Game-Based Verification of Multi-Party Contract Signing Protocols

Ying Zhang[1,2], Chenyi Zhang[1], Jun Pang[1], and Sjouke Mauw[1]

[1] University of Luxembourg, 6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
[2] Shandong University, Jinan, 250101 China

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 Mukhamedov and Ryan (MR), and 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. We analyse the MR protocol with up to 5 signers and our analysis does not reveal any flaws. MRT protocols can be generated from minimal message sequences, depending on the number of signers. We discover an attack in a published MRT protocol with 3 signers, and present a solution for it. We also design a number of MRT protocols using minimal message sequences for 3 and 4 signers, all of which have been model checked in Mocha.

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 involve a trusted third party (T). This trusted third party simply collects the signatures of all signers and then distributes them to all parties. A major drawback of this approach is that the trusted third party easily becomes a bottleneck, since it will be involved in all communications for all contracts. This problem is addressed by the introduction of, so-called, *optimistic* multi-party contract signing protocols [1]. 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 is 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 [2, 3]. In 1999, Garay and Mackenzie proposed the first optimistic contract signing protocol [4] with 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 a fixed protocol [5] 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 [6] in 2008 based on the CKS protocol (which we call the MR 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 5 signers.

Using the notion of abort-chaining attacks, Mauw, Radomirović and Torabi Dashti analysed the message complexity of MPCS protocols [7]. 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 3 signers. However, they only provided a verification of the protocol at a conceptual level.

In this paper, we follow the approach of Chadha, Kremer and Scedrov [5] to model check the two recently proposed protocols, the MR and MRT protocols, in Mocha [8]. Mocha can be used to model check properties specified in ATL [9]. This allows us to have a precise and natural formulation of desired properties of contract signing, as the model checking problem for ATL requires the computation of winning strategies. We model the MR protocol with up to 5 signers and verify both fairness and timeliness properties, while Mukhamedov and Ryan only analysed fairness of their protocol with 5 signers.

We clarify how to construct an MRT protocol from a minimal signing sequence. According to this methodology, we design a number of MRT protocols for 3 and 4 signers, all of which have been model checked in Mocha. In particular, we discover a fairness attack on the published MRT protocol with 3 signers [7] and we present a solution to it. The fixed protocol is shown to satisfy fairness.

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 only 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 P_i 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 the main protocol any more. 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 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, *private contract signatures* (PCS) [4]. 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] Otherwise T's 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 [6], 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 any signature. Fairness ensures that no signer can get any valuable information without sending out his signature, and once a signer sends out his signature, he will eventually get all the others' signatures. An *abort chaining* [6] 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 user who has already sent out his signature. Abort-chaining attacks are a major challenge to fairness, and were instrumental to deriving the *resolve-impossibility* result for a trusted third party for a certain class of MPCS protocols [6].

Abuse-freeness. 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 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 unexpected 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 [9].

3.1 Concurrent game structures and ATL

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

- $k \in \mathbb{N}^+$ is the number of players, identified with the numbers $1, \ldots, k$.
- 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 $1, \ldots, d_a(q)$.
- δ is a transition function. 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 finite set $\Sigma = \{1, \ldots, k\}$ 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 $S = \langle k, Q, \Pi, \pi, d, \delta\rangle$. 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 $f_a(\lambda) \leq d_a(q)$ if λ's last state is q. In other words, a strategy f_a represents a set of computations that player a can enforce. Hence, $F_A = \{f_a \mid a \in A\}$ induces a set of computations that all the players in A can cooperate to enforce. Given a state $q \in Q$, $out(q, F_A)$ is the set of computations enforced 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) [10] 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 $\langle\langle\emptyset\rangle\rangle$.

3.2 Modelling MPCS protocols in Mocha

Mocha [8] is an interactive verification environment for the modular and hierarchical verification of heterogeneous systems. Its model framework is in the form

of reactive modules [11]. 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 [8]. Intuitively, each player $a \in \Sigma$ conducts a set of guarded commands in the form of $guard_\xi \rightarrow update_\xi$. 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 [5]. Each participant is modelled as a player using the above introduced guarded command language. 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 messages after he is supposed to stop. The trusted third party T is modelled to be honest throughout the time. We express the communicational messages as shared boolean variables. The variables are false by default and set to true when they are sent out by the signers.

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. For each P_i, a variable Pi_stop models whether signer P_i has quitted 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. The integer $Pr\_i\_j\_L = \tau$ represents 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. All Mocha models can be found at [12].

3.3 Expressing properties of MPCS protocols in ATL

We formalise both fairness and timeliness as in [5].

Fairness. A protocol is fair for signer P_i can be expressed as: 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:

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

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

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

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 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* [5, Thm. 3].[4]

4 Model Checking the MR Protocol

In this section, we give the description of the MR protocol [6] proposed by Mukhamedov and Ryan. We build models using a program for any number n of signers, and model check both fairness and timeliness for the models with up to 5 signers in Mocha.

4.1 Description of the MR protocol

The MR protocol is based on $PCSs$ and consists of one main protocol, one abort sub-protocol and one resolve sub-protocol.

Main protocol The main protocol consists of $\lceil n/2\rceil + 1$ rounds for n signers, and requires $n(n-1)(\lceil n/2\rceil + 1)$ messages for the optimistic execution. In each round τ ($\tau \leq \lceil n/2\rceil$), a signer P_i starts with waiting for the τ-level promises from lower signers P_j ($j < i$). After receiving all the lower signers' promises, he sends out his τ-level promise to all the higher signers P_k ($k > i$) and then waits for the promises from higher signers. On receipt of all higher signers' promises, P_i then sends out his own τ-level promise to lower signers and finishes his current round. If P_i has received the $(\lceil n/2\rceil + 1)$-th level promises and signatures from all the lower signers, he broadcasts his $(\lceil n/2\rceil + 1)$-th level promise and signature to all the other signers.

If P_i does not receive all the expected messages, he may quit the protocol, or send an abort or a resolve request to the trusted third party T, according to his current position in the main protocol. The abort request has the following form:

$$S_{P_i}((c, P_i, (P_1, \ldots, P_n), abort))$$

The resolve request is as follows:

$$S_{P_i}(\{PCS_{P_j}((m, \tau_j), P_i, T)\}_{j\in\{1,\ldots,n\}\setminus\{i\}}, S_{P_i}(m, 0))$$

where for $j > i$, τ_j is the maximal level promise received from all the signers $P_{j'}$ such that $j' > j$; for $j < i$, τ is the maximal level promise received from all the signers $P_{j'}$ such that $j' < j$.

[4] For MRT protocols with 4 signers, we verify *invfairnessPi* instead of *fairnessPi* on their Mocha models after we have successfully checked timeliness for P_i.

$$\tau_j = \begin{cases} max\{\tau \mid \forall j' > i, P_i \text{ has received } PCS_{P_{j'}((m,\tau),P_i,T)}\} & \text{if } j > i \\ max\{\tau \mid \forall j' < i, P_i \text{ has received } PCS_{P_{j'}((m,\tau),P_i,T)}\} & \text{if } j < i \end{cases}$$

Sub-protocols T maintains a boolean variable *validated* to indicate whether T has ever replied with a full signed contract. T uses a set $S(c)$ to record all the signers which have contacted T and received an abort reply. T also controls two variables $h_i(c)$ and $\ell_i(c)$ for each P_i to record P_i's executing position at the moment P_i contacts T. The variable $h_i(c)$ indicates the highest level promise P_i has sent to all the signers P_j where $j > i$, and $\ell_i(c)$ indicates the highest level promise P_i has sent to all the signers P_j where $j < i$.

Abort sub-protocol. When receiving an abort request from P_i, T first checks if she has ever sent 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.

Resolve sub-protocol. When receiving a resolve request form P_i, T checks if it is the first request she has ever received. If it is, T simply replies P_i with a signed contract and sets *validated* to true. T judges P_i's current execution position and updates $h_i(c)$ and $\ell_i(c)$ according to that position. If it is not the first request, T checks if she has ever sent a signed contract by checking if *validated* is true. (1) If yes, T sticks to the decision and replies P_i with a signed contract and updates $h_i(c)$ and $\ell_i(c)$. (2) If not, that means T has ever replied an abort to some signer. In order to make a decision on whether to stick to the abort or to overturn it, T checks if it is the case that all signers in $S(c)$ are cheating. That is, for each $j \in S(c)$, T compares τ_j from P_i's request and $h_j(c)$ and $\ell_j(c)$ from T's record to check if P_j continues the main protocol after receiving an abort reply. If all $j \in S(c)$ are dishonest, T overturns her abort decision, and replies P_i with a signed contract, at the same time updating $h_i(c)$ and $\ell_i(c)$. Otherwise, T sticks to her abort reply to P_i and updates $h_i(c)$ and $\ell_i(c)$.

4.2 Automatic analysis

We now give the analysis results of the MR protocol. We have verified fairness and timeliness of the MR protocol with 2, 3 and 4 signers, and our analysis did not reveal any flaw.

In our analysis of the MR protocol with 5 signers, timeliness can be checked in Mocha. While for fairness, it seems infeasible for Mocha to verify. So instead of building one entire model covering all possible behaviours, we built a number of specific models, each of which focuses on one certain possible abort-chaining attack scenario. For instance, in order to check if there is an abort-chaining attack in which P_1 aborts first and cooperates with P_2, P_3 and P_4 to get honest P_5's signature, we built a model where only P_5 is honest, P_1 will abort firstly and the other dishonest signers only resolve rather than abort. In this way, we reduce the size of the state space significantly when checking fairness.[5]

[5] In the future, we want to apply this technique to even larger protocol instances.

We explain that the above checks can cover all possible abort-chaining scenarios (in case of 5 signers). As mentioned in Sect. 2.3, an abort-chaining attack is achieved by collaborative malefactors to enforce an aborting outcome after they obtain the honest signer's signature. We use $M_i\,(i \in \mathbb{N})$ to indicate a set consisting of dishonest signers. Intuitively, if $M_1 \subseteq M_2$, and M_2 is not able to achieve an abort-chaining attack, then neither is M_1. So for the MR protocol with 5 signers, we only need to check scenarios where one of the signers is the victim, and all the other 4 signers are dishonest and collude to cheat. If there does not exist an abort-chaining attack for such scenario, then there does not exist an abort-chaining attack for a scenario with fewer dishonest signers. Since each abort-chaining attack starts with some dishonest signer contacting T with an abort reply, and ends up with the victim signer sending out his signature, we choose one signer to abort in our model, and choose another signer to be the victim from the last 4 signers. For the MR protocol with 5 signers, only signers P_1, P_2, P_3 and P_4 have the possibility to abort. We use $iAjH(i \in [1,4], j \in [1,5], i \neq j)$ to indicate a model in which P_i aborts and P_j is the victim. So, in total we get 16 possible attack scenarios to check. Ultimately, the analysis result shows that no abort-chaining attack is detected for the MR protocol with 5 signers.

If no one aborts, an honest signer can always get other signers' signature by simply sending a resolve request to T. This means our analysis of fairness in Mocha is exhaustive. A formal correctness argument of our reasoning is postponed for future research.

Mukhamedov and Ryan have shown that the CKS protocol [5] fails to satisfy the property of fairness with $n \geq 5$. They propose a new protocol [6] and give its formal analysis in the model checker NuSMV [13] for 5 signers. They split fairness into two sub-properties in order to cover all possible scenarios, for which it is necessary to go through a number of cases. ATL has the advantage to express fairness in terms of strategies, so that our fairness specifications turn out more natural than are definable in CTL [10]. Comparing to Mukhamedov and Ryan's work, we reduce the verification problem of fairness on the system model level instead of decomposing specifications.

5 Model Checking MRT Protocols

The main result of Mauw, Radomirović and Torabi Dashti [7] makes it feasible to construct MPCS protocols excluding abort-chaining attacks with minimal communication messages. We first describe a methodology for designing an MRT protocol in Sect. 5.1. The description of the message order in the derived protocol is fully determined as in [7]. However, since Mauw, Radomirović and Torabi Dashti only gave a high-level description of the message contents, we make the underlying assumptions precise. In Sect. 5.2, we design a family of MRT protocols and give their analysis in Mocha. Our model checking results reveal an abort-chaining attack on an example protocol with 3 signers described in [7, Sect. 7], for which we propose a fix based on an important assumption that was not made explicit in the original paper.

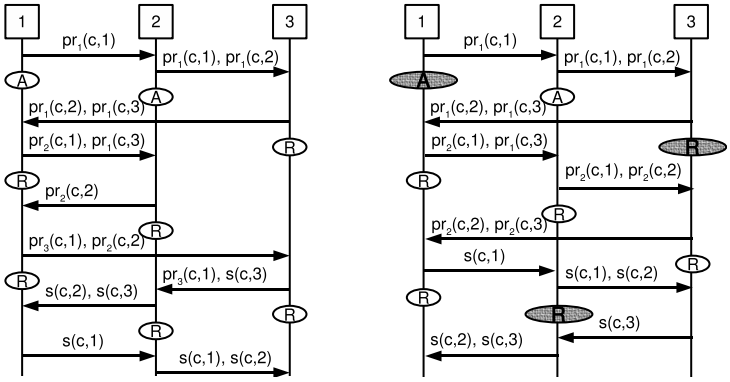

Fig. 1. MRT protocols with 3 signers (the left one is based on the signing sequence 12 | 3121 | 3212, the right one describes the protocol in [7]).

5.1 Design methodology of MRT protocols

An MRT protocol defines a sequence of messages $m_1, m_2, \ldots, m_\ell$ to be exchanged between a group of n signers in the main protocol, where every m_i is supposed to be received before m_j is sent out if $i < j$. The principles of MRT protocols are exactly those of MR, except that we have the following *additional assumptions*.

1. In each step a signer sends out message m_i $(1 \leq i \leq \ell)$ to another signer.
2. The receiver of m_i is the sender of m_{i+1}, where $i < \ell$.
3. The receiver of each message is allowed to have the most recent promises (signatures) of all the other signers, provided that they have ever sent out promises (signatures). That is, a sender may need to forward up to $n-2$ promises of other signers besides his own promise.

Based on the assumptions, an MRT protocol can be regarded as a list of the indices of the signers in which order they send their messages. Such a list is called a *signing sequence*.

A signing sequence α for an MRT protocol with signers $P_1, \ldots, P_n$ can be divided into three phases. In the *initial phase*, the first $n-1$ signers send out their promises according to the first $n-1$ distinct elements of α. 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.

$$1234 \mid 543212345432 \mid 12345123$$

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.

An MRT protocol does not explicitly distinguish abort and resolve, i.e., every request to the trusted third party T is a resolve. 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. A major contribution of [7] is showing that a protocol generated by 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 $\{1, \ldots, n\}$. Therefore, finding the shortest sequence containing all permutations yields a solution to minimize the number of message exchanges in this particular class of protocols.

To design an MRT protocol for n signers, we first find a shortest sequence α containing all permutations of the set $\{1, \ldots, n\}$, using Adleman's algorithm [14]. This sequence serves as the middle phase and partial end phase of a signing sequence. To complete the end phase, we append more indices of the signers at the end of α such that the end phase is able to distribute all the signatures to all signers. The initial phase can be obtained simply by pre-pending a sequence of length $n-1$ to α to construct a full permutation at the beginning. There exist 7 (isomorphically) distinct shortest sequences which contain all permutations in $\{1, 2, 3\}$ and they are presented below.[6]

❶ 3123 | 123 ❷ 3121 | 321 ❸ 3123 | 132 ❹ 31323 | 13
❺ 31321 | 31 ❻ 3123 | 213 ❼ 3121 | 312

The symbol '|' is used to separate different phases in the final signing sequence. Taking sequence ❷ as an example. First we complete the end phase by appending a 2 in the end. After adding the initial phase 12 at the beginning, we get a complete signing sequence 12 | 3121 | 3212. The main protocol derived from this signing sequence is depicted in the left-hand side of Fig. 1.[7] Note that a shortest sequence containing all permutations does not necessarily give rise to a protocol with minimal messages: sequence ❹ requires appending two numbers in the end phase for completing the final signature distribution. For 4 signers, there are 9 distinct sequences modulo isomorphism:

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

Fig. 2 shows a protocol designed from sequence ②.

5.2 Design and verification of MRT protocols

In this section, we design a number of MRT protocols based on the methodology in Sect. 5.1. Each MRT protocol consists of a main protocol and a resolve sub-

[6] Sequence ❶ determines the example protocol in [7, Sect. 7].

[7] We circle the positions where a signer is allowed to send a request to T. $pr_\tau(c, i)$ and $s(c, i)$ denote P_i's τ-level promise and P_i's signature on c, respectively.

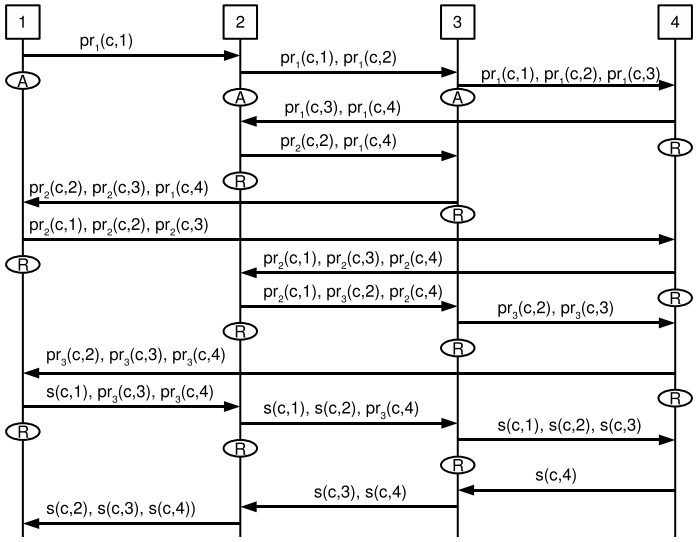

Fig. 2. An MRT protocol with 4 signers based on the sequence 123 | 42314234 | 123432.

protocol. Similar to the MR protocol, the MRT protocols assume *resilient* communication channels and private contract signatures (PCS). We have modelled and verified fairness and timeliness properties of the MRT protocols generated from all 7 shortest sequences for 3 signers. As for 4 signers, we verified the protocols generated from sequence ②, sequence ④ and sequence ⑦ as aforementioned. We briefly present our modelling of MRT protocols as follows.

Main protocol. The signers send and receive messages in the order specified by a signing sequence which is generated from a shortest sequence containing all permutation 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 $\{dispute, i, H_i, c\}_i$, where *dispute* 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, the signing partners and the contract text. P_i's request does not indicate whether P_i asks T to *abort* or *resolve*. It is T's responsibility to make a decision and to reply with an abort or a signed contract.

Resolve Sub-protocol. T maintains a tuple $\langle c, status \rangle$ in her database indicating a list of signers who have requested so far. 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 the MR protocol: a signer is considered dishonest if he is shown by another signer's request to have continued in the main protocol after having sent a request to 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, a sub-protocol of MRT has to be slightly adjusted from that of MR, and the sub-protocols may differ from each other.

5.3 An attack on the example protocol

Our analysis in Mocha reveals an abort chaining attack in the example MRT protocol with 3 signers in [7]. This is due to that the protocol does not strictly follow the methodology as described in Sect. 5.1. Here we also present a simple fix.

The protocol with its attack scenario is depicted in Fig. 1 (right). The abort-chaining attack is highlighted as shadowed circles. 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-level promise of P_i on contract c:

- 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 circle 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.
- 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 [7, Sect. 7].

The flaw of this protocol is due to a violation of assumption 3 in Sect. 5.1. 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. 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.

6 Discussion and Conclusion

In this paper, we have used the model checker Mocha to analyse two types of MPCS protocols – the MR protocol [6] and a number of MRT protocols [7].[8] 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. Thus the use of Mocha allows us to have a precise and natural formulation of desired properties of contract signing.

Mukhamedov and Ryan showed that the CKS protocol is not fair for $n \geq 5$ by giving an abort-chaining attack. The fairness of their fixed protocol [6] has been analysed in NuSMV for 5 signers. Instead, we modelled the MR protocol in Mocha with up to 5 signers and both fairness and timeliness properties have been checked. 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 [6], Sect. 7).

The main result of Mauw, Radomirović and Torabi Dashti [7] made it feasible to construct fair MPCS protocols with a minimal number of messages. Their main theorem [7] states that there is a fair signing sequence of length $n^2 - n + 3$, where n is the number of signers in an MPCS protocol. This fair sequence must contain all permutations of $\{1, \ldots, n\}$ as sub-sequences, and it can be transformed back into an 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 MR protocol from a minimal signing sequence explicitly. Following this methodology, we designed a number of MRT protocols for 3 and 4 signers, all of which have been checked in Mocha. In particular, we discovered an abort-chaining attack in the published MRT protocol with 3 signers [7]. The flaw is due to a mistake in the protocol design. We also presented a solution to it, and the fixed protocol is shown to satisfy fairness in Mocha.

Chadha, Kremer and Scedrov used Mocha to check abuse-freeness in the GM protocol and the CKS protocol, and found a vulnerability in the first protocol [5]. The vulnerability is due to the fact that T's reply to a signer's abort or resolve request contains additional information, which can be used by the signer as a proof for an outside challenger. Their fix is to exclude the additional information from T's replies. The MR protocol uses similar abort and resolve sub-protocols. Mukhamedov and Ryan claimed that their protocol is abuse-free because of the use of PCS. However, the situation with MRT protocols is different: a single signer not only sends out his own promise to the intended receiver, but forwards the other signers' promises. It might give a coalition of signers an advantage to the remaining signers. However, the advantage has to be *provable*. How to formalise abuse-freeness in a precise and correct way is a challenging research topic [15–17]. Our immediate future work is to analyse abuse-freeness in the MRT protocols; either we prove the designed MRT protocols abuse-free or we

[8] All Mocha models and ATL properties can be found at [12].

can use the built models to identify a point that a coalition of signers have a provable advantage against an honest signer. In this paper, we have verified protocols with a quite limited number of signers (up to five), and the verification of timeliness properties in Mocha usually took minutes while for fairness properties it might need a number of days. Another future direction is to study abstract interpretation [18] in order to analyse the models in Mocha with more signers.

Acknowledgement. We thank Saša Radomirović for many helpful discussions.

References

1. Asokan, N., Waidner, M., Schunter, M.: Optimistic protocols for fair exchange. In: Proc. CCS, ACM (1997) 7–17
2. Asokan, N., Shoup, V., Waidner, M.: Optmistic fair exchange of digital signatures. Selected Areas in Communications **18**(4) (2000) 591–606
3. Kremer, S., Markowitch, O., Zhou, J.: An intensive survey of fair non-repudiation protocols. Computer Communications **25**(17) (2002) 1606–1621
4. Garay, J.A., MacKenzie, P.D.: Abuse-free multi-party contract signing. In: Proc. PODC. LNCS 1693., Springer (1999) 151–165
5. Chadha, R., Kremer, S., Scedrov, A.: Formal analysis of multi-party contract signing. J. Autom. Reasoning **36**(1-2) (2006) 39–83
6. Mukhamedov, A., Ryan, M.D.: Fair multi-party contract signing using private contract signatures. Inf. Comput. **206**(2-4) (2008) 272–290
7. Mauw, S., Radomirović, S., Torabi Dashti, M.: Minimal message complexity of asynchronous multi-party contract signing. In: Proc. CSF, IEEE CS (2009) 13–25
8. Alur, R., Henzinger, T.A., Mang, F.Y.C., Qadeer, S., Rajamani, S.K., Tasiran, S.: Mocha: Modularity in model checking. In: Proc. CAV. LNCS 1427., Springer (1998) 521–525
9. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5) (2002) 672–713
10. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science (B), MIT Press (1990) 955–1072
11. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods in System Design **15**(1) (1999) 7–48
12. 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/`.
13. 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. CAV. LNCS 2404., Springer (2002) 359–364
14. Adleman, L.: Short permutation strings. Discrete Mathematics **10** (1974) 197–200
15. Chadha, R., Mitchell, J.C., Scedrov, A., Shmatikov, V.: Contract signing, optimism, and advantage. J. Log. Algebr. Program. **64**(2) (2005) 189–218
16. Kähler, D., Küsters, R., Wilke, T.: A Dolev-Yao-based definition of abuse-free protocols. In: Proc. ICALP. LNCS 4052., Springer (2006) 95–106
17. Cortier, V., Küsters, R., Warinschi, B.: A cryptographic model for branching time security properties - the case of contract signing protocols. In: Proc. ESORICS. LNCS 4734., Springer (2007) 422–437
18. Henzinger, T.A., Majumdar, R., Mang, F.Y.C., Raskin, J.F.: Abstract interpretation of game properties. In: Proc. SAS. LNCS 1824., Springer (2000) 220–239