Formal analysis of multi-party contract signing

Rohit Chadha, Steve Kremer, and Andre Scedrov

University of Sussex Université Libre de Bruxelles University of Pennsylvania

Abstract. We analyze the multi-party contract-signing protocols of Garay and MacKenzie (GM) and of Baum and Waidner (BW). We use a finite-state tool, MOCHA, which allows specification of protocol properties in a branching-time temporal logic with game semantics. While our analysis does not reveal any errors in the BW protocol, in the GM protocol we discover serious problems with fairness for four signers and an oversight regarding abuse-freeness for three signers. We propose a complete revision of the GM subprotocols in order to restore fairness.

1 Introduction

The problem of digitally signing a contract over a network is more complicated than signing a contract by "pen and paper". The problem arises because of an inherent asymmetry: no signer wants to be the first one to sign the contract because another signer could refuse to do so after having obtained the first signer's contract.

A simple solution consists in using a trusted party (T) as an intermediary. Signers send their respective contracts to T, which first collects the contracts and then distributes them among the signers. An intermediary is known to be necessary [8]. However, because of the communication and computation bottleneck at T, this solution is inefficient. Other solutions include randomized protocols as well as protocols based on gradual information exchange. More recently, the so-called *optimistic* approach was introduced in [2, 4]. The idea is that T intervenes only when a problem arises, *e.g.,* a signer is trying to cheat or a network failure occurs at a crucial moment during the protocol. Such protocols generally consist of a main protocol and one or several subprotocols, each with a fixed number of messages. The main protocol is executed by the signers in order to exchange their signatures. The subprotocols are used to contact T in order to force a successful outcome or to abort the protocol.

A contract-signing protocol should respect several desirable properties. The first property is *fairness*. Intuitively, a contract-signing protocol is fair if at the end of the protocol either each signer obtains all the other signers' contracts or no signer gets any valuable information. A second property, *timeliness*, ensures that signer has some recourse to prevent endless waiting. Both fairness and timeliness are standard properties that are also important in fair exchange, certified e-mail and fair non-repudiation protocols. A property that is specific to contract signing, *abuse-freeness*, was introduced in [9]. A protocol is abuse-free if no signer A is able to prove to an external observer that A has the power to choose between successfully concluding the protocol and aborting the protocol. A protocol that is not abuse-free gives an undesirable advantage to one signer, say Alice, who has the power to decide the outcome of the protocol and can prove this to an external observer. If, for instance, Alice wants to sell a house to Charlie, she could initiate a contract with Bob just to force Charlie to increase his offer.

There have been several applications of formal methods to contract signing, so far only for the special case of two signers. The finite model-checker Murφ is used in [13] to analyze two contract-signing protocols, discover subtle errors and suggest corrections. In [5] inductive methods are used to reason about contract-signing protocols specified in the multiset-rewriting framework, MSR. Protocol properties are expressed in terms of strategies, which provide a natural framework for the analysis. In [12] the finite model-checker MOCHA is used to analyze two contract-signing protocols. The advantage of using MOCHA rather than Murφ is that MOCHA allows to specify protocol properties in ATL, a temporal logic with game semantics, which in turn allows reasoning about strategies. The most recent work on contract signing [6] introduces the notion of an *optimistic* signer, *i.e.,* a signer that prefers to wait for "some time" for messages from the other signers before contacting the trusted party. The main theorem of [6] is that, independently of a specific protocol, if any of the signers is optimistic, then the other signer will at some point of the protocol have the power to decide the outcome.

All the efforts just described consider only two-party protocols. In this paper we analyze multi-party contract-signing protocols [3, 10]. The protocol goal in that case is that each signer sends its signature on a previously agreed upon contract text to all other signers and that each signer receives all other signers' signatures on this contract. In a multi-party framework, fairness, timeliness, and abuse-freeness should hold against *any* coalition of dishonest parties. Unlike in the two-party case, the complexity level of the multi-party protocols, especially [10], is such that a tool, e.g., a model-checker, is indispensable in the analysis. This partly comes about from an important difference between the two-party and the multi-party case, namely, in the multi-party case T has to be able to overturn its previous abort decisions [9]. As our analysis shows, this feature is particularly difficult to design correctly. We have discovered an essential obstacle in the GM protocol [10], which appears not to be removable without completely changing the subprotocols for T and which leads to the failure of fairness in the case of four signers. We present this attack in detail in the paper. MOCHA did not find any problems with fairness in the BW protocol [3] nor in the GM protocol with only three signers. In the latter case MOCHA did find an amusing problem with abuse-freeness, but this problem is easily corrected. We will discuss these points only briefly because of the lack of space.

Outline of the paper. The rest of the paper is organized as follows. In section 2, we describe the GM protocol. In section 3, we present briefly the finite-state tool, MOCHA, the temporal logic ATL and its game semantics. Modeling of the protocol and protocol assumptions in the game semantics along with the modeling of fairness in ATL is briefly discussed. In section 4, we report on our analysis of the protocol using MOCHA, and present the fairness attack on four signers in detail. We discuss briefly how to restore fairness and present the anomaly with respect to abuse-freeness for three signers. We summarize our results and discuss directions for future work in section 5.

2 Garay-Mackenzie Protocol

In this section we describe the multi-party contract-signing protocol proposed in [10]. For this section and for the rest of the paper, we shall assume that each protocol participant has a private signing key and a corresponding public verification key. Each participant shall be identified with this private/public key pair, and if we say that "*A can ...*", we shall mean anyone that possesses the private key of A.

The protocol allows n ($n \geq 2$) participants, say $P_1, \ldots, P_n$, to exchange signatures with the help of a trusted party T on a preagreed contract text m. P_i is said to have a *contract* if it has everybody's signature on the text m. The order of the participants $P_1, \ldots, P_n$, henceforth referred to as *signers*, and the identity of T are also agreed upon before the protocol begins. The preagreed contract text m contains an identifier that uniquely identifies each protocol instance. In [10], the communication amongst the participants is assumed to be over a network channel in control of a "Dolev-Yao intruder", while the communication between the participants and the trusted party is assumed to be over a private channel.

The protocol uses zero-knowledge cryptographic primitives, *private contract signatures*, that were first introduced in [9]. The private contract signature of A for B on text m with respect to a trusted party T, denoted as $PCS_A(m, B, T)$ has the following properties:
a) $PCS_A(m, B, T)$ can be created by A.
b) $PCS_A(m, B, T)$ can be faked by B. Only A, B and T can tell difference between PCS and its simulation.
c) $PCS_A(m, B, T)$ can be converted into a conventional universally-verifiable digital signature, $S_A(m)$, by both A and T. Only A and T can do this conversion.

The protocol itself consists of three subprotocols: *main*, *abort*, and *recovery* subprotocols. Usually signers try to achieve the exchange by executing the main subprotocol. They contact T using one of the other two subprotocols when they think something is amiss. Once a signer contacts T, it no longer takes part in the main subprotocol. T responds to a request with either an abort token or a signed contract. The decision whether to reply with an abort token or with a signed contract is based on a database maintained by T, which stores all the relevant information of the requests and its responses. Once T sends back a signed contract, it always replies with the signed contract. As discussed below, a decision to abort may, however, be overturned in order to maintain fairness. We discuss the subprotocols in some detail.

Main protocol. The main protocol for n signers is divided into n-levels, that can be described recursively. For each level of recursion, a different "strength" of promise is used. The strength of a promise is denoted by an integer "level", and an "i-level promise from signer A to signer B on a message m" is implemented using PCS: $PCS_A((m, i), B, T)$.

In level i, signers P_i through P_1 exchange i-level promises to sign the contract. The i-level protocol is triggered when P_i receives 1-level promises from $P_{i+1}, \ldots, P_n$. After receiving these promises, P_i sends out its 1 level promise to signers $P_{i-1}, \ldots, P_1$ and waits for $i-1$ level to finish. At the end of the $i-1$ level, $P_1, \ldots P_{i-1}$ have exchanged $i-1$ level promises and P_i receives a $i-1$ level promise from each of the signers $P_1, \ldots, P_{i-1}$. Now $P_i \ldots, P_1$ exchange i level promises, and close the higher levels.

In order to close level a where $a > i$, P_i sends an $(a-1)$-level promise to P_a and waits for a-level promises from signers $P_{i+1}, \ldots, P_a$. After receiving these promises, P_i indicates its willingness to close the level a to signers $P_1, \ldots, P_{i-1}$ by sending them its a-level promise, and in return waits for a-level promises from them. Upon receiving these, P_i sends its a-level promises to $P_{i+1}, \ldots, P_a$ completing its obligation in the a-level protocol. P_i then proceeds to complete $a+1$ level.

Once the n-levels are completed, each signer has a n-level promise from everybody else, and the contract exchange is ready to begin. In this exchange, each signer also sends a $n+1$-level promise to everybody along with its signature on the preagreed text. In order to complete the exchange, signer P_i waits for the contract and $n+1$-level promises from $P_n, \ldots, P_{i+1}$. Upon receiving these, P_i sends its signature and $n+1$-level promises to everybody, and waits for the signatures and $n+1$-level promises from $P_{i-1}, \ldots, P_1$. Once these are received, the protocol ends for P_i, and P_i has the contract.

If some expected messages are not received, P_i may either quit the protocol or contact T. P_i may simply quit the protocol if it has not sent any promises or contact T if it has sent some promises. It may contact T with a request to abort if it has not received any promise from some signer. It may request T to recover the protocol if it has a promise from every other signer. A detailed description of the main protocol is given in table 1.

Table 1 GM multi-party contract-signing protocol—Main

Wait for all higher recursive levels to start

1. $P_j \rightarrow P_i$: $PCS_{P_j}((m, 1), P_i, T)$ $(n \geq j > i)$
   If P_i does not receive 1-level promises from $P_n \ldots P_{i+1}$ in a timely manner, P_i simply quits.

Start recursive level i

2. $P_i \rightarrow P_j$: $PCS_{P_i}((m, 1), P_j, T)$ $(i > j \geq 1)$

Wait for recursive level i-1 to finish

3. $P_j \rightarrow P_i$: $PCS_{P_j}((m, i-1), P_i, T)$ $(i > j \geq 1)$
   If P_i does not receive (i-1)-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i aborts.

Send i-level promises to all lower-numbered signers

4. $P_i \rightarrow P_j$: $PCS_{P_i}((m, i), P_j, T)$ $(i > j \geq 1)$

Finish recursive level i when i-level promises are received

5. $P_j \rightarrow P_i$: $PCS_{P_j}((m, i), P_i, T)$ $(i > j \geq 1)$
   If P_i does not receive i-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i recovers.

Complete all higher recursive levels

For $a = i + 1$ to n, P_i does the following:

6.1. $P_i \rightarrow P_a$: $PCS_{P_i}((m, a-1), P_a, T)$
6.2. $P_j \rightarrow P_i$: $PCS_{P_i}((m, a), P_j, T)$ $(a \geq j > i)$
If P_i does not receive a-level promises from $P_a \ldots P_{i+1}$ in a timely manner, P_i recovers.
6.3. $P_i \rightarrow P_j$: $PCS_{P_i}((m, a), P_j, T)$ $(i > j \geq 1)$
6.4. $P_j \rightarrow P_i$: $PCS_{P_i}((m, a), P_j, T)$ $(i > j \geq 1)$
If P_i does not receive a-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i recovers.
6.5. $P_i \rightarrow P_j$: $PCS_{P_i}((m, a), P_j, T)$ $(a \geq j > i)$

Wait for signatures and (n+1)-level promises from higher-numbered signers

7. $P_j \rightarrow P_i$: $PCS_{P_j}((m, n+1), P_i, T), S_{P_j}(m, 1)$ $(n \geq j > i)$
   If P_i does not receive signatures and (n+1)-level promises from $P_n \ldots P_{i+1}$ in a timely manner, P_i recovers.

Send signatures and (n+1)-level promises to signers

8. $P_i \rightarrow P_j$: $PCS_{P_i}((m, n+1), P_j, T), S_{P_i}(m, 1)$ $(j \neq i)$

Wait for signatures from lower-numbered signers

9. $P_j \rightarrow P_i$: $PCS_{P_j}((m, n+1), P_i, T), S_{P_j}(m, 1)$ $(i > j \geq 1)$
   If P_i does not receive signatures and (n+1)-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i recovers.

Abort protocol. T maintains two sets, S_m and F_m, that are used by T to make decisions when a signer contacts T. These sets are created when T is contacted for the first time for m and are initialized to be empty. The set S_m contains the indices of all signers that have contacted T and received an abort token from T in response. The intuitive meaning of the set F_m is not clearly stated in [10], but it contains some additional information that T uses in deciding when to overturn an abort decision that T has taken before.

The details of the abort protocol are given in table 2. Mainly, if T is contacted with a request to abort, then T checks its database. If this is the first request or if the protocol has not already been recovered, T sends back an abort token and updates the sets S_m and F_m. If the protocol has already been successfully recovered, T sends back a signed contract.

Recovery protocol. The details of the recovery protocol are given in table 3. For P_i to recover, it sends the message $S_{P_i}(\{PCS_{P_j}((m, k_j), P_i, T)\}_{j \in \{1,\ldots n\} \setminus \{i\}}, S_{P_i}((m, 1)))$, where

– if $j > i$, k_j is the maximum level of a promise received from P_j on m,

Table 2 GM multi-party contract-signing protocol—Abort

1. $P_i \rightarrow$ T: $S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort)$
   if not validated(m) then
   if $S_m = \emptyset$, T stores $S_T(S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort))$; $S_m = S_m \cup \{i\}$;
   if i is larger than the maximum index in S_m, T clears F_m
   2. T $\rightarrow P_i$: $S_T(S_{P_j}(m, P_j, (P_1, \ldots, P_n), abort), S_T(m, S_m, abort))$
   else (validated(m)=true)
   3. T $\rightarrow P_i$: $\{S_{P_j}((m, k_j))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$
   where k_j is the level of the promise from P_j that was converted to a universally-verifiable signature during the recovery protocol.

– if $j < i$, k_j is the maximum level of promises received from all signers $P_{j'}$, with $j' < i$, i.e. the min-max of the level of promises from signers with lower index. (E.g., if the maximum level of the promises received by P_4 from P_3 and P_2 was 6, and the maximum level received by P_4 from P_1 was 5, then it would send the 5-level promises for P_1, P_2 and P_3.)

If T is contacted with a request to recover, then T checks its database. If this is the first request for m or if the protocol has already been recovered, T replies with a signed contract which it obtains by converting the promises into conventional digital signatures. Otherwise, if the protocol has already been aborted, T must decide whether to maintain the abort or to overturn it. Overturning of the abort is necessary in order to maintain fairness. Indeed, consider the scenario in which a dishonest P_{n-1} contacts T with an abort request, receives an abort token and dishonestly continues the protocol. After the n-levels are completed, P_n sends its signature to others and waits for signatures from other signers. If P_{n-1} does not send back its signature, then P_n will be forced to contact T with a request to recover. Now, T must overturn its previous abort, otherwise P_n will not receive the signature of P_{n-1}. The decision whether to overturn is based on the contents of the sets S_m and F_m, as described in table 3.

3 Model

ATS, ATL *and* MOCHA. The desired properties of contract signing are easily described using games, and hence we chose a game-variant of Kripke structures, alternating transition systems (ATS) [1], to model the protocols. An ATS is composed of a set of players Σ, a set of states Q that represents all possible game configurations, a set $Q_0 \subseteq Q$ of initial states, a finite set of propositions Π, a labeling function $\pi : Q \rightarrow 2^{\Pi}$ that labels states with propositions, and a game transition function $\delta : Q \times \Sigma \rightarrow 2^{2^Q}$. For a player a and a state q, $\delta(q, a)$ is the set of choices that a can make in the state q. A choice is a set of possible next states. One step of the game at a state q is played as follows : each player $a \in \Sigma$ makes its choice and the next state of the game q' is the intersection (required to be a singleton) of the choices made by all the players of Σ, i.e. $\{q'\} = \cap_{a \in \Sigma} \delta(q, a)$. A computation is an infinite sequence $\lambda = q_0 q_1 \ldots q_n \ldots$ of states obtained by starting the game in q_0, where $q_O \in Q_0$.

In order to reason about ATS, we use alternating-time temporal logic (ATL) [1]. For a given set of players $A \subseteq \Sigma$, a set of computations Λ, and a state q, consider the following game between a protagonist and an antagonist starting in q. At each step, to determine the next state, the protagonist selects the choices controlled by the players in the set A, while the antagonist selects the remaining choices. If the resulting infinite computation belongs to the set Λ, then the protagonist wins. If the protagonist has a winning strategy, we say that the

Table 3 GM multi-party contract-signing protocol—Recovery

1. $P_i \rightarrow \mathsf{T}$: $S_{P_i}(\{PCS_{P_j}((m,k_j),P_i,T)\}_{j\in\{1,\dots n\}\setminus\{i\}}, S_{P_i}((m,1)))$
 if $i \in S_m$, T ignores the message
 else if validated(m)
  2. $\mathsf{T} \rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\dots n\}\setminus\{i\}}$
  where k_j is the level of the promise from P_j that was converted to a universally-verifiable signature.
 else if $S_m = \emptyset$
  validated(m):=true
  3. $\mathsf{T} \rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\dots n\}\setminus\{i\}}$
 else (validated(m)=false $\wedge$ $S_m \neq \emptyset$)
  a. if $i \notin F_m$, then
   (i) if for any $\ell \in S_m$ there is a $j \in S_m$ such that $j > k_\ell$
    $S_m := S_m \cup \{i\}$
    let a be the maximum value in S_m
    if $a > i$ then $\forall j$, such that $k_j = a - 1 \cdot\ F_m := F_m \cup \{j\}$
    else $F_m := \emptyset$
     4.1.1. $\mathsf{T} \rightarrow P_i$: $S_T(S_{P_j}(m,P_j,(P_1,\dots,P_n),abort), S_T(m,S_m,abort))$
     where $S_T(S_{P_j}(m,P_j,(P_1,\dots,P_n),abort))$ corresponds to the stored abort token
   (ii) else
    validated(m):=true
    4.1.2. $\mathsf{T} \rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\dots n\}\setminus\{i\}}$
  b. else $(i \in F_m)$
   let a be the maximum value in S_m
   (i) if $(\forall j$, such that $i < j \leq a \cdot\ k_j < a) \wedge (\forall j < i \cdot\ k_j \geq a)$
    validated(m):=true
    4.2.1. $\mathsf{T} \rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\dots n\}\setminus\{i\}}$
   (ii) else
    $S_m := S_m \cup \{i\}$
    if $a > i$ then $\forall j, k_j = a - 1 \cdot\ F_m := F_m \cup \{j\}$
    if $a = i$ then $F_m := \emptyset$
     4.2.2. $\mathsf{T} \rightarrow P_i$: $S_T(S_{P_j}(m,P_j,(P_1,\dots,P_n),abort), S_T(m,S_m,abort))$
     where $S_T(S_{P_j}(m,P_j,(P_1,\dots,P_n),abort))$ corresponds to the stored abort token

ATL formula $\langle\langle A\rangle\rangle\Lambda$ is satisfied in state q. Here, $\langle\langle A\rangle\rangle$ is a path quantifier, parameterized by the set A of players, which ranges over all computations that the players in A can force the game into, irrespective of how the players in $\Sigma \setminus A$ proceed. The set Λ is defined using temporal logic formulas. For those familiar with branching time temporal logics, the parameterized path quantifier $\langle\langle A\rangle\rangle$ can be seen as a generalization of the CTL path quantifiers: the existential path quantifier $\exists$ corresponds to $\langle\langle \Sigma\rangle\rangle$ and the universal path quantifier $\forall$ corresponds to $\langle\langle \emptyset\rangle\rangle$.

We now illustrate the expressive power of ATL that allows to model both cooperative as well as adversarial behavior amongst the players. Consider the set of players $\Sigma = \{a,b,c\}$ and the following formulas with their verbal reading:

- $\langle\langle a\rangle\rangle\Diamond p$, player a has a strategy against players b and c to eventually reach a state where the proposition p is true;
- $\neg\langle\langle b,c\rangle\rangle\Box p$, the coalition of players b and c does not have a strategy against a to make p true forever;
- $\langle\langle a,b\rangle\rangle\bigcirc(p \wedge \neg\langle\langle c\rangle\rangle\Box p)$, a and b can cooperate so that the next state satisfies p and from there c does not have a strategy to impose p forever.

The details of ATS and ATL can be found in [1]. Instead of modeling protocols directly with ATS we use a more user-oriented notation: a guarded command language a la Dijkstra. The details about the syntax and semantics of this language (given in terms of ATS) can be found in [11]. Intuitively, each player $a \in \Sigma$ disposes of a set of guarded commands of the form $guard_\xi \rightarrow update_\xi$. A computation-step is defined as follows: each player $a \in \Sigma$ chooses one of its commands whose boolean guard evaluates to true, and the next state is obtained by taking the conjunction of the effects of each update part of the commands selected by the players. Given an ATS described in terms of guarded commands, the finite state tool MOCHA automates the model-checking of ATL formulae over the specified ATS.

Modeling protocols. Unlike the classical security protocols aiming at secrecy and authentication, optimistic contract-signing protocols usually consist of subprotocols that can be invoked at specified moments. Running a protocol at a time not foreseen by the designer, may have unexpected side-effects. This may be used by a signer to gain an advantage over other signers. We believe that such concurrency issues are a major source of problems. Therefore, and since the high number of messages would create a serious state explosion, we only analyze the *structure* of the protocols and concentrate only on single protocol instance. We now discuss our model in detail.

The protocol instance is modeled as an ATS and each protocol participant is modeled as a player in the ATS using the above introduced guarded command language. Besides, the branching aspect, another notable difference with more classical secrecy and authentication protocols, is that contract-signing protocols must be secure against malicious signer, rather than an external intruder. In order to model this we have two process for each signer, one describing it's honest behavior and the other the dishonest one. Communication is modeled using shared variables. Each protocol message is modeled using a boolean variable, initialized to *false* and set to *true* when it is sent. Sending of a message is modeled using guarded commands, where the guard depends on previously sent out messages. When modeling the honest behavior of a participant, we ensure that a given message is sent out only when specified by the protocol. In contrary, the guards are relaxed in the malicious version of the signer so that each message can be sent out, as soon as possible, i.e., as soon as all messages needed to compose the given message are received. We do not explicitly model any cryptographic primitives, but only the fact that protocol messages can be sent out of order. Hence, a dishonest signer can send messages out of order and continue the protocol, even if it is supposed to stop. We manually decide which messages must be known in order to send some other message. Moreover, the communication between any two signers is assumed to be on private channels and we do not model the possibility to spy other channels. The trusted party is modeled to be always honest.

As an example, consider the short extract of the modeling of the three-party GM protocol depicted in figure 1. In the extract, the integer variable `Pr_i_j_L` models the promises that P_i has sent to P_j, and `Pr_i_j_L = k` means that P_i has sent out up to k-level promises to P_j. (For efficiency reasons, we use the logarithmic encoding of a ranged integer variable, rather than having one boolean variable for each level of promise). In the extract, the first rule of honest P_1 says that P_1 may quit the protocol, if it has not contacted the trusted party, and has neither received nor sent any promises. The corresponding modeling of dishonest behavior of P_1 states that P_1 may quit the protocol at any moment. The second rule of honest P_1 gives the exact condition when the first level promise has to be sent to P_2. The corresponding dishonest rule, merely requires that P_1 has not quit the protocol before sending the promise.

As we are unable to verify parametric systems with MOCHA, we simplify our task and verify the protocols only for a given number n of signers. In order to avoid encoding each instance of the protocol using guarded commands, we have written a dedicated C++ program

Extract of honest modeling of P_1 for the three-party GM protocol:

```
[] ~P1_stop & ~P1_contacted_T & Pr_1_3_L=0 & Pr_1_2_L=0 & ~( Pr_3_1_L>0 & Pr_2_1_L>0 )
   -> P1_stop':=true
[] ~P1_stop & ~P1_contacted_T & Pr_1_3_L=0 & Pr_1_2_L=0 & Pr_2_1_L>0 & Pr_3_1_L>0
   -> Pr_1_2_L':=1
```

The corresponding actions of a dishonest modeling:

```
[] ~P1_stop -> P1_stop':=true
[] ~P1_stop & Pr_1_2_L<1 -> Pr_1_2_L':=1
```

Fig. 1: Extract of the three-party GM protocol modeling

for each protocol which takes the number n of signers as a parameter and generates the protocol specification. Although our model is restricted with regard to several aspects, the model seems to be of interest as several unknown anomalies have been revealed.

Modeling properties. We express the desired security guarantees using ATL. For lack of space, we concentrate on modeling of fairness. Consider an instance of the protocol with n signers, which we denote as $P_1, \ldots, P_n$. In the following, we assume that only one of the signers, say P_1 is honest, and the other dishonest signers are colluding to cheat the honest signer.

A protocol is fair for an honest P_1, if at the end of the protocol, either P_1 receives signed contracts from all the other signers or it is not possible for any other signer to obtain P_1's signed contract. One possible ATL formula for modeling this says that if any signer receives P_1's signed contract, then P_1 has a strategy to get the signed contracts from other signers:

$$\forall\Box\,((P_2.S_{P_1}(m) \vee \ldots \vee P_n.S_{P_1}(m)) \rightarrow \langle\langle P_1\rangle\rangle\Diamond\,(P_1.S_{P_2}(m) \wedge \ldots \wedge P_1.S_{P_n}(m))) \tag{1}$$

where $P_i.S_{P_j}(m)$ denotes that player P_i received P_j's signature on the contract text m.

It can of course be argued that P_1 having a strategy to receive the signed contracts is not a sound modeling of fairness: P_1 may have this strategy but if it is ignorant or if it mistakenly does not follow this strategy, then the protocol may end in an unfair state. Therefore one could require the following stronger property: in whatever way P_1 resolves the remaining choices specified by the protocol, P_1 receives all the signed contract.

$$\forall\Box\,((P_2.S_{P_1}(m) \vee \ldots \vee P_n.S_{P_1}(m)) \rightarrow \forall\Diamond\,(P_1.S_{P_2}(m) \wedge \ldots \wedge P_1.S_{P_n}(m))) \tag{2}$$

In the same vein, a third formulation only requires that there exists a path where P_1 receives the signed contracts.

$$\forall\Box\,((P_2.S_{P_1}(m) \vee \ldots \vee P_n.S_{P_1}(m)) \rightarrow \exists\Diamond\,(P_1.S_{P_2}(m) \wedge \ldots \wedge P_1.S_{P_n}(m))) \tag{3}$$

Formula (2) implies formula (1), which in turn implies formula (3). We concentrate on the last, weakest version of fairness. As we show that fairness is violated even in this weakest version, the other stronger versions are also violated. Now, we are ready to discuss our analysis.

4 Analysis

We now report on our the analysis of the GM contract-signing protocol. The protocol has several peculiarities. The most notable one is that the protocol changes with the number of signers, e.g., the protocol specification of P_1 differs when the value of n changes. The number of protocol messages increases considerably with the number of signers. For instance, if we have $n = 3$, the main protocol has 20 messages and there are 14 different recovery requests. When $n = 4$, the corresponding numbers are 41 and 36. Moreover, the protocol is not symmetric for the signers: the protocol specification for P_i is different from that for P_j, for all $i \neq j$. For instance, when $n = 4$, P_1 can launch 18 different recovery requests and P_4 only 2.

As mentioned in section 3, we have written a dedicated C++ program that takes the number of signers, n, as a parameter and generates the protocol specification. Our analysis revealed problems with fairness, when n is 4. Although, we did not discover any fairness problems when $n = 3$, we did find an amusing problem with abuse-freeness. We did not discover any problems with timeliness in the protocol. All these anomalies are novel and the protocol was believed to be secure since it was first published. We discuss our results in detail.

Fairness. We did not discover any problems with fairness when $n = 3$. The formulas representing fairness for P_1, P_2 and P_3, introduced in section 3, are validated by MOCHA. However, as we use a restricted model and consider single runs, we can only conclude that the protocol does not present any *structural* weakness for $n = 3$. Indeed, if we relax the assumption of the private channels, the anomaly presented by Shmatikov and Mitchell in [13] on the two-party GJM protocol can be adapted to the multi-party version. In this scenario, a malicious signer eavesdrops on the channel between the honest signer and T, and succeeds in compromising fairness. With our present modeling, we do not find such flaws, as this requires to eavesdrop channels and to decompose messages. The fix proposed by Shmatikov and Mitchell applies to the multi-party protocol too. However, we should emphasize that the authors of the GM protocol require the channels to T to be private and hence this scenario does not represent a valid attack on the protocol.

We discovered several scenarios that compromised fairness when $n = 4$. The first secenario was discovered by hand, when we found an error in the proof of correctness given in the original paper [10]. A detailed analysis using MOCHA detected seven other scenarios. An analysis of these revealed that the proof also did not cover a case. In each scenario, an honest signer is cheated by the coalition of three malicious signers. These scenarios follow the general outline:

1. A dishonest signer contacts the trusted party, T, at the beginning of the protocol, gets an abort token, and dishonestly continues participating in the main protocol.
2. A second dishonest signer tries to recover at some later point. It does not succeed, but manages to put the honest signer in the list F_m. It dishonestly continues the main protocol.
3. The honest signer is forced to recover, but is not successful in getting the abort decision overturned since it is in the list F_m.
4. The third dishonest signer contacts T and manages to overturn the decision. Hence, while the honest signer does not get any signed contract, the honest signer's contract is obtained.

For lack of space, we just describe one of these scenarios in detail. In this scenario, P_1, P_3 and P_4 collude to cheat P_2. The scenario proceeds as follows:

- At the beginning of the protocol, P_3 aborts the protocol and T updates $S_m = \{3\}$. However, unlike specified by the protocol, dishonest P_3 continues the main protocol execution.
- As soon as P_1 receives the second level promise from P_2, it asks T to recover by sending

$$S_{P_1}(\{PCS_{P_2}((m,2),P_1,T), PCS_{P_3}((m,1),P_1,T), PCS_{P_4}((m,1),P_1,T)\}, S_{P_1}((m,1))).$$

T refuses this request, answers with an abort message and updates $S_m = \{1,3\}$ and $F_m = \{2\}$. As P_3 did before, P_1 also continues the protocol.

- The main protocol is executed normally until signer P_2 reaches point 6.2. (see table 1) of the protocol with $a = 4$. At that point P_2 has sent out the set of message

$$\{PCS_{P_2}((m,1),P_1,T), PCS_{P_2}((m,2),P_1,T), PCS_{P_2}((m,2),P_3,T), \\ PCS_{P_2}((m,3),P_1,T), PCS_{P_2}((m,3),P_3,T), PCS_{P_2}((m,3),P_4,T)\}$$

  and has received the set of messages

$$\{PCS_{P_4}((m,1),P_2,T), PCS_{P_3}((m,1),P_2,T), PCS_{P_1}((m,1),P_2,T), \\ PCS_{P_1}((m,2),P_2,T), PCS_{P_3}((m,3),P_2,T), PCS_{P_1}((m,3),P_2,T)\}.$$

  P_2 is at position 6.2. with $a = 4$ and is waiting for 4-level promises from P_3 and P_4. P_3 and P_4 do not reply and P_2 is forced to send the following recovery request to T.

$$S_{P_2}(\{PCS_{P_2}((m,3),P_2,T), PCS_{P_3}((m,3),P_2,T), PCS_{P_4}((m,1),P_2,T)\}, S_{P_2}((m,1))).$$

  P_2 is in F_m, the tests in the protocol description (see table 3) indicate that T refuses the request, updates $S_m = \{1, 2, 3\}$ and replies with an abort message. F_m remains unchanged.
- P_4 launches a resolve request, sending

$$S_{P_4}(\{PCS_{P_1}((m,3),P_4,T), PCS_{P_2}((m,3),P_4,T), PCS_{P_3}((m,3),P_4,T)\}, S_{P_4}((m,1))).$$

  This request overturns the previous aborts, and hence violates fairness as T sends the signed contract back to P_4.

In other scenarios, we discovered that protocol is unfair for signers P_1, P_2 and P_3 when $n = 4$. We did not find any scenario that compromised fairness for P_4. This is probably because the tests indicate that P_4 can never be added to F_m, when $n = 4$.

Correcting the Garay-MacKenzie Protocol. In order to restore fairness in the Garay-MacKenzie protocol, we had to do major revisions in the recovery messages and the protocol for T. We were unsuccessful to restore fairness with minor changes, and we believe that this is because the meaning of the list F_m is not clear in the protocol. The central idea behind the revision is that T, when presented with a recovery request, overturns its abort decision if and only if T can infer dishonesty on the part of each of the signer that contacted T in the past. This is also the main idea behind the recovery protocol in [3]. For lack of space considerations, we describe the revision briefly.

The main protocol and the abort messages for a signer are essentially the same. The recovery messages are however slightly modified. These messages are designed so that T can infer the promises that an honest signer would have sent when it launched the recovery protocol (note that a signer may have dishonestly sent other promises).

T maintains the set S_m of indices of signers that contacted T in the past and received an abort token. For each signer P_i in the set S_m, T also maintains two integer variables $h_i(m)$ and $l_i(m)$. Intuitively, h_i is the highest level promise an honest P_i could have sent to any higher indexed signer before it contacted T. l_i is the highest level promise an honest P_i could have sent to a lower indexed signer before it contacted T. The protocol for T works as follows:

- If T ever replies with a signed contract for m, then T responds with the contract for any further request.
- If the first request to T is a resolve request, then T sends back a signed contract.
- If the first request is an abort request, then T aborts the contract. T may overturn this decision in the future if it can deduce that all the signers in S_m have behaved dishonestly. T deduces that a signer P_i in S_m is dishonest when contacted by P_j if

1. $j > i$ and P_j presents to T a k-level promise from P_i such that $k > h_i(m)$, or
2. $j < i$ and P_j presents to T a k-level promise from P_i P_i such that $k > l_i(m)$.

We analyzed the revised protocol for both 3 and 4 signers and MOCHA did not detect any errors in the revised protocol. Please note that this should not be construed as proof of correctness since we are using a restricted communication model and are modeling a single run. Nevertheless, we believe that the revised protocol would be fair in a more general setting, and for an arbitrary number of signers.

Abuse-freeness. We briefly describe the anomaly that we discovered for $n = 3$ signers in the GM protocol. The anomaly exploits the fact that when T replies with an abort decision, it also signs the list S_m of the signers who have received an abort from T. Recall that an optimistic signer [6] is one that prefers to wait for "some time" before contacting the trusted party. Following [6], we say that a protocol is abuse-free for a signer P_i if the protocol does not provide *provable advantage* to the remaining signers. A coalition of signers is said to have provable advantage against P_i at a point in the protocol if (i) they have a strategy to abort the contract against an optimistic P_i, (ii) they have a strategy to get optimistic P_i's contract, and (iii) they can prove to an outside challenger, Charlie, that P_i is participating in the protocol.

Now consider the protocol instance with three signers P_1, P_2 and P_3. Assume that P_3 is optimistic and P_1 and P_2 are colluding to cheat P_3. P_3 starts the protocol by sending its level 1 promises to P_1 and P_2, and waits for level 2 promises from them. P_2 on receiving this sends its level 1 promise to P_1, and then sends an abort request to T which aborts the protocol. Now, P_1 has received level 1 promises from P_2 and P_3. Using these first level promises, P_1 sends a recovery request to T. Note that, in the protocol, P_1 is never allowed to abort and T would not accept an abort request from P_1. P_1's recovery request is refused and T sends

$$S_T(S_{P_2}(m, P_2, (P_1, P_2, P_3), abort)) \text{ and } S_T(m, S_m = \{1, 2\}, abort)$$

At this point, we make the following observations:

- the abort reply contains the set $S_m = \{1, 2\}$ and is different from the one P_2 received,
- if P_1 receives an abort reply from T, it is always the answer to a recovery request,
- a recovery request always includes a promise from each signer which is verifief by T.

From these remarks, we can conclude that if P_1 shows the abort reply to Charlie, then Charlie will be convinced that P_3 has started the protocol even though Charlie is unable to verify the PCS from P_3. In other words, we can say that T has verified the PCS for Charlie. Also note that at this point P_1 and P_2 can force the exchange to abort by simply quitting the protocol: P_3 has no promises from P_1 and P_2. P_1 and P_2 can also force a successful completion of the contract exchange by simply (dishonestly) engaging P_3 in the main protocol. Hence the protocol is not abuse-free for P_3.

This vulnerability can be easily addressed by excluding the set S_m from the abort reply. In this case, the abort messages from P_3 and P_2 are exactly similar and can be obtained by P_2 without P_3's participation. Hence, an abort reply does not prove P_3's participation in the protocol. This rather amusing scenario illustrates that sometimes additional information may be harmful. While explicitness is often considered a good engineering practice (and we do not attempt to criticize such thumb rules), care should be taken when applying these principles. In personal communication with the authors of the protocol, they propose a different fix in letting P_1 abort the protocol rather than just quitting. The above weakness is detected by MOCHA using an ATL formula that approximates abuse-freeness. Currently, we are working on a full formulation of abuse-freeness and optimistic behavior in our model.

5 Conclusions and Future Work

We have studied two multi-party contract-signing protocols [10, 3] using a finite-state tool, MOCHA, that allows specification of properties in a branching-time temporal logic with game semantics. In order to make this analysis feasible, we model single runs and assume a restricted communication model. Our analysis did not find any errors in the BW protocol [3]. We did encounter problems with fairness in the case of four signers in the GM protocol [10]. It appears that fairness cannot be restored without completely rewriting the subprotocols. We also discovered a rather amusing problem with abuse-freeness in the GM protocol with three signers that occurs because abort messages from the trusted party reveal who have contacted it in the past. This problem is easily addressed by ensuring that the trusted party does not send this extra information.

We are currently working on a complete formalization of optimistic signers and abuse-freeness in MOCHA. We plan to verify the protocols without fixing the number of signers. One major challenge in such a parametric verification is that the protocol descriptions change fundamentally with the number of signers in that the protocol for n signers is not merely putting n identical processes in parallel. We hope to prove the correctness of these protocols in a more general setting which accounts for cryptography, multiple concurrent sessions, and relaxes the communication model. We plan to use, at least partially, abstraction techniques such as proposed by Das and Dill [7] to achieve this.

References

1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *38th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 1997.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *4th ACM Conference on Computer and Communications Security*, Zurich, Switzerland, Apr. 1997. ACM Press.
3. B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In *Automata, Languages and Programming — ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 524–535, Geneva, Switzerland, July 2000. Springer-Verlag.
4. H. Burk and A. Pfitzmann. Value exchange systems enabling security and unobservability. In *Computers and Security, 9(8):715–721*, 1990.
5. R. Chadha, M. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In *8th ACM Conference on Computer and Communications Security*, Philadelphia, PA, USA, Nov. 2001. ACM Press.
6. R. Chadha, J. C. Mitchell, A. Scedrov, and V. Shmatikov. Contract signing, optimism, and advantage. In *CONCUR 2003 — Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
7. S. Das and D. L. Dill. Successive approximation of abstract transition relations. In *Sixteenth Annual IEEE Symposium on Logic in Computer Science*, 2001.
8. S. Even and Y. Yacobi. Relations among public key signature systems. Technical Report 175, Technion, Haifa, Israel, Mar. 1980.
9. J. A. Garay, M. Jakobsson, and P. D. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology—Crypto 1999*, volume 1666 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.
10. J. A. Garay and P. D. MacKenzie. Abuse-free multi-party contract signing. In *International Symposium on Distributed Computing*, volume 1693 of *Lecture Notes in Computer Science*, Bratislava, Slavak Republic, Sept. 1999. Springer-Verlag.
11. T. A. Henzinger, R. Manjumdar, F. Y. Mang, and J.-F. Raskin. Abstract interpretation of game properties. In *SAS 2000: Intertional Symposium on Static Analysis*, Lecture Notes in Computer Science. Springer-Verlag, 2000.
12. S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *15th IEEE Computer Security Foundations Workshop*, Cape Breton, Canada, June 2002. IEEE Computer Society Press.
13. V. Shmatikov and J. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science, special issue on Theoretical Foundations of Security Analysis and Design*, 283(2):419–450, 2002.