPCS protocols excluding abort-chaining attacks but with minimal messages, which we call the MRT protocols, based on so-called *signing sequences*. They also gave an example protocol with three signers. However, they only justified the correctness of the protocol at a conceptual level.

Our contributions. In the current paper, we follow the approach of Chadha, Kremer and Scedrov [9] to model check the MRT protocols,[1] in Mocha [3]. Mocha can be used to verify properties specified in alternating-time temporal logic (ATL) [4]. This allows us to have a precise and natural formulation of desired properties of contract signing, e.g., fairness can be specified as the existence of a user's *strategy* to obtain other parties' signatures once his signature is obtained by others, as model checking of ATL requires the computation of winning strategies.

We clarify how to construct an MRT protocol from a minimal signing sequence, according to [23]. In particular, we discover a fairness attack on the published MRT protocol with three signers [23] and a general abuse-freeness attack on MRT protocols.[2] For both attacks, we present solutions. As a consequence, the methodology to construct an MRT protocol is revised. Following the revised methodology, we design a number of MRT protocols for three and four signers, and all of them have been successfully model checked in Mocha.

Structure of the paper. The rest of the paper is organised as follows. Sect. 2 presents the basic assumptions and notions that are used for description of MPCS protocols. Sect. 3 briefly introduces concurrent game structures, the temporal logic ATL and the model checker Mocha. This section also shows how to build MPCS protocol models in Mocha and how to express their desired properties in ATL. In Sect. 4, we recall the design methodology of MRT protocols as discussed in [23], and describe the attacks found on MRT protocols with proposed solutions. In Sect. 5, we revise the MRT design methodology, and design a number of MRT protocols all of which have been successfully model checked in Mocha. The related work on formal analysis of contract signing protocols is discussed in Sect. 6. We conclude the paper with some future research topics in Sect. 7.

2 Preliminaries

This section describes the basic structure of an optimistic contract signing protocol with its underlying assumptions. A few cryptographic primitives are employed in such protocols which we briefly introduce. We also explain the security requirements associated with MPCS protocols.

2.1 Basic notions

An optimistic MPCS protocol generally involves a group of signers $P_1, \ldots, P_n$, who want to sign a contract monitored by a trusted third party T. A signer may be *honest* and thus strictly follow the protocol, or he may be dishonest and deviate from the protocol in order to collude with other dishonest signers to get undesirable advantages over the remaining signers. The structure of a protocol consists of a main protocol and one or several sub-protocols. The main protocol is executed by signers to exchange their promises at different levels and signatures without the intervention from the trusted third party T. The sub-protocols, which usually include an abort protocol and a resolve protocol, are launched by a user on contacting T to deal with awry situations.

Once having contacted T by initiating a sub-protocol, the signers would never be allowed to proceed with the main protocol. T makes a decision on basis of the information contained in a request provided by a signer as well as all previous requests that have been sent by other participants. A request consists of the promises

[1] We also verified instances of the MR protocol [28]. In this paper, we focus on the MRT protocols.

[2] The abuse-freeness attack is not reported in [28].

that the requesting signer has received so far, serving as a clue for T to judge the signer's position in the current protocol execution. On making a decision, T presumes that all the signers are honest, unless the received requests contradict, showing that someone has lied. A reply from T can be either an abort confirmation or a contract signed by all the participants. After T has sent an abort reply, she may later overturn that abort and reply with a signed contract to subsequent requests if T detects that all the signers who have previously contacted T are dishonest.[3] However, once T has sent a signed contract, she will have to stick to that decision for all subsequent requests. Without launching a sub-protocol, a signer P_i quits a protocol if he simply follows the main protocol till the end. Otherwise, P_i quits the protocol once a reply from T is received.

An important assumption of optimistic contract signing protocols is that all communication channels between the signers and the trusted third party are *resilient*, which means that messages sent over the channels are guaranteed to be delivered *eventually*.

2.2 Cryptographic primitives

An optimistic MPCS protocol usually employs zero-knowledge cryptographic primitives, i.e., *private contract signatures* (PCS) [18]. Informally, a PCS is a "semi-signature", which serves as a promise from a signer, showing his commitment to sign a contract at later stages of a protocol. In case of dishonesty, a trusted third party T is able to convert the PCS into a "true" signature, should it be necessary to guarantee fairness. As a protocol continues with multiple rounds, levels of PCS with incremental degrees of commitment are employed in a protocol. We write $PCS_{P_i}((c,\tau),P_j,T)$ for a promise made by P_i to P_j ($i \neq j$) on contract c at level τ, where τ indicates the current level of a protocol execution where P_i makes the promise. A promise is assumed to have the following properties.

- $PCS_{P_i}((c,\tau),P_j,T)$ can only be generated by P_i and P_j.
- Only P_i, P_j and T can verify $PCS_{P_i}((c,\tau),P_j,T)$.
- $PCS_{P_i}((c,\tau),P_j,T)$ can be transformed into P_i's signature only by P_i and T.

Intuitively, $PCS_{P_i}((c,\tau),P_j,T)$ acts as a promise by P_i to P_j to sign the contract c at level τ. However, the properties guarantee that P_j cannot use it to prove to anyone except T that he has this promise. This is essential to achieve abuse-freeness for MPCS protocols. Since these properties sufficiently describe the purpose and use of this primitive, we will not discuss its implementation.

[3] If not all received requests are from dishonest signers, an overturn decision may impair fairness of an honest signer who has previously received an abort reply.

2.3 Desirable properties

All contract signing protocols are expected to satisfy three security properties [24], viz. fairness, abuse-freeness and timeliness.

Fairness. At the end of the protocol, either each honest signer gets all the others' signatures, or no signer gets the signatures of any other honest signer. Fairness ensures that no signer can get any valuable information without sending out his signature, and once an honest signer sends out his signature, he will eventually get all the others' signatures. An *abort chain* [24] is a sequence of abort and resolve messages to T in a particular order, such that it enforces T to return an abort reply to an honest signer who has already sent out his signature. Abort-chaining attacks are a major challenge to fairness, and this concept was instrumental to deriving the *resolve-impossibility* result for a trusted third party for a certain class of MPCS protocols [24].

Abuse-freeness. At any stage of the protocol, there does not exist a coalition of signers who are able to prove to an outside observer that they have the power to choose between aborting the protocol and getting the signature from another signer who is honest and optimistically participating in the protocol. Intuitively, a protocol not being abuse-free implies that some of the signers have an undesirable advantage over other signers, and therefore they may enforce others to compromise on a contract.

Timeliness. Each signer has a solution to prevent endless waiting at any time. That means no signer is able to force anyone else to wait forever.

3 Formal Model

In this section, we discuss how to model protocols in Mocha using a concurrent game structure, and how to express specifications for the desired properties in alternating-time temporal logic (ATL) with game semantics. We start with the introduction of concurrent game structures and ATL [4].

3.1 Concurrent game structures and ATL

A (concurrent) game structure is defined as a tuple $S = \langle \Sigma, Q, \Pi, \pi, d, \delta \rangle$ with components:

– A finite set $\Sigma = \{1, \ldots, k\}$ of players that are identified with natural numbers (i.e., $|\Sigma| = k$).
– Q is a finite set of states.
– Π is a finite set of propositions.
– $\pi : Q \to 2^{\Pi}$ is a labeling function. For each state $q \in Q$, a set $\pi(q) \subseteq \Pi$ of propositions are true.
– $d : \{1, \ldots, k\} \times Q \to \mathbb{N}^+$. $d_a(q)$ represents the number of available moves for player $a \in \{1, \ldots, k\}$ at state $q \in Q$. We identify the moves of player a at state q with the numbers $1, \ldots, d_a(q)$.
– $\delta : Q \times (\mathbb{N}^+)^{\Sigma} \to Q$ is a transition function. Define a *move vector* to be a tuple of k actions from the k distinct players. Then for each $q \in Q$ and each move vector $\langle j_1, \ldots, j_k \rangle$, $\delta(q, j_1, \ldots, j_k)$ is the state that results from q if every player $a \in \{1, \ldots, k\}$ chooses move $j_a \leq d_a(q)$.

The temporal logic ATL (Alternating-time Temporal Logic) is defined with respect to a finite set Π of propositions and a given set Σ of players. An ATL formula is one of the following:

– p for propositions $p \in \Pi$.
– $\neg\phi$ or $\phi_1 \vee \phi_2$, where ϕ, ϕ_1, and ϕ_2 are ATL formulas.
– $\langle\langle A \rangle\rangle \bigcirc \phi$, $\langle\langle A \rangle\rangle \Box \phi$, or $\langle\langle A \rangle\rangle \phi_1 \mathcal{U} \phi_2$, where A$\subseteq \Sigma$ is a set of players, and ϕ, ϕ_1 and ϕ_2 are ATL formulas.

We interpret ATL formulas over the states of a concurrent game structure S that has the same propositions and players. The labeling of the states of S with propositions is used to evaluate the atomic formulas of ATL. The logical connectives $\neg$ and $\vee$ have the standard meaning.

In order to give the definition of the semantics of ATL, we first give the notion of strategies. Consider a game structure $S = \langle \Sigma, Q, \Pi, \pi, d, \delta \rangle$. A *strategy* for player $a \in \Sigma$ is a mapping $f_a : Q^+ \to \mathbb{N}$ such that λ is a non-empty finite state sequence and $f_a(\lambda) \leq d_a(last(\lambda))$ where $last(\lambda)$ is the last state in λ. A strategy f_a can be used to represent a set of (infinite) computations that player a may enforce. Formally, an infinite computation λ is *enforceable* by f_a if for all prefixes $\lambda' \cdot q$ (of length at least two) of λ, there exists a move vector $\langle j_1, \ldots, j_a, \ldots, j_k \rangle$, s.t., $q = \delta(last(\lambda'), j_1, \ldots, j_a, \ldots, j_k)$ and $f_a(\lambda') = j_a$. Hence, $F_A = \{f_a \mid a \in A\}$ induces a set of computations that all the players in A can cooperatively enforce, by taking the intersection of all sets of computations that are enforceable by each f_a. Given a state $q \in Q$, $out(q, F_A)$ is the set of computations starting from q that are enforceable by the set of players A applying strategies in F_A. Write $\lambda[i]$ for the i-th state in the sequence λ starting from 0.

We are now ready to give the semantics of ATL. We write $S, q \models \phi$ to indicate that the state q satisfies the formula ϕ in the structure S. And if S is clear from the context we can omit S and write $q \models \phi$. The satisfaction relation $\models$ is defined for all states q of S inductively as follows:

– $q \models p$, for propositions $p \in \Pi$, iff $p \in \pi(q)$.
– $q \models \neg\phi$ iff $q \not\models \phi$.
– $q \models \phi_1 \vee \phi_2$ iff $q \models \phi_1$ or $q \models \phi_2$.
– $q \models \langle\langle A \rangle\rangle \bigcirc \phi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models \phi$.
– $q \models \langle\langle A \rangle\rangle \Box \phi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$ and for all positions $i \geq 0$, we have $\lambda[i] \models \phi$.
– $q \models \langle\langle A \rangle\rangle \phi_1 \mathcal{U} \phi_2$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, there exists a position $i \geq 0$ such that $\lambda[i] \models \phi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models \phi_1$.

Note that $\Diamond\phi$ can be defined as $true\,\mathcal{U}\phi$. The logic ATL generalises Computation Tree Logic (CTL) [13] on game structures, in that the path quantifiers of ATL are more general: the existential path quantifier $\exists$ of CTL corresponds to $\langle\langle \Sigma \rangle\rangle$, and the universal path quantifier $\forall$ of CTL corresponds to $\langle\langle \emptyset \rangle\rangle$. To this point, for the sake of readability, we use $\forall$ ($\exists$) instead of $\langle\langle \emptyset \rangle\rangle$ ($\langle\langle \Sigma \rangle\rangle$) to denote quantification over all paths (some path) starting from a state. (E.g., $\forall \Box p$ is a syntactic sugar, which is semantically equivalent to $\langle\langle \emptyset \rangle\rangle \Box p$, but more readable.)

3.2 Modelling Methodology for MPCS protocols in Mocha

Mocha [3] is an interactive verification environment for the modular and hierarchical verification of heterogeneous systems. Its model framework is in the form of reactive modules [2]. The states of a reactive module are determined by variables and are changed in a sequence of rounds. Mocha can check ATL formulas, which express properties naturally as winning strategies with game semantics. This is the main reason we choose Mocha as our model checker in this work.

Mocha provides a guarded command language to model the protocols, which uses the concurrent game structures as its formal semantics. The syntax and semantics of this language can be found in [3]. Intuitively, each player $a \in \Sigma$ conducts a set of guarded commands in the form of $guard_\xi \to update_\xi$. The update step is executed by each player choosing one of its commands whose boolean guard evaluates to true. The next state combines the outcomes of the guarded commands chosen by the players.

We now describe how to model MPCS protocols in detail, following [9]. Each participant is modelled as a player[4] using the above introduced guarded command language. Different from other security protocols, the security of MPCS protocols is threatened by dishonest participants rather than an external intruder. In order to model that a player could be either honest or malicious, for each player P_i we build a process PiH, which honestly follows the steps of his role in the protocol, and another process Pi, which is allowed to cheat. An honest signer only sends out a message when the required messages according to the protocol are received, i.e., he faithfully follows the protocol all the time. A dishonest signer may send out a message if he gets enough information for generating the message. He can even send out messages when he is supposed to stop. The trusted third party T is modelled to be honest throughout the time. (More details about how to model MRT protocols in Mocha in given in Sect. 4.2.)

3.3 Expressing properties of MPCS protocols in ATL

We formalise both fairness and timeliness as in [9]. For signers P_i and P_j, a variable Pi_Sj represents that P_i has got P_j's signature. For each P_i, a variable Pi_stop models whether signer P_i has quit the protocol.

Timeliness. At any time, every signer has a strategy to prevent endless waiting. Signer P_i's timeliness is expressed as:

$$timelinessPi \equiv \forall\Box\,(\langle\langle PiH\rangle\rangle\Diamond\, Pi\_stop).$$

where P_i_stop represents that P_i has quit the protocol.

Fairness. A protocol is fair for signer P_i can be expressed by: if any signer obtains P_i's signature, then P_i has a strategy to get all the others' signatures. In ATL, it can be formalised as follows:

$$fairnessPi \equiv \forall\Box\,((\bigvee_{1\leq j\neq i\leq n} Pj\_Si) \Rightarrow \langle\langle PiH\rangle\rangle\Diamond\,(\bigwedge_{1\leq j\neq i\leq n} Pi\_Sj))$$

where $P_i\_S_j$ represents that P_i has received P_j's signature.

Chadha, Kremer and Scedrov also gave an invariant formulation of fairness for P_i as follows:

$$invfairnessPi \equiv \forall\Box\,(Pi\_stop \Rightarrow ((\bigvee_{1\leq j\neq i\leq n} Pj\_Si) \Rightarrow (\bigwedge_{1\leq j\neq i\leq n} Pi\_Sj))).$$

They have proved that if a contract signing protocol interpreted as a concurrent game structure satisfies $timelinessPi$ for P_i then the protocol satisfies $fairnessPi$ iff it satisfies $invfairnessPi$ [9, Thm. 3].[5]

Abuse-freeness. The formalisation of abuse-freeness in ATL is more involved. Recall that abuse-freeness means at any stage of the protocol, any set of signers are unable to *prove* to an outside observer that they have the power to choose between aborting the protocol and a fully signed contract. Depending on protocols, to find abuse-freeness attacks, we need to show that it is possible for the protocol to reach a stage of the protocol where a coalition of signers, e.g., P_i and P_j, have a strategy either to end the protocol with an abort result or to end the protocol with the result in which the coalition of signers gets other signer's signature, e.g., the signature of P_k. This can be formalised as follows:

$$\begin{aligned}&\exists\Diamond\,(\textit{an identified stage} \wedge\\ &\quad\langle\langle P_i, P_j\rangle\rangle\Box(\neg P_k\_S_i \vee \neg P_k\_S_j) \wedge\\ &\quad\langle\langle P_i, P_j\rangle\rangle\Box(P_k\_stop \rightarrow (P_i\_S_k \wedge P_j\_S_k))\end{aligned}$$

In addition, we also need to show that at the identified stage of the protocol, the coalition of signers (P_i and P_j) prove to an outside observer that P_k commits to sign the contract. Normally, the proof might be some evidence signed by the TTP.

4 Analyses of Original MRT Protocols

The work of Mauw, Radomirović and Torabi Dashti [23] aims to define a class of optimistic MPCS protocols excluding abort-chaining attacks. Their main contribution is to derive the lower-bound message complexity of such protocols from a long standing open problem in number theory — the minimal length of a (number) sequence containing all permutations of its elements as subsequences, to ensure fairness. We give a brief description of their work, and illustrate a fairness attack and an abuse-freeness attack on the example protocol [23, Sect. 7] for three signers. The fairness attack is due to the fact that the example protocol does not faithfully follow the design methodology of MRT which is not explicitly given in [23]. The abuse-freeness attack is due to a design flaw in the abort sub-protocol, thus it applies to all MRT protocols.

4.1 Design methodology of MRT protocols

Similar to other MPCS protocols (see Sect. 2), an MRT protocol consists of a `main` protocol and a resolve sub-protocol. In the main protocol signers exchange promises

[4] In Mocha, a player is modelled as an interactive module, and a protocol is modelled as a set of interactive modules

[5] Due to this result, for MRT protocols with four signers we verify *invfairnessPi* instead of *fairnessPi* on their Mocha models after we have successfully checked timeliness for P_i.

of different levels, followed by a last round of signature exchange. A signer can abort or resolve by launching the resolve sub-protocol. In the following we briefly introduce the design methodology of MRT protocols.

Signers are modelled as a finite set of numbers $\Gamma = \{1, 2, \dots n\}$. A main protocol can be expressed as a numeral string $\alpha \in \Gamma^*$, so called a *signing sequence*, regarded as the list of the indices of the signers in which order they send out their messages. A signing sequence for n signers can be divided into three phases. In the *initial phase*, the first $n-1$ signers send out their first level promises according to the first $n-1$ distinct elements in the sequence. The *middle phase* is initiated by a first level promise of the signer who was missed out in the initial phase, followed by a sequence of numbers indicating the particular order of further promise exchanges. In the *end phase* the signers exchange their signatures. A typical signing sequence for $n = 5$ is of the following form. The symbol '|' is used to separate different phases.

1234 | 543212345432 | 12345123

From the example one may easily observe that the end phase needs to be at least of length $2n-2$, in that the first n numbers, as a permutation, are for all the signers to send out their signatures, and the remaining $n-2$ messages are necessary to further distribute the signatures. The last receiver is implicit in a sequence but can be uniquely determined, e.g., signer P_4 in the above example.

We write α_i for the i-th member in a signing sequence α. Explicitly, a signing sequence can be uniquely interpreted as a protocol in the following way.

1. In the i-th step signer P_{α_i} sends out message m_i ($1 \le i \le |\alpha|$) to another signer.
2. If it is P_{α_i}'s τ-th time appearing in a complete signing sequence, then P_{α_i}'s promise level in m_i is exactly τ. (Signers may have different promise levels at the same point of time.)
3. Signatures are sent only in the end phase. The first signature is sent by the first signer appearing in the end phase.
4. The receiver of m_i is the sender of m_{i+1}, where $1 \le i < |\alpha|$.
5. The receiver of each message is allowed to have the most recent promises or signatures of all the other signers, provided that they have sent out promises or signatures before. That is, a signer is supposed to transfer all the necessary promises to his receiver. Therefore, a sender may need to forward up to $n-2$ promises of other signers besides his own promise.

To design the content of messages, we must note that, an MRT protocol is executed in a linear way, which means, at each point of time there is at most one message in traffic according to its design. Several other protocols in the literature (e.g., [9, 24]) allow a signer to send messages to multiple other signers. At each round a signer in the MRT protocols sends a message to exactly one intended receiver (instead of sending $n-1$ messages to all other signers), and such a message is supposed to let the receiver obtain the whole history up to that received message. For example, the message of signer P_i contains all his τ-th level promises (on contract c) to every other signer in the form of $prom_\tau(c, i)$, and several promises (perhaps of different levels) that he has received from others. Intuitively, $prom_\tau(c, i)$ is an encrypted message consisting of all the $PCS_{P_i}((c, \tau), P_j, T)$ where $j \neq i$. By forwarding promises, an MRT protocol reduces the number of communication messages.

A breach of fairness happens when an honest signer sends out his signature without an effective way to consequently receive messages from other signers. Intuitively, a history of all signers' promises of sufficient length helps to convince the trusted third party T that every other signer in that protocol run has been continuously active (in a certain sense), so that the construction of an *abort chain* is prohibited. Such a history also provides T with the promises of all the participants, to be later converted into a valid document with all signatures. A major contribution of MRT [23] is showing that a signing sequence α is free of abort-chaining attacks iff α's middle phase together with the first n elements from its end phase contains all permutations of the set Γ. Protocols generated from such sequences Therefore, finding the shortest sequence containing all permutations yields a solution to minimise the number of message exchanges in this particular class of protocols.

In the following we give a more detailed explanation of its sub-protocols.

Main protocol. The signers send out and receive messages in the order specified by a signing sequence which is generated from a shortest sequence containing all permutations as introduced before. Upon receipt of a message containing all required information, a signer P_i generates a message consisting of all the up-to-date promises and signatures and sends it to the next designated receiver.

If P_i does not receive the expected message, he may quit the protocol if he has not sent out any messages yet, or he may start the resolve protocol by sending a resolve request to T. The request is in the form of $\{resolve, i, H_i, c\}_i$, where *resolve* is a reserved keyword indicating P_i is contacting T for intervention, and H_i

is P_i's *history* including all the messages he has sent or received so far, which gives T sufficient information to judge P_i's current position in an execution. The identifier c is meant to uniquely identify this contract signing session that includes the contract text and the signing partners. P_i's request does not indicate whether P_i asks T for abort or resolve. It is T's duty to make a decision and to reply with an abort or a signed contract.

Resolve sub-protocol. For sub-protocols, an MRT protocol does not explicitly distinguish abort and resolve request, i.e., every request to the trusted third party T is a resolve. It's T's responsibility to make a decision to reply with an abort or a signed contract. It is obvious that if a signer in the initial phase sends a request to T, an abort will always be replied. However in the middle phase and end phase, T will have to make a decision based on whether all the previously requested signers have been dishonest.

The trusted third party T maintains a tuple $\langle c, status\rangle$ in her database indicating a list of signers who have contacted her so far. T also controls a variable $T\_i(c)$ for each P_i to record P_i's executing position at the moment P_i contacts T. The variable $T\_i(c)$ indicates the highest level of promise P_i has sent out. Together with the history H_i of each received request, T is able to make a decision on whether to reply with an abort or a signed contract. The reasoning patterns of T in the sub-protocols of MRT are very similar to that of other optimistic MPCS protocols (e.g. [9,24]): a signer is considered dishonest if he is shown by another signer's request to have continued in the main protocol after having contacted T. However in the MRT protocols, different signers may have different promise levels at a particular position, which are induced by the signing sequences of the main protocols. As a consequence, different signing sequences decide slightly different sub-protocols for T. Informally, after receiving a request from P_i:

- T checks if it is the first request that she has ever received. If it is, T judges from H_i whether P_i's current execution position is in the initial phase. If yes, T replies P_i with an abort and stores $\langle c, (i : H_i)\rangle$ into the database. If not, T replies P_i with a signed contract and stores $\langle c, S\rangle$;
- If the request is not the first one, T checks if she has ever sent a signed contract by checking if there exists an entry $\langle c, S\rangle$ in its database,
  - If yes, then T sticks to the decision and replies P_i with a signed contract and stores $\langle c, S\rangle$;
  - If not, that means T has replied an abort to a signer. In order to decide whether to stick to the abort or overturn it, T checks if all the signers who have received an abort reply are cheating. For each $j \in \langle c, (j : H_j)\rangle$, T checks whether the current history H_i contains a new promise of P_j, which means P_j continues the protocol after having contacted T. If yes, then T detects that P_j is dishonest. If for all $j \in \langle c, (j : H_j)\rangle$, P_j is detected to be dishonest, then T overturns its abort decision and replies P_i with a signed contract.[6] Otherwise, T sticks to her abort decision and replies P_i with an abort. Meanwhile, T stores $\langle c, (i : H_i)\rangle$.

4.2 Modelling MRT protocols in Mocha

Following the modelling methodology in Sect. 3.2, we explain our Mocha models of MRT protocol in more details as follows.

We use an integer $Pr\_i\_j\_L = \tau$ to represent that P_i has sent out his τ-th level promise to P_j. In particular, for MRT protocols, the integer $Pr\_i\_k\_j\_L = \tau$ represents that P_i has forwarded P_k's τ-th level promise to P_j.[7] An action that a player sends out a certain level of promise is modelled as a guarded command in which the sender updates the corresponding integer variable to the right level. The corresponding guard consists of the received promises and previously sent out promises. For signers P_i and P_j, a variable Pi_Sj represents that P_i has got P_j's signature. Since P_i continues to hold P_j's signature once P_i gets it, we model that once Pi_Sj is set to true its value would never be changed thereafter. In our models, when an honest signer P_i receives a signature from another player P_j, he will set the corresponding variable Pi_Sj to true. When P_i receives a signed contract from the trusted third party (as a reply to his resolve request), he will also set the corresponding variables Pi_Sjs to true. For each P_i, a boolean variable $Pi\_contacted\_T$ models whether signer P_i has sent out a resolve request to T. Another set of boolean variables, in the form of $Pi\_request\_\tau_1\_\ldots\_\tau_{i-1}\_\tau_{i+1}\ldots\_\tau_n$, are used to indicated that he has received promises of levels $\tau_1, \ldots \tau_{i-1}, \tau_{i+1}, \ldots, \tau_n$ from the other signers.[8] For instance, the boolean variable $P2\_request\_1\_0$ indicates that the signer P_2 has received P_1's 1-st level promise

[6] Note that if there exists $j \in \langle c, (j : H_j)\rangle$ such that P_j is honest, overturning an abort decision may cause the current run unfair to P_j, which is an assumption that contributes to *abort-chaining attack* and *resolve-impossibility* [24].

[7] MRT protocols reduce message complexities by allowing signers to forward other signers' signatures. A detailed description of MRT protocols can be found in Sect. 4.

[8] Since there are only few points in an MRT protocol for a signer to send a resolve request, the number of such boolean variables are small (see examples of MRT protocols in Sect. 5).

but no promise from P_3 in a three-party MRT protocol. When a signer contacts T for a recovery, he will also send these information to T. Based on these, T will decide whether to abort or reply P_i with a signed contract. These variables are false by default and set to true when a resolve request is sent out by the signer while he must have the corresponding levels of promises from the other signers as the guard. For each P_i, a variable Pi_stop models whether signer P_i has quit the protocol. Since $\neg Pi\_stop$ is one of conditions within each P_i's guarded command, P_i would never change any of its variables once Pi_stop is set to true. If a signer P_i has not received the first expected messages during the initial phase[9] form other signers, he can quit the protocol by setting Pi_stop to true. If he has received all necessary signatures or he has received an abort token or a signed contract from T, he finishes the protocol by setting Pi_stop to true as well. (For dishonest signers, they might not stop at this stage.) There is an additional idle action for the signers meaning that they can wait for some promises while doing nothing.

In our Mocha models, the guard for honest PiH consists of all the conditions strictly according to the protocol. While for the dishonest Pi, the guard just consists of only the messages needed to generate a sending message. We treat the robustness of the cryptographic primitives as a basic assumption which are not explicitly modelled for our protocol. All Mocha models can be found at [29].

4.3 A fairness attack on the example protocol in [23]

An MRT protocol with three signers was proposed in [23] as an illustrating example. It is based on the signing sequence 12 | 3123 | 1232 (see Fig. 1). The message $pr_\ell(c,i)$ in the illustration is an abbreviation of $prom_\ell(c,i)$, and $s(c,i)$ represents P_i's signature and last level promise on contract c.

Our analysis in Mocha reveals an abort-chaining attack in the example MRT protocol with three signers in [23]. This is due to the fact that the protocol does not strictly follow the methodology as described in Sect. 4.1. Here we also present a simple fix.

The protocol with its attack scenario is depicted in Fig. 1. The abort-chaining attack is highlighted as shadowed rectangles. In this scenario, P_1 and P_3 are dishonest and collude to obtain P_2's signature. The attack is achieved as follows, where $prom_\tau(c,i)$ denotes the τ-th level promise of P_i on contract c:

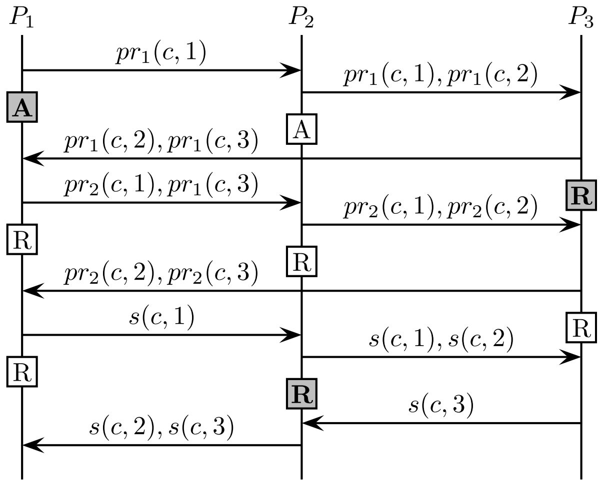

Fig. 1 The example MRT protocol described in [23].

- P_1 sends his first message out, and then contacts T with $H_1 = \{prom_1(c,1)\}$, by which T presumes P_1 is in the initial phase, and replies with an abort, at the same time storing $\langle c, (1 : \{prom_1(c,1)\})\rangle$ into her database. After having contacted T, P_1 continues in the main protocol till the end.
- P_3 contacts T at the position of the first highlighted R rectangle with

  $$H_3 = \{prom_1(c,1), prom_1(c,2), prom_1(c,3)\}.$$

  This message does not reveal that P_1 is continuing the main protocol, thus T also replies with an abort and stores

  $$\langle c, (3 : \{prom_1(c,1), prom_1(c,2), prom_1(c,3)\})\rangle$$

  into her database. After having contacted T, P_3 continues in the main protocol up to the receipt of P_2's signature $\{sig(c,2)\}$.
- P_2 faithfully follows the main protocol till the end. After sending out his signature, P_2 will never receive P_3's signature. Then P_2 contacts T with $H_2 = \{prom_1(c,1), prom_1(c,2), prom_1(c,3), prom_2(c,1), prom_2(c,2), sig(c,1), sig(c,2)\}$. On receipt of such a request, T is able to deduce that P_1 has been dishonest. However, T is unable to conclude that P_3 is cheating, because P_3's second level promise was not forwarded by P_1 according to the protocol design as shown in [23, Sect. 7].

This flaw is due to a violation of the design methodology in Sect. 4.1 (see the fifth item). In order to fix the problem, we change P_1's last message from $\{sig(c,1)\}$ into $\{sig(c,1), prom_2(c,3)\}$, i.e., P_1 is required to forward all the up-to-date promises and signatures in his hand to P_2 (see Fig. 2). With P_3's second level promise in H_2, T is able to find out that P_3 is dishonest. Therefore, T can overturn her abort decision and guarantee fairness for P_2 by sending him the fully signed contract.

[9] See more explanation about the initial phase in Sect. 4.

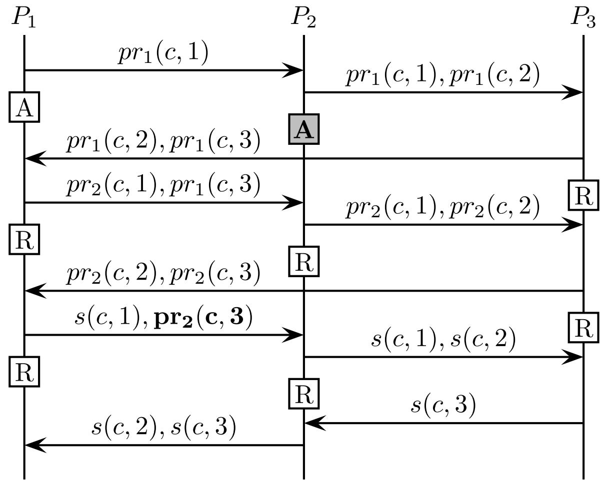

Fig. 2 The fixed MRT protocol.

4.4 An abuse-freeness attack on MRT protocols

We now describe how abuse-freeness fails in MRT protocols by the protocol described in Fig. 2 as an example (after fixing the fairness attack).

In our scenario, where P_1 is now honest and optimistic, we show that P_2 and P_3 are able to collude to get a proof that (1) P_1 is in the protocol, (2) they have a strategy to abort the protocol and (3) they have a strategy to get a fully signed contract. As shown in Fig. 2, P_1 starts the protocol by sending his first message to P_2. On receipt of P_1's first message, P_2 sends his first message to P_3, then contacts T with $H_2 = \{prom_1(c,1), prom_1(c,2)\}$, by which T presumes P_2 is in the initial phase. Thus T replies P_2 with $[Abort, c]_T$. In Fig. 2 this corresponds to the first rectangle labelled A below P_2.

At that point, if P_2 and P_3 show this reply to an outside observer, say Charlie, Charlie will be convinced that P_1 has started the protocol. That is because P_2's resolve request contains P_1's promise indicating P_1's participation in this protocol. Charlie is not able to verify $prom_1(c,1)$, but T has the ability to verify it, and P_2 enforces T to do this job for Charlie by replying $[Abort, c]_T$ for this contract c. Implicitly, c indicates the participation of P_1, which is thus proved authentic by T. On the other hand, P_2 is unable to get that reply without P_1's participation. Therefore, Charlie will be convinced that P_1 has started the protocol. Besides, P_2 and P_3 are able to decide the result of the protocol: If they want the protocol to be aborted, they can simply quit the protocol. P_1 may contact T with a resolve request but can only get an abort reply, because T has replied P_2 with an abort and she has not detected that P_2 is dishonest, so she sticks to her abort decision; if they want to get a signed contract, they can just continue the main protocol till the end.

We can express the abuse-freeness property in this particular protocol for P_3 in ATL as follows:

$$\neg\exists\Diamond\,(\,T\_Abort\_Send\_P_2 \wedge \langle\langle P_2, P_3\rangle\rangle\Box(\neg P_1\_S_2 \vee \neg P_1\_S_3) \wedge \langle\langle P_2, P_3\rangle\rangle\Box(P_1\_stop \rightarrow (P_2\_S_1 \wedge P_3\_S_1))$$

where $T\_Abort\_Send\_P_2$ represents that T has replied P_2 with an abort reply. As discussed before, this reply means that T has validated P_2's resolve request which contains some communication history, i.e., it is a proof of P_1's participation in the current run. $P_i\_S_j$ represents that P_i has received P_j's signature. $P1\_stop$ is used to prevent P_1 from idling forever (since P_1 is optimistic). Consequently, this formula represents that it is not possible to reach a point where P_2 and P_3 can collude to prove to an outside observer that:

1. P_1 is participating in the protocol: $T\_Abort\_Send\_P_2$ is true;
2. they have a strategy to end the protocol with an abort result, namely, P_1 cannot get all the signatures of others: $\neg P1\_S_2 \vee \neg P1\_S_3$;
3. they have a strategy to end the protocol (when the boolean variable P_1_stop becomes true) with a fully signed contract: $P_2\_S_1 \wedge P_3\_S_1$.

In order to model P_1 as an optimistic signer to verify the protocol in Fig. 2 for abuse-freeness, we could follow [9] and modify our original model by adding a module $Timer$, in which we define a timer for each resolve of P_1. Each time P_1 sends out a message, he sets the value of the corresponding timer to true. P_1 only contacts T when the timer is expired. Intuitively, $Timer$ enforces P_1 to wait sufficiently long before contacting T. Thus, in order to check abuse-freeness for optimistic P_3, the above formula can be re-formulated as follows:

$$\neg\exists\Diamond\,(\,T\_Abort\_Send\_P_2 \wedge \langle\langle P_2, P_3, Timer\rangle\rangle\Box(\neg P_1\_S_2 \vee \neg P_1\_S_3) \wedge \langle\langle P_2, P_3, Timer\rangle\rangle\Box(P_1\_stop \rightarrow (P_2\_S_1 \wedge P_3\_S_1))$$

where the $Timer$ in $\langle\langle P2, P3, Timer\rangle\rangle$ enforces P_1 to be optimistic.

Mocha detects the violation of abuse-freeness for our optimistic model for the protocol in Fig. 2. Such abuse-freeness flaw also exists in MRT protocols with four signers. For four-party MRT protocols, suppose P_1 is honest and optimistic, then if dishonest P_2 contacts T after he receives P_1's first message, he could also collude with P_3 and P_4 to break P_1's abuse-freeness in the similar way. Chadha, Kremer and Scedrov has detected a similar vulnerability against abuse-freeness for GM protocols [18] in [9].

Note that if signer P_1 is honest but not optimistic, such abuse-freeness attack cannot be achieved. The reason is that if P_1 is not optimistic, he could contact T after sending out his first message as permitted in the protocol, which makes P_1 get an abort reply and prevents P_2 and P_3 getting P_1's signature.

The violation of abuse-freeness in this protocol is due to the content of the resolve message. Thus, this attack generally applies to every MRT protocol. The MRT protocols do not distinguish abort and resolve requests. All requests in the MRT protocols have the same form. Thus, if P_2 contacts T with his communication history, namely, P_1's promise, and gets an abort reply from T, then this reply will indicate that T has validated P_1's promise contained in P_2's request, which serves as an evidence proving P_1's participation.

A solution to fix such a flaw is to reintroduce the distinction between the abort and resolve requests (see [24]. In that case, an abort request contains no history, thus an abort reply cannot prove P_1's participation. [10]

5 Analyses of Revised MRT Protocols

In this section we assume that the MRT design methodology has been revised in response to the attacks described in the previous section. Each MRT protocol consists of a main protocol, a resolve sub-protocol and an abort sub-protocol. The design of the main protocol and resolve sub-protocol remain the same as in Sect. 4.1. We focus on the abort sub-protocol.

In revised MRT protocols, the signers at the initial phase are allowed to send abort requests to the trusted third party T. After receiving an abort request from P_i, T first checks if she has ever sent out a signed contract. If not, i.e., *validated* is false, T adds i into $S(c)$, sends P_i an abort reply, and stores the reply. Besides, T sets $h_i(c) = 1$ and $\ell_i(c) = 0$. Otherwise, T sends a signed contract to P_i. In the following, we design and verify a number of MRT protocols for 3 and 4 signers.

MRT protocols with three signers. To design MRT protocols with three signers, first we need to find the shortest sequences containing all permutations of the set $\{1, 2, 3\}$ as a subsequence. There exist 7 distinct shortest sequences (modulo isomorphism) which contain all permutations of $\{1, 2, 3\}$ and they are presented below. The sequence to the left of the '|' symbol is the middle phase, and the right sequence is the partial end phase. We design MRT protocols with three signers using these sequences.

❶ 3123 | 123 (the instance sequence in [23])
❷ 3121 | 321
❸ 3123 | 132
❹ 31323 | 13
❺ 31321 | 31
❻ 3123 | 213
❼ 3121 | 312

For sequence ❷, first we complete the end phase by appending a 2 in the end, then add the initial phase 12 at the beginning, and get a complete signing sequence 12 | 3121 | 3212. After that, we design the contents of communication messages. First P_1 sends his 1-st level promise to P_2. P_2 generates a message containing his 1-st level promise and P_1's 1-st level promise to P_3. P_3 generates a message containing his 1-st level promise and P_2's 1-st level promise to P_1. Then P_1 appears in the sequence again, so P_1 sends his 2-nd level promise as well as P_3's 1-st level promise to P_2. Everyone sends his signature in the end phase. As the number of each signer's appearances in the final signing sequence are different, the promise levels of each signer are also different. In the end phase, P_3 is the first one who sends his signature out, and his signature is with his 3-rd level promise. While for P_1, his signature is with his 4-th level promise. The designed protocol is specified in Fig. 3. The rectangles represent the points where a signer can send a request to contact T. A signer can contact T at such a point to complain that he has sent his message out but has not received the expected message. The labels in the rectangles represent what a signer would expect from T's reply when sending a request from that point. Label A denotes T replies with an abort, and label R denotes T may reply with an abort or a signed contract. T makes her reply decision at R points according to her knowledge about signers. Whether the points are labelled to A or R depends on the position of the request. If the request position is in the initial phase, then the points are labelled as A. Otherwise, the points are labelled as R. For example, the first rectangle with a label A denotes that: P_1 has sent out the first message $\{pr_1(c, 1)\}$ to T, indicating that he has not received the expected incoming message $\{pr_1(c, 2), pr_1(c, 3)\}$. On receipt of that resolve request, T checks her database and the request content. If no inconsistencies are detected, T replies P_1 with an abort.

For sequence ❸, we can get the corresponding signing sequence 12 | 3123 | 1321. For this sequence, each signer commits the same level of promises (3-rd level)

[10] This even helps in subsequent unsuccessful resolve requests. Suppose P_2 aborts and then P_3 resolves and gets an abort reply from T. Since all the abort replies are of the same form $[Abort, c]_T$, the second abort token does not make it more convincing to an outside party.

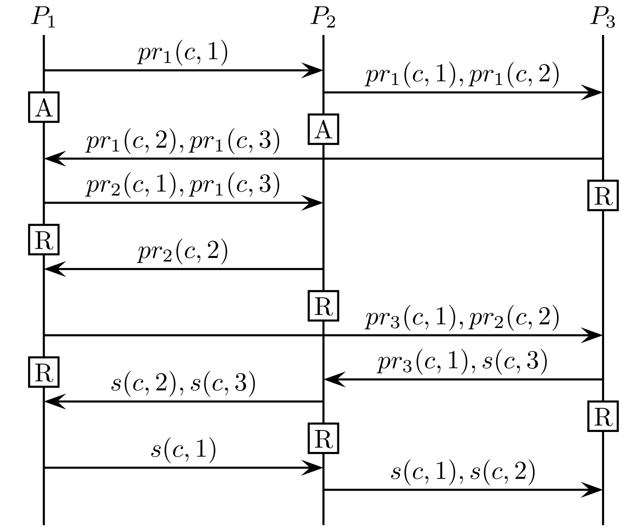

Fig. 3 A 3-party MRT protocol for sequence ❷

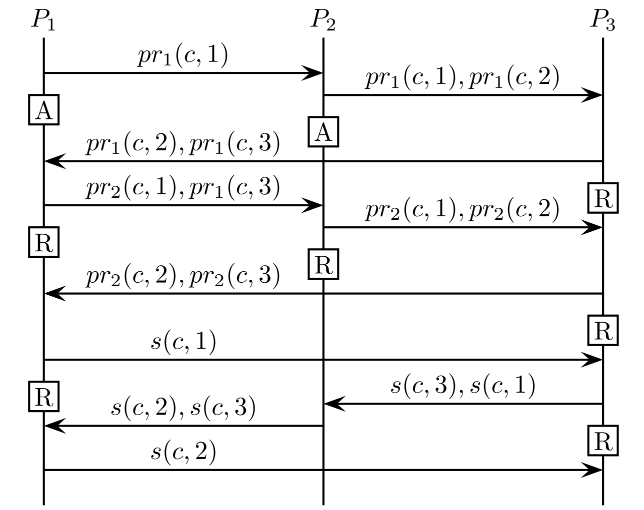

Fig. 4 A 3-party MRT protocol for sequence ❸

upon reaching the end phase. The designed protocol is specified in Fig. 4.

For sequence ❹, we can get the corresponding signing sequence 12 | 31323 | 1321. In order to complete the final signature distribution, we need to add two more messages in the end phase. This is an example showing that a shortest sequence containing all permutations does not necessarily give rise to a protocol with minimal messages. Fig. 5 is the illustration of the designed protocol.

The completed signing sequence for sequence ❺ is 12 | 31321 | 3123. This sequence also requires appending two numbers in the end phase for completing the final signing sequence. Fig. 6 is the illustration of the designed protocol.

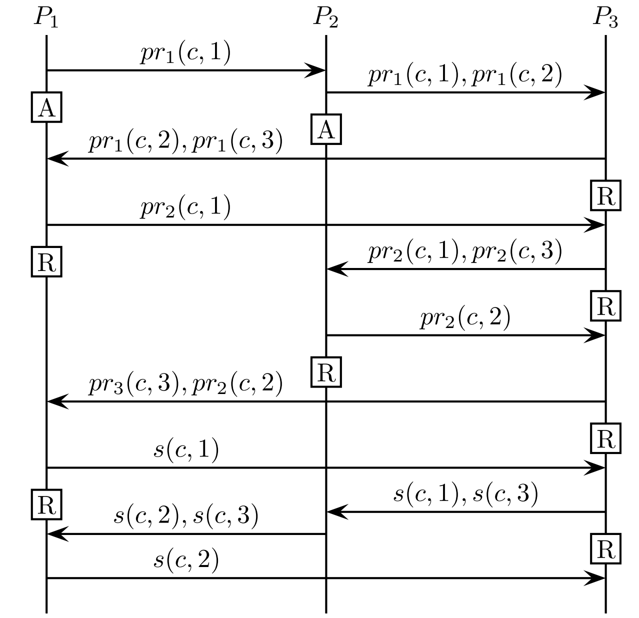

Fig. 5 A 3-party MRT protocol for sequence ❹

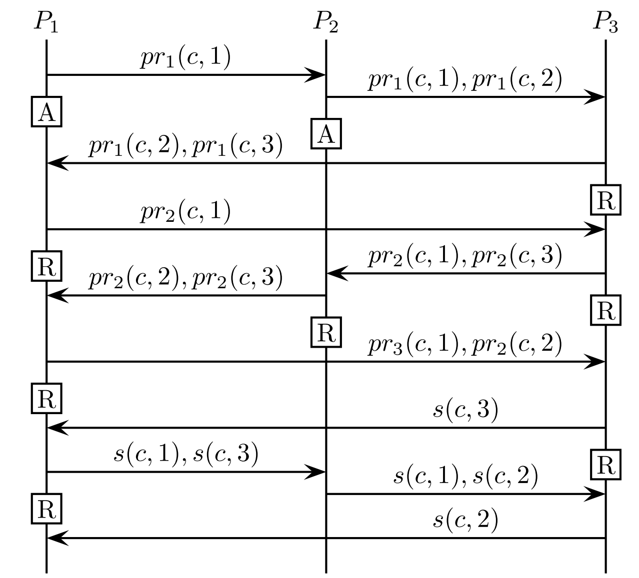

Fig. 6 A 3-party MRT protocol for sequence ❺

For sequence ❻, we get the corresponding signing sequence 12 | 3123 | 2132, in which every signer sends out his signature together with his 3-rd level promise.

The completed signing sequence for sequence ❼ is 12 | 3121 | 3123. Fig. 8 illustrates the protocol designed from the sequence. In the end phase, P_1's signature is sent with his 4-th level promise, and all the others' signatures are sent with their 3-rd level promises.

MRT protocols with four signers. For four signers, there are 9 distinct sequences modulo isomorphism that contain all permutations of $\{1, 2, 3, 4\}$:

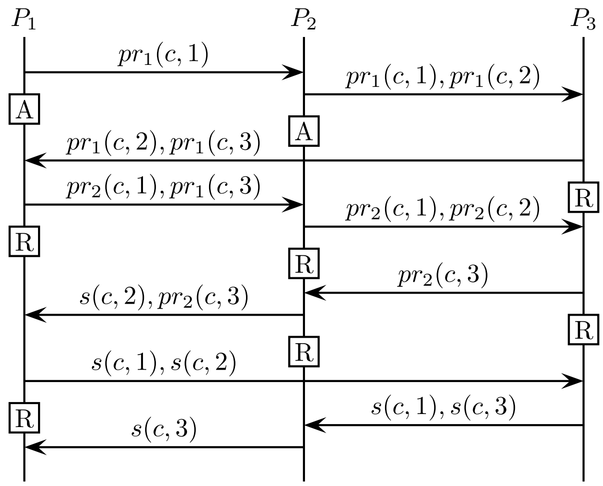

Fig. 7 A 3-party MRT protocol for sequence ❻

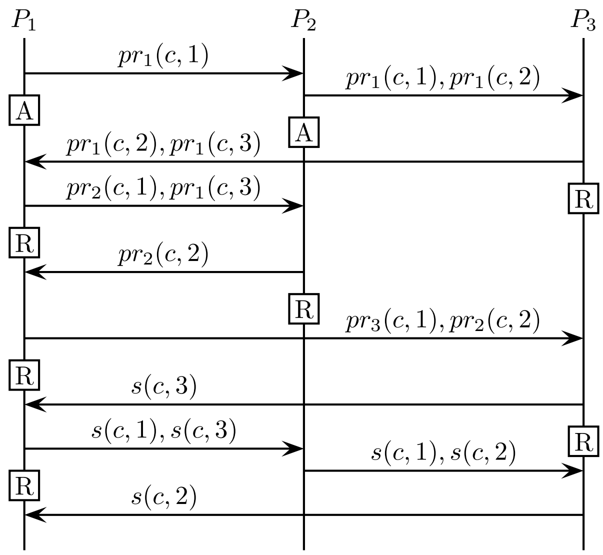

Fig. 8 A 3-party MRT protocol for sequence ❼

① 42314234 | 1243
② 42314234 | 1234
③ 42314234 | 1324
④ 42314324 | 1234
⑤ 42314324 | 1342
⑥ 42314324 | 1324
⑦ 42312432 | 1423
⑧ 42312432 | 1432
⑨ 42312432 | 1342

We take 3 typical sequences – sequence ②, ④ and ⑦ and generate protocols out of them.

For sequence ②, the completed signing sequence is 123 | 42314234 | 123432. When designing the message contents, we notice that as the number of signers increases, the promises levels contained in one message become more and more complex. Therefore, we must make sure that we strictly follow the methodology in Sect. 4.1 to avoid mistakes. Fig. 9 shows a designed protocol for this sequence.

The completed sequences for sequence ④ and sequence ⑦ and the corresponding protocols are as follows:

④ 42314324 | 1234 → 123 | 42314324 | 123432
⑦ 42312432 | 1423 → 123 | 42312432 | 142324

We have verified fairness and timeliness properties of the MRT protocols generated from all 7 shortest sequences for three signers. As for four signers, we verified the protocols generated from sequence ②, ④ and ⑦. All Mocha models can be found at [29].

6 Related Work

We first discuss the literature that are most related to our work on formal verification of MPCS protocols. Chadha, Kremer and Scedrov [9] used Mocha to check properties (fairness, timeliness and abuse-freeness) of the GM protocol and discovered a problem with fairness in the case of four signers. To fix this problem, they revised the GM protocol by modifying one of its subprotocol (resulting in a protocol which we call the CKS protocol). An abuse-freeness vulnerability was also found in the GM protocol for three signers. This is due to the fact that T's reply to a signer's abort or resolve request contains additional information, which can be used as a proof to an outside challenger showing the other signers' participation in the protocol. Their fix is to exclude the additional information from T's replies. The CKS protocol was successfully verified using Mocha for four signers. Mukhamedov and Ryan [24] later showed that the CKS protocol is not fair for $n \geq 5$ by giving an abort-chaining attack. By an informal argument they also showed that no resolve protocol can fix the problem. They proposed a fixed protocol (which we call the MR protocol) based on the CKS protocol (using similar abort and resolve sub-protocols). The fairness of the MR protocol has been analysed in NuSMV for 5 signers. Mukhamedov and Ryan claimed that their protocol is abuse-free because of the use of PCS. We followed the approach of Chadha, Kremer and Scedrov to use Mocha to model check the MR protocol with up to 5 signers and both fairness and timeliness properties were successfully checked [28]. In our work [28], the formulation of fairness in ATL as winning strategies is model independent, while Mukhamedov and Ryan have to split fairness into two CTL sub-properties in order to cover all possible scenarios, for which it is necessary to go through a number of cases (see [24], Sect. 7).

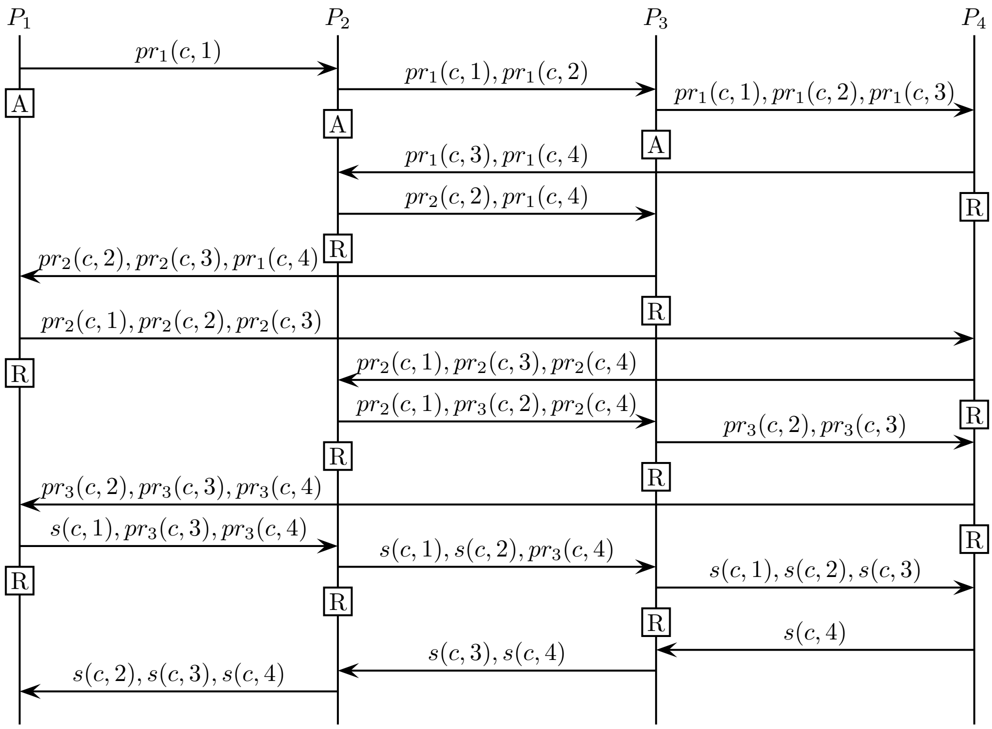

Fig. 9 A 4-party MRT protocol for sequence ② 123 | 42314234 | 123432.

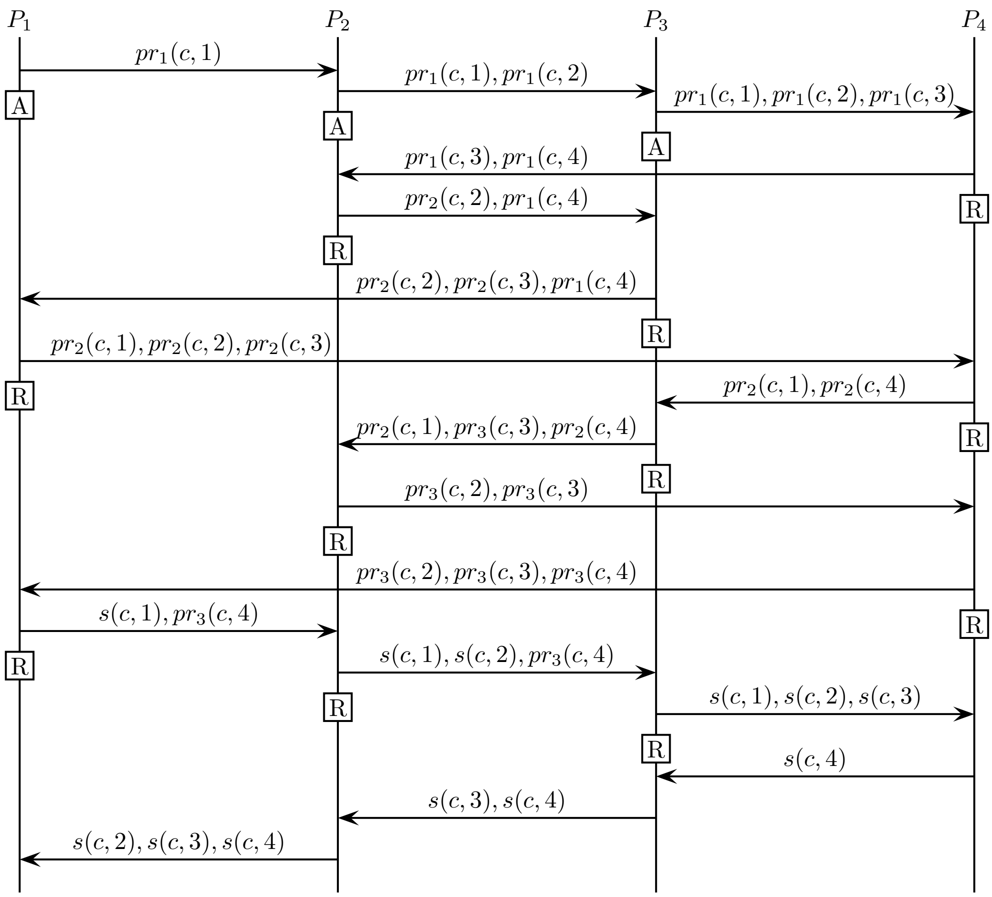

Fig. 10 A 4-party MRT protocol for sequence ④ 123 | 42314324 | 123432.

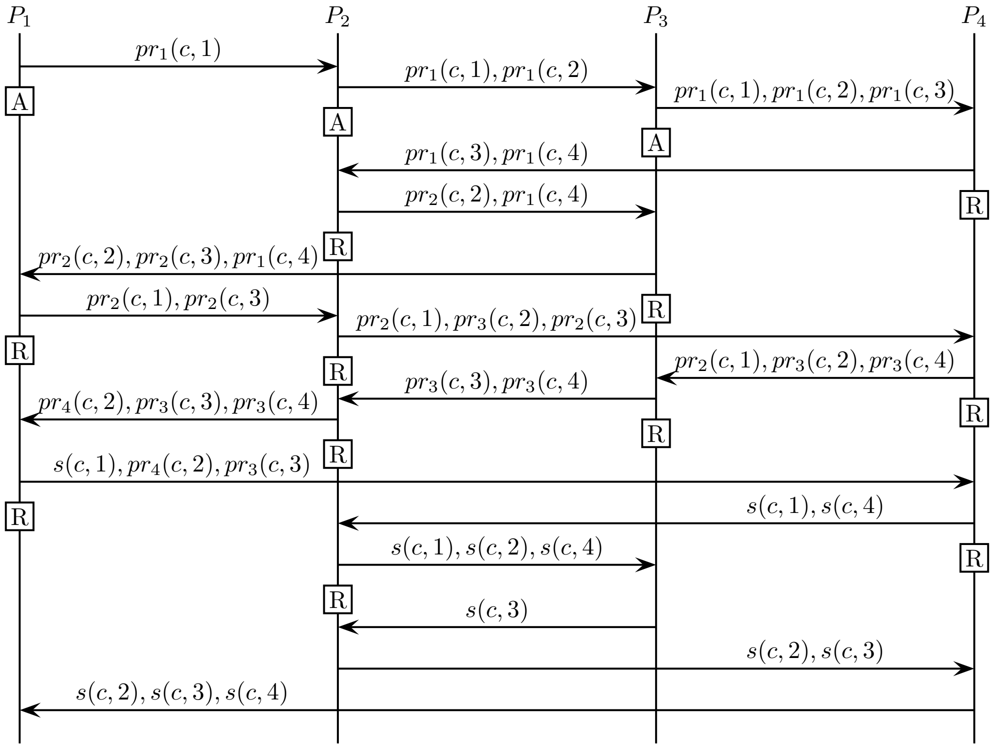

Fig. 11 A 4-party MRT protocol for sequence ⑦ 123 | 42312432 | 142324.

Two-party contract signing protocols have been previously analysed using the finite model checker Murϕ [26], an inductive method based on multiset rewriting [8], and Mocha [15] to discover errors and suggest fixes. Unlike in the two-party cases, the complexity of multi-party protocols comes from the requirement that a security requirement (e.g., fairness, timeliness and abuse-freeness) needs to be satisfied for every partipant, and in order to achieve this requirement, a trusted third party is allowed to overturn an abort decision she has previously made. Consequently, it is much more difficult to design and verify MPCS protocols. Especially it is the case of MRT protocols in which the behaviours of distinct parties are totally asymmetric (as derived from minimal message sequences), unlike GM, CKS and MR protocols in which signers have relatively symmetric behaviours.

7 Discussion and Conclusion

We have applied the model checker Mocha to verify a number of MRT protocols [23]. All the Mocha models and ATL properties used by our models are available at [29]. Mocha allows one to specify properties in ATL which is a branching-time temporal logic with game semantics, and the model checking problem for ATL requires the computation of winning strategies. The use of Mocha allows us to have a precise and natural formulation of desirable properties of contract signing. The generality of this approach is demonstrated by its use in the verification of other fair exchange protocols, e.g., [16, 20, 21].

The main result of Mauw, Radomirović and Torabi Dashti [23] made it feasible to construct fair MPCS protocols with a minimal number of messages. Their main theorem [23] states that there is a fair sequence of length $n^2 - n + 3$, where n is the number of signers in an MPCS protocol. This fair sequence contains all permutations of $\{1, \ldots, n\}$ as sub-sequences, and it can be transformed into a (linearly ordered) MPCS protocol of length $n^2 + 1$. However, the resulting MPCS protocol is only free of abort-chaining attacks, and it is merely conjectured that this implies fairness. We described how to derive an MRT protocol from a minimal signing sequence explicitly. In particular, we discovered an abort-chaining attack in the published MRT protocol with three signers [23] and an abuse-freeness attack for all MRT protocols. The first attack is due to a mistake in designing this particular protocol, which can be fixed by following MRT's design methodology. The second attack is due to a design flaw in the content of the resolve messages. A solution to fix it is by reintroducing an abort sub-protocol to treat abort requests differently. After revising the design methodology with our solution to the abuse-freeness attack, we developed a number of MRT protocols for three and four signers and successfully verified all of them in Mocha.

To demonstrate the attack on abuse-freeness using Mocha, we have to model optimistic behaviour of signers and to identify a stage in a protocol execution where a coalition of signers have a strategy to abort the protocol, and another strategy to end the protocol with the coalition getting all the other signers' signatures. In addition, we also need to show that at this particular stage the coalition can *prove* to an outsider that the other signers have committed to sign the contract. Such proofs may vary for different protocols. In general, how to formalise a generic notion of abuse-freeness in a precise and correct way is still a challenging research topic [10,14,12].

In this paper, our analyses are performed automatically using Mocha. In principle, we could also apply MCMAS [22], a model checker for verification of epistemic and ATL properties in multi-agent systems, which has been applied to security protocols, e.g., [7,27]. However, as the performance of MCMAS on ATL properties is still unclear to us, our choice to use Mocha in this work is arbitrary. We believe it will be an interesting topic to produce a translation of our Mocha models to MCMAS for a comparative study.

We have verified protocols for fairness and timeliness with a quite limited number of signers (up to four). The verification of the timeliness property in Mocha usually took minutes while for fairness properties it might need a number of days, possibly due to the asymmetry in MRT's protocol design. A possible future direction is to study ways of abstractions [19] in order to analyse models in Mocha with more signers. Using an inductive approach, e.g. [25], to prove correctness of the protocols with a more general setting is also interesting. However, such proofs might be highly nontrivial if the behaviours of signers in an MPCS protocol are not totally symmetric and especially for MRT protocols, have a non-uniform construction on the order of signers' positions.

Acknowledgement. We thank Saša Radomirović for many helpful discussions and the anonymous reviewers for their constructive comments.

References

1. Adleman, L.: Short permutation strings. Discrete Mathematics **10** (1974) 197–200
2. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods in System Design **15**(1) (1999) 7–48
3. Alur, R., Henzinger, T.A., Mang, F.Y.C., Qadeer, S., Rajamani, S.K., Tasiran, S.: Mocha: Modularity in model checking. In: Proc. 10th Conference on Computer Aided Verification. LNCS 1427., Springer (1998) 521–525
4. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5) (2002) 672–713
5. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for fair exchange. In: Proc. 4th ACM conference on Computer and Communications Security. ACM (1997) 7–17
6. Asokan, N., Shoup, V., Waidner, M.: Optmistic fair exchange of digital signatures. Selected Areas in Communications **18**(4) (2000) 591–606
7. Boureanu, I., Cohen, M., Lomuscio, A.: Automatic verification of temporal-epistemic properties of cryptographic protocols. Journal of Applied Non-Classical Logics **19**(4) (2009) 463-487
8. Chadha, R., Kanovich, M., Scedrov, A.: Inductive methods and contract-signing protocols. in: Proc. 8th ACM Conference on Computer and Communications Security. ACM (2001) 176–185
9. Chadha, R., Kremer, S., Scedrov, A.: Formal analysis of multi-party contract signing. Journal of Automated Reasoning **36**(1-2) (2006) 39–83
10. Chadha, R., Mitchell, J.C., Scedrov, A., Shmatikov, V.: Contract signing, optimism, and advantage. Journal of Logic and Algebraic Programming **64**(2) (2005) 189–218
11. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An open source tool for symbolic model checking. In: Proc. 14th Conference on Computer Aided Verification. LNCS 2404., Springer (2002) 359–364
12. Cortier, V., Küsters, R., Warinschi, B.: A cryptographic model for branching time security properties - the case of contract signing protocols. In: Proc. 12th European Symposium on Research in Computer Security. LNCS 4734., Springer (2007) 422–437
13. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science (B), MIT Press (1990) 955–1072
14. Kähler, D., Küsters, R., Wilke, T.: A Dolev-Yao-based definition of abuse-free protocols. In: Proc. 33rd Colloquium on Automata, Languages and Programming. LNCS 4052., Springer (2006) 95–106
15. Kremer, S., Raskin, J.-F.: Game analysis of abuse-free contract signing. In: Proc. 15th IEEE Computer Security Foundations Workshop. IEEE CS (2002) 206–222
16. Kremer, S., Raskin, J.-F.: A game-based verification of non-repudiation and fair exchange protocols. Journal of Computer Security **11**(3) (2003) 399–430
17. Kremer, S., Markowitch, O., Zhou, J.: An intensive survey of fair non-repudiation protocols. Computer Communications **25**(17) (2002) 1606–1621
18. Garay, J.A., MacKenzie, P.D.: Abuse-free multi-party contract signing. In: Proc. 13th Symposium on Distributed Computing. LNCS 1693., Springer (1999) 151–165
19. Henzinger, T.A., Majumdar, R., Mang, F.Y.C., Raskin, J.F.: Abstract interpretation of game properties. In: Proc. 7th Conference on Statics Analysis Symposium. LNCS 1824., Springer (2000) 220–239
20. Liu, Z., Pang, J., Zhang, C.: Extending a key-chain based certified email protocol with transparent TTP. In: Proc. 6th IEEE/IFIP Symposium on Trusted Computing and Communications. IEEE CS (2010) 630–636
21. Liu, Z., Pang, J., Zhang, C.: Verification of a key-chain based TTP transparent CEM protocol. In: Proc. 3rd Workshop on Harnessing Theories for Tool Support in Software. ENTCS 274., Elsevier (2011) pp 51–65

22. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for the verification of multi-agent systems. In: Proc. 21st Conference on Computer Aided Verification. LNCS 5643., Springer (2009) 682-688
23. Mauw, S., Radomirović, S., Torabi Dashti, M.: Minimal message complexity of asynchronous multi-party contract signing. In: Proc. 22nd IEEE Computer Security Foundations Symposium, IEEE CS (2009) 13–25
24. Mukhamedov, A., Ryan, M.D.: Fair multi-party contract signing using private contract signatures. Information and Computation **206**(2-4) (2008) 272–290
25. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6**(1-2) (1998) 85-128
26. Shmatikov, V., Mitchell, J.: Finite-state analysis of two contract signing protocols. Theoretical Computer Science **283**(2) (2002) 419–450
27. Zhang, C., Pang, J. How to work with honest but curious judges? (preliminary report). In: Proc. 7th Workshop on Security Issues in Concurrency. EPTCS 7, (2009) 31-45
28. Zhang, Y., Zhang, C., Pang, J., Mauw, S.: Game-based verification of multi-party contract signing protocols. In: Proc. 6th Workshop on Formal Aspects in Security and Trust. LNCS 5983., Springer (2009) 186–200
29. Zhang, Y., Zhang, C., Pang, J., Mauw, S.: Game-based verification of multi-party contract signing protocols – Mocha models and ATL properties (2009) Available at `http://satoss.uni.lu/members/jun/mpcs/`.