Research Article

Formal Analysis of Fairness for Optimistic Multiparty Contract Signing Protocol

Xiaoru Li, Xiaohong Li, Guangquan Xu, Jing Hu, and Zhiyong Feng

School of Computer Science and Technology, Tianjin University, Tianjin 300072, China

Correspondence should be addressed to Xiaohong Li; xiaohongli@tju.edu.cn

Received 13 February 2014; Accepted 20 April 2014; Published 18 June 2014

Academic Editor: Guiming Luo

Copyright © 2014 Xiaoru Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Optimistic multiparty contract signing (OMPCS) protocols are proposed for exchanging multiparty digital signatures in a contract. Compared with general two-party exchanging protocols, such protocols are more complicated, because the number of protocol messages and states increases considerably when signatories increase. Moreover, fairness property in such protocols requires protection from each signatory rather than from an external hostile agent. It thus presents a challenge for formal verification. In our analysis, we employ and combine the strength of extended modeling language CSP# and linear temporal logic (LTL) to verify the fairness of OMPCS protocols. Furthermore, for solving or mitigating the state space explosion problem, we set up a state reduction algorithm which can decrease the redundant states properly and reduce the time and space complexity greatly. Finally, this paper illustrates the feasibility of our approach by analyzing the GM and CKS protocols, and several fairness flaws have been found in certain computation times.

1. Introduction

The first optimistic multiparty contract signing protocol (OMPCS) was designed by Asokan et al. in 1997 [1]. The goal of this protocol is for all signatories to send signatures on a preagreed-upon contract text to all others and for every signatory to obtain all other signatories’ signatures on this contract. With the asymmetric structure, one partner must first send out his signature to the others; thus the fairness of such protocols is hard to guarantee. One method to solve the problem is to use a Trust Third Party (TTP) as an intermediary [2], but this method is not efficient and the TTP becomes a bottleneck as all signatories have to communicate with it. The other method is to design the so-called optimistic multiparty contract signing protocol which has the idea that only when an unfairness problem arises the TTP intervenes [1]. In 2009, Mauw et al. proposed the notion abort-chaining attacks and analyzed the message complexity of OMPCS protocols [3]. Resolve-impossibility which means that it is impossible to define a trusted party protocol for a special OMPCS protocol was presented [4].

Some specific properties such as fairness, timeliness, and abuse-freeness should be satisfied in OMPCS protocols. Asokan defines a fairness system as that if a player behaves correctly, the other players will not gain any advantage over the correctly behaving player, and he divides fairness into strong fairness and weak fairness. Strong fairness means that, when the protocol has completed, A has B’s item, or B has gained no additional information about A’s item, and vice versa. Here, the assumed “item” means the signed contract, and “additional information” means information which can be obtained from the signed contract. However, weak fairness means that either strong fairness is achieved or a correctly behaving node can prove to an arbiter that an unfair situation has occurred [5].

Although not so much attention has been paid to optimistic multiparty contract signing protocols, there have been several researchers who did certain remarkable work in the weak fairness verification field and got some achievements. The pioneer work can be traced to Chadha et al. [6]. Based on the alternation transition system (ATS) and symbolic tool Mocha, GM protocol was verified to be unfair when the number of signatories is four. Those authors also presented a
CKS protocol and verified that the fairness can be guaranteed in the case of four. Then such method was used in [7] to analyze the fairness of MRT [8] and MR [9] protocols. The MRT protocol failed to satisfy fairness when the number of participators is three, and no flaw has been revealed in the MR protocol.

As an analyzing method, the strand space model has lots of magnificent qualities such as intuitiveness and strictness. In [10], Mukhamedov et al. used a strand space model to describe the GM protocol and found a flaw. Theorem proving method uses formulae to verify the correctness of protocols. The CKS protocol was shown to have flaws in [11] by using the Isabelle/HOL theorem proving machine.

This paper describes a precise and nature formulation of the desired weak fairness and universal OMPCS protocol models that includes multisignatories and contract texts promises simultaneously, formalizes protocol arithmetic, and considers composite attackers of the dishonest signatory. Besides, to mitigate the state explosion problem, a state reduction algorithm is proposed, and conversions between processes are supported, all of such actions can help our analysis to reach cases of more signatories. Furthermore, a visual attacking trace will be given when there exist error traces.

The paper is structured as follows. Section 2 defines the notion of OMPCS protocols and weak fairness. The CSP# language and the LTL logic are briefly introduced in Section 3. This section also explains how to build models for OMPCS protocols and how to express the weak fairness property. In the next section, we apply our novel method to the analysis of two typical OMPCS protocols GM and CKS. Finally, conclusions and future work are summarized.

2. Definitions

Definition 1 (optimistic multiparty contract signing). A protocol for at least $n$ ($n \geq 2$) signatories $P_1, \ldots, P_n$ to sign a contract $m$ over a Trust Third Party is called a multiparty contract signing (MPCS) protocol. If all signatories are honest, the protocol terminates without $T$ ever sending or receiving any messages; it is called an optimistic multiparty contract signing (OMPCS) protocol. An OMPCS protocol consists of three subprotocols: main subprotocol, recovery subprotocol, and abort subprotocol. The main subprotocol for $n$ signers is divided into $n$-levels, which can be described recursively. The recovery and abort subprotocols are used to contact $T$ when something goes wrong.

In an OMPCS protocol, every signer has a public and private key pair and can make a digital signature with the private key. $PCS_{P_i}(m, P_j, T)$ ($T$ is short for TTP) denotes the promise signed by $P_i$ and sent to $P_j$, and $[10]\text{Definition 2} (\text{weak fairness})$. For a protocol $\Gamma$, a contract $m$, and signers $P_i$ and $P_j$ with each one's signed contract $S_{P_i}(m)$ and $S_{P_j}(m)$, we call the protocol $\Gamma$ which satisfies weak fairness if and only if we get one of the following conditions:

(a) when the protocol is terminated, both $P_i$ and $P_j$ have not received $S_{P_i}(m)$ and $S_{P_j}(m)$ from each other.

(b) or when the protocol is terminated, both $P_i$ and $P_j$ have received $S_{P_i}(m)$ and $S_{P_j}(m)$ from each other.

3. Formal Method

Unlike classical security protocols, besides the security property, the OMPCS protocols also need to guarantee fairness between participants. For verifying such property, this paper described an innovative method and Figure 1 shows the structure of it.

3.1. Assumption. Considering the general conditions of OMPCS protocol, assumptions for our fairness analyzing method are as follows.

(a) Channel assumption (lower-case): channels between participants are not trusty; that is to say, the transmitting messages may be delayed and lost. However, channels between participants and the TTP are really reliable; that is, although the transmitting messages may be delayed, they will reach the destination in the limited time.

(b) Participants of protocols: participants may not be honest, but there is at least one honest among a number of participants. The TTP is always honest, and the honest one will follow protocols. A dishonest one can be legal user, possessing his/her own public and private keys.

(c) Attacker: the cryptography is assumed to be perfect, and external hostile agent is assumed not existing for fairness property in OMPCS protocols requires protection from each signatory rather than from external attackers. All of the assumption can let us only concentrate on the structure of the protocols.

3.2. System Variables. CSP# was proposed as an extension of CSP; it combines high-level modeling operators with shared variables and low-level programming constructs. The idea is to treat sequential terminating programs as atomic events [12].

The CSP# model of an OMPCS protocol consists of participants (honest or dishonest), TTP and the communication channel. CSP# mathematical signals were applied to describe the protocols' participants and the components which combine participants together. The syntax of basic operators is listed in Table 1. Where $P, Q$ are processes, $e$ is a name representing an event with an optional sequential program prog, cond is a boolean formula, $X$ is a set of events names, $ch$ is a channel, $exp$ is an expression, and $x$ is a variable.
3.3. Modeling Protocol. The semantic model is based on the labeled transition system (LTS). An LTS is a three-tuple $L = (S, S_0, R)$.

(i) $S$ is set of states. A state can be described by giving value for all valuations in the CSP# model.

(ii) $S_0 \in S$ is the set of initial states.

Under the modeling language CSP#, the system variables of protocol model are defined in Table 2. Each sending message between signers is modeled using a variable, initialized to 0 and set as $k$ when the successfully sent message level is $k$. Each sending message between signers and TTP is modeled using a boolean variable, initialized to false and set true when message was sent.

3.4. Modeling Weak Fairness. In order to reason protocol model, fairness assertion is described by LTL (linear temporal logic). LTL was proposed by Pnueli in 1977 [13] and is used to verify computer program logic language.

LTL is defined by assuming that the atomic formulae are state predicates. Formulae are built up in the usual way according to the following grammar. $AP$ is a collection of atomic propositions of LTL:

(a) for all the atomic propositions $\rho \in AP$, $\rho$ is a LTL formula;

(b) the constants “true” and “false” are both LTL formulae;

(c) if $\phi, \psi$ are two LTL formulae, then $\neg \phi, \phi \lor \psi, \phi \land \psi, G \phi$, and $F \phi$ are all LTL formulae;

$$R : S \times \sum \tau \times S$$ is the transition relation, $\sum \tau$ is a set of all events, $\sum \ast$ is a set of all traces, and a trace is a sequence of events. For every state $s \in S$, there is a state $s' \in S$ such that $(s, s') \in R$. $R$ is process expressions in our CSP# protocol model.

A CSP# model configuration is composited of two components $(V, P)$, where $V$ is a function mapping a variable name to its value and $P$ is a process expression. Then we can get the LTS $= (S, \text{init}, \rightarrow)$, where $S$ is the set of reachable system configurations, init is the initial configuration $(V, P)$, and $\rightarrow$ is a transition relation.

For every signer in protocol, we use a process to describe $P_{\text{H-process}}$, which describes how honest signer $P_i$ behaves and the dishonest $P_j$ has a corresponding process $P_j\text{-process}$. $T\text{-process}$ is defined to model $T$ which represents the processes of recovery and abort subprotocols. The protocol is then described as formula (1); $P$ is the set of signatories:

$$\text{sys} = P_{\text{H-process}} ||| P_{j\text{-process}} \quad (i, j \in P).$$

**Table 1: Syntax of the CSP#.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>Processes do nothing</td>
</tr>
<tr>
<td>Skip</td>
<td>Successful termination</td>
</tr>
<tr>
<td>$P \mid Q$</td>
<td>General choice</td>
</tr>
<tr>
<td>$P \ast$</td>
<td>External choice</td>
</tr>
<tr>
<td>$P \mid Q$</td>
<td>Internal choice</td>
</tr>
<tr>
<td>$P \parallel Q$</td>
<td>Interleaving</td>
</tr>
<tr>
<td>$e { p r o g } \rightarrow P$</td>
<td>Event prefixing</td>
</tr>
<tr>
<td>$P \setminus X$</td>
<td>Hiding all occurrences of event in $X$</td>
</tr>
<tr>
<td>$[\text{cond}]P$</td>
<td>Guarded process</td>
</tr>
<tr>
<td>$P; Q$</td>
<td>Sequential composition</td>
</tr>
<tr>
<td>$\text{ch!exp} \rightarrow P$</td>
<td>Channel output</td>
</tr>
<tr>
<td>$\text{ch?exp} \rightarrow P$</td>
<td>Channel input</td>
</tr>
</tbody>
</table>
Table 2: System variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{j,j} )</td>
<td>( P_{j,j} = k ), if ( P_j ) has successfully sent the ( k ) level promise to ( P_j )</td>
</tr>
<tr>
<td>( P_{j,\text{recovery},j} )</td>
<td>( P_j ) sends recovery requirement message to TTP; ( j ) is the max level of message ( P_j ) has sent to ( P_j )</td>
</tr>
<tr>
<td>( P_{j,\text{recovery},j,j} )</td>
<td>( P_j ) has successfully received ( P_j ) signed contract</td>
</tr>
<tr>
<td>( P_{stop} )</td>
<td>( P_j ) quits the protocol</td>
</tr>
<tr>
<td>( P_{\text{contacted},T} )</td>
<td>( P_j ) has sent recovery or abort requirement messages to TTP</td>
</tr>
<tr>
<td>( P_{Sj} )</td>
<td>( P_j ) has successfully received the ( P_j ) signed contract</td>
</tr>
<tr>
<td>( P_{\text{AbortTaken}} )</td>
<td>( P_j ) sends abort requirement message to TTP</td>
</tr>
<tr>
<td>( T_{\text{Respondi}} )</td>
<td>TTP has responded to the requirement of ( P_j )</td>
</tr>
<tr>
<td>( T_{\text{Recovery send},P_i} )</td>
<td>TTP sends recovery message to ( P_i )</td>
</tr>
<tr>
<td>( T_{\text{Abort send},P_i} )</td>
<td>TTP sends abort message to ( P_i )</td>
</tr>
<tr>
<td>( T_{Fi} )</td>
<td>( P_i ) has not sent abort message and TTP will force him/her to abort in certain situation</td>
</tr>
<tr>
<td>( T_{Si} )</td>
<td>( P_i ) has sent abort message</td>
</tr>
<tr>
<td>( T_{ji} )</td>
<td>The highest level ( P_j ) has sent to higher signer before it contacts TTP</td>
</tr>
<tr>
<td>( T_{ji} )</td>
<td>The lowest level ( P_j ) has sent to lower signer before it contacts TTP</td>
</tr>
<tr>
<td>( T_{ki} )</td>
<td>The highest level ( P_k ) has received from all signers ( m ), ( m, j &lt; i )</td>
</tr>
</tbody>
</table>

(d) every LTL formula can be built up by using finite times of the above formation rules.

Here \( \phi \) and \( \psi \) are two formulae, \( G \phi \) reads as “global” (also can be written as [],), and \( G \phi \) means that event \( \phi \) has to hold on the entire subsequent path. \( F \) reads as “finally” (also can be written as <>,), \( F \phi \) means that event \( \phi \) eventually has to hold (somewhere on the subsequent path).

According to the fairness definition and LTL logic, we assume that only signer \( P_i \) is honest, and the description of fairness of signer \( P_i \) is given in formula (2). It means that, when the protocol is terminated, if there is a reachable trace \( \tau \) in which a signer in \( P_1, ..., P_{i-1}, P_{i+1}, ..., P_n \) can receive \( P_i \)'s digital signature \( S_{Pi}(m) \), then there must exist \( \tau' \) which is reachable from \( \tau \) and can make \( P_i \) receive the digital signature of \( P_1, ..., P_{i-1}, P_{i+1}, ..., P_n \):

\[
\langle \rangle \langle \rangle (P_i \cdot S_{Pi}(m) \lor \cdots \lor P_{i-1} \cdot S_{Pi-1}(m) \\
\lor P_{i+1} \cdot S_{Pi+1}(m) \lor \cdots \lor P_n \cdot S_{Pn}(m)) \\
\rightarrow \langle \rangle (P_i \cdot S_{Pi}(m) \land \cdots \land P_{i-1} \cdot S_{Pi-1}(m) \\
\land P_{i+1} \cdot S_{Pi+1}(m) \land \cdots \land P_n \cdot S_{Pn}(m))
\]

\( (2) \)

3.5. State Reduction Algorithm. The model checking method has many advantages, such as, high level automation and exact description ability. But the idea of this method is based on exhausted state space searching. When we use it to verify some concurrent systems, the state space may be increased exponentially. That is the space explosion problem. Towards the space explosion problem, a detailed state reduction algorithm is demonstrated below and the algorithm is shown in Algorithm 1.

(a) Deleting states that system cannot reach to decrease the searching time.

(b) Deleting states that will absolutely lead to fairness which can make the track of attack trace more conveniently.

(c) Combing transition relations which reach the same next states to compress the state space, that is, \( A \lor B \) \( \rightarrow \) \( \text{trans} \) \( \Rightarrow \) \( \text{trans} \). Here \( A \) and \( B \) are states and \( \rightarrow \) is transition function. The proof can be found in “Hoare Logic,” rules of consequence.

(d) Letting the dishonest signers send the highest promises to each other, for the transaction between dishonest signers will not influence the fairness of the honest one. Such behaviors can cut down the number of messages between the dishonest signers and then can reduce the state space.

\( \text{trans} \) is an array which stores the transition relations in the OMPCS model, and every transition relation in \( \text{trans} \) contains two states as well as a variable; state 1 is the current state, state 2 is the next state, and \( \text{trans} \).mark is the weight of states. If a state in a protocol model has been reached for \( m \) times, the value of mark for such state is \( m \). The variable \( \text{exishon}(i) \) is a function to judge whether, after \( P_i \) contacting \( T \), there also exists an honest signer who has no contract \( T \), validate(\( \text{trans}[i] \)) = true means that the transition relation \( \text{trans}[j] \) contracted \( T \), and \( T \) sent a recovery message to reply to it; otherwise, \( T \) sent an abort message.

Line 1 to line 2 are corresponding to (a) and (b), according to the recovery subprotocols in OMPCS; if validate = true and the honest signer have not contracted TTP before, this protocol will be fair. Lines 4 to 7 mean (c); if \( \text{trans}[i] \) and \( \text{trans}[j] \) have the same next state (state2), \( \text{trans}[i] \).state1 will be combined with \( \text{trans}[j] \).state1, and \( \text{trans}[i] \) will be deleted.
Procedure StateReduction(trans)
(1) for $i \leftarrow 1$ to $\text{length}(trans)$
(2) do if (validate(trans[$i$]) = 1 && exishon($i$))($[$trans[$i$]].mark = 0)
(3) then delete(trans[$i$])
(4) for $j \leftarrow 1$ to $\text{length}(trans)$
(5) do if ($trans[i].state2 = trans[j].state2$) && ($i = j$)
(6) then $trans[j].state1 \leftarrow trans[i].state1 + trans[j].state1$
(7) delete(trans[$i$])
(8) for $i \leftarrow 1$ to $n$
(9) do for $j \leftarrow 1$ to $n$
(10) do if ($pi, pj \in \text{dis}$) && ($i \neq j$)
(11) then $p_{i,j} \leftarrow n$
(12) end

Algorithm 1: The state reduction algorithm.

Modeling the behaviors of honest $P_1$ in the four-party GM main subprotocol

$P_1H\_\text{process}()$

\[\ldots\]

//(1) honest $P_1$ sends 1-level promises to $P_2$
\[
\begin{align*}
\text{[!]P1\_stop && !P1\_contacted,T && Pr_1\_1L==0 && Pr_1\_3L==0 && Pr_2\_1L==1 && &
\text{Pr_3\_1L==1 && Pr_4\_1L==1]} & \text{P1\_sendP2}[Pr_1\_1L=1]} & \text{->P1H\_process()} \end{align*}
\]

//(2) honest $P_1$ sends recovery requirement to $T$
\[
\begin{align*}
\text{[!]P1\_stop && !P1\_contacted,T && Pr_1\_1L==0 && Pr_1\_3L==0 && Pr_2\_1L==1 && &
\text{Pr_3\_1L==1]} & \text{P1\_Cont{'}acted,T=true; P1\_Recovery_1L=1]} & \text{->T\_process()} \end{align*}
\]

Modeling the behaviors of dishonest $P_2$ in the four-party GM main subprotocol

$P_2\_process()$

\[\ldots\]

//(3) dishonest $P_2$ sends 1-level promise to $P_1$
\[
\begin{align*}
\text{[!]P2\_stop && Pr_2\_1L<1]} & \text{P2\_sendP32}[Pr_2\_1L=1]} & \text{->P2\_process()} \end{align*}
\]

//(4) dishonest $P_2$ sends recovery requirement to $T$
\[
\begin{align*}
\text{[!]P2\_stop && Pr_2\_1L<1 && Pr_1\_1L==1 && &
\text{Pr_3\_1L==1]} & \text{P2\_recovery311}[P1\_contacted,T=true; P1\_Recovery_1L=1]} & \text{->T\_process()} \end{align*}
\]

System definition

sys1H = $P1H\_process() \| P2\_process() \| P3\_process() \| P4\_process();$

Algorithm 2: Modeling of four-party GM main subprotocol.

$T\_\text{process}()$

Modeling of the four-party GM abort subprotocol

\[\ldots\]

//(1) $T$ agrees with the abort requirement from $P_2$
\[
\begin{align*}
\text{[!]T\text{\_Respond2} && P2\_Abort\_Send && !T\_Validated && (T\_S4 || T\_S3)} & \text{T\_Abort\_Send,P2=true; T\_Respond2=true]} & \text{-P2H\_process()} \end{align*}
\]

//(2) $T$ refuses the abort requirement from $P_2$
\[
\begin{align*}
\text{[!]T\text{\_Respond2} && P2\_Abort\_Send && !T\_Validated]} & \text{T\_Abort\_Send,P2=true; T\_Respond2=true]} & \text{-P2H\_process()} \end{align*}
\]

Modeling of the four-party GM recovery subprotocol

\[\ldots\]

//(3) $T$ agrees with the recovery requirement from $P_3$
\[
\begin{align*}
\text{[!]P3\_recovery1L=1]} & \text{T\_Respond4 && T\_Respond3 && T\_Respond1]} & \text{T\_recoveryP31}[T\_recovery\_Send,P3=true; T\_Respond3=true]} & \text{-P3\_process()} \end{align*}
\]

//(4) $T$ refuses the recovery requirement from $P_3$
\[
\begin{align*}
\text{[!]P3\_recovery1L=1]} & \text{T\_Respond3 && T\_Respond4]} & \text{T\_recoveryP31}[T\_recovery\_Send,P3=true; T\_Respond3=true]} & \text{-P3\_process()} \end{align*}
\]

Algorithm 3: Modeling of four-party GM abort and recovery subprotocols.
process();
#definegoals1(P4...)
#definegoals2(P1S2&&P1S3&&P1S4);
#definegoals3(P2S1&&P2S2);
#definegoals4(P3S1&&P3S2);
#definegoals5(P4S1&&P4S2);
#definegoals6(P5S1);
#definegoals7(P6S1);

sys1H= P1H_process() ||| P2_process() ||| P3_process() ||| P4_process();
#definegoals1(P4...)
#definegoals2(P1S2&&P1S3&&P1S4);
#definegoals3(P2S1&&P2S2);
#definegoals4(P3S1&&P3S2);
#definegoals5(P4S1&&P4S2);
#definegoals6(P5S1);
#definegoals7(P6S1);

T_process()=
Modeling of the five-party CKS abort subprotocol
.....

// (1) T agrees with the abort requirement from P2
[[ !T_Respond2 && P2_Abort_Send && !T_Validated ] TP2abort1 [ T_S2 = true; T_Abort_Send_P2 = true; T_Respond2 = true; T_h2 = 0; T_j2 = 1 ] P2H_process()
// (2) T refuses the abort requirement from P2
[[ !T_Respond2 && P2_Abort_Send && !T_Validated ] TP2abort2 [ T_S2 = true; T_Recovery_Send_P2 = true; T_Respond2 = true ] P2H_process()

Modeling of the five-party CKS recovery subprotocol
.....

// (3) T agrees with the recovery requirement from P3
[[ P3_recovery_1_1_3_3 && T_Respond3 && !T_Respond4 && !T_Validated && !T_Respond2 && !T_Respond3 ] P3reco1 [ T_Recovery_Send_P3 = true; T_Respond3 = true; T_Validated = true ] P3_process()
// (4) T refuses the recovery requirement from P3
[[ P3_recovery_1_1_3_3 && T_Respond3 && ( T_Respond5 || T_Respond4 ) ] T_Respond3 ] P3reco3 [ T_Recovery_Send_P3 = true; T_Respond3 = true; T_Abort_Send_P3 = true; T_h3 = 3; T_j3 = 3 ] P3_process()

Algorithm 4: Modeling fairness of $P_1$.

Algorithm 5: Modeling of five-party CKS abort and recovery subprotocols.

From lines 8 to 11, $n$ is the number of signers, $i$, $j$ here are the unique ID for signatories, and $dis$ are the sets for the dishonest signatories. In line 10, if $P_i$ and $P_j$ are dishonest signatories, then they will send the highest promise to each other and it is the pseudocode of (d).

4. Case Study

As an OMPCS protocol, the GM protocol was proposed by Garay and Mackenzie in [14]; then in [5] Asokan modified it and proposed the CKS protocol. In this section, the two protocols are modeled and analyzed on the platform PAT [15] and fairness flaws have been discovered.

4.1. GM and CKS. Each of the GM and CKS protocols has three subprotocols: main subprotocol, recovery subprotocol, and abort subprotocol. Such protocols use zero-knowledge primitives, private contract signatures [14]. The main subprotocol of the two protocols is the same, and major changes are in the recovery and abort subprotocols.

The main subprotocol for $n$ signers is divided into $n$-level recursions with $n$-level promises. $P_i$ sends $i$-level promise to $P_j$ which can be denoted as $PCS_p((m,i),P_j,T)$. The third party is $T$; if there is nothing wrong in the execution of main subprotocol, $T$ will not be invoked. Conversely, requirement messages will be sent by signers to $T$ to guarantee fairness. For $P_j$ to abort, it will send the abort message to $T$; for $P_j$ to recover, it will send the corresponding recovery message. The messages are designed so that $T$ can infer the promises that an honest signer would have sent when it launched the recovery subprotocol. For lack of space, we will not describe the two protocols and the details can be found in [6, 14].

4.2. Modeling GM Protocol. We have modeled and analyzed the GM protocol for three cases, and here we take the case of four parties for example. Figure 2 describes the communications between signers, third party, and channels. We assume that the honest signer is $P_1$, $P_2$, $P_3$, and $P_4$ are dishonest, and $P_2$, $P_4$ can collude to cheat $P_1$. $P_1H_process()$ was defined as the behavior of $P_1$ and $P_2$-$process()$, $P_3$-$process()$, $P_4$-$process()$, and $T$-$process()$ as the behaviors of $P_2$, $P_3$, $P_4$, and $T$. After doing that, the GM protocol was modeled as a parallel system called “sys1H.”

4.2.1. Modeling GM Main Subprotocol. Main subprotocol is executed when signers exchange their promises. When $n = 4$, the main subprotocol has 4 levels (see the OMPCS definition in Section 2) recursions. We use integer variables to describe promises between signers, and boolean variables represent messages between signers and $T$. The details of the main subprotocol model are shown in Algorithm 2.

The honest $P_1$ mainly performs two kinds of actions in the main subprotocol, which includes sending promises to other signers and sending requirement to $T$. They are described in step (1) and step (2). Step (1) models the action of sending $i$-level promise, in which we use boolean variables,
such as \(Pr_1\_2\_L\), to represent the promises exchanging. Setting \(Pr_1\_2\_L = 1\) means \(P_1\) has successfully sent 1-level promise to \(P_2\). Step (2) says that if \(P_1\) has not received the correct promises, he can set \(P_2\), \(Recovery\_1\_1\_1\) as true, which represents the action of sending out recovery requirement. Step (3) and Step (4) describe the malicious behaviors of dishonest \(P_2\). Step (3) models that \(P_2\) can send 1-level promise to \(P_1\) and Step (4) specifies that \(P_2\) can send recovery requirement to \(T\) at a relatively relaxing condition.

4.2.2. Modeling GM Abort and Recovery Subprotocol. Algorithm 3 models the actions of the \(T\), that is, the abort and recovery subprotocols. \(T\) is a special player that has to be modeled in a particular way. The definition and grammar are the same as the main subprotocol. \(T\), \(process()\) maintains two sets, \(S(m)\) and \(F(m)\), which are initialized as empty. When \(P_1\) sends abort message to \(T\), \(S(m)\) is set to be \(S(m) = S(m) \cup \{i\}\). Elements inside \(F(m)\) are those signers who have not sent abort message to \(T\). Based on the values of \(S(m)\) and \(F(m)\), the recovery or abort decision will be made by \(T\).

The actions of \(T\) can be divided into two parts, the first part describes how \(T\) deals with abort request from \(P_2\). \(T\) sends out abort token to \(P_2\) if the status is that \(T\) _Validated is false and \((T\_S4 || T\_S3)\) is true. However, if \(T\) _Validated is true, this means the recovery message has already been sent. Then the abort request is to be refused. Part two models the behaviors of dealing with recovery requests from \(P_3\); if \(T\) _Validated is true, the recovery message will be sent. However, conversely, \(T\) will make decisions based on the current conditions.

4.3. Modeling GM Fairness. The fairness of \(P_1\) can be divided into two parts. The first is that when GM protocol is finished, if \(P_1\) has not received the contract signed by other signers, then every other signer also will not receive the contract signed by \(P_1\). The remainder is that if any other signers have received the contract signed by \(P_1\), then \(P_1\) must have received the contract signed by other signers as well. So the LTL modeling of \(P_1\)’s fairness is illustrated in Algorithm 4.

Goal 1 means that at least one signer in \(P_2\), \(P_3\), and \(P_4\) has received the contract signed by \(P_1\). Meanwhile, goal 2 means that \(P_1\) has received the contract signed by \(P_2\), \(P_3\), and \(P_4\). The fairness of \(P_1\) can be described as \(sys[H] = G((goals\_11) \rightarrow F(goals\_2))\), which means that, for all the traces of GM modeling system, if there is one trace to make one of \(P_1\), \(P_3\), and \(P_4\) receive the signed contract from \(P_1\), there must exist another trace that can guarantee that \(P_1\) receives the signed contract from \(P_2\), \(P_3\), and \(P_4\).

4.4. Modeling CKS Protocol. The model of CKS main subprotocol and the description of fairness remain the same with the GM protocol, so this section will concentrate on the abort and recovery part. The main difference 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 signers that contracted \(T\) in the past. Compared with GM, \(T\) in CKS also maintains two integer variables, but the difference is that the variables have different meanings than that in GM.

We modeled cases of four and five signers of the CKS protocol and a fairness flaw was found in the case of five. For the sake of contrastive analysis, specifics of \(P_3\) abort request and \(P_3\), \(Recovery\_1\_3\_3\_3\) recovery requirements are shown in Algorithm 5.

4.5. State Reduction. When modeling GM and CKS protocol, we mitigate the state space explosion problem based on the algorithm proposed in Section 3.5. The detailed examples are given as below.

(a) Deleting states which cannot be reached: for instance, in the GM protocol, when \(T\) is dealing with the requirement \(P_2\), \(Recovery\_1\_3\_2\) from \(P_2\), there is a state which requires that \((T\_S3 \land T\_S4)\) and two elements in set \(\{T\_S1, T\_S3, T\_S4\}\) should be true. Then a conclusion can be reached that this state is unreachable, for the state constraint must be \((!T\_S3 \land T\_S4) \land (T\_S3 \lor T\_S4)\), which is contradictory. Therefore, we can delete this state.

(b) Deleting states which will absolutely lead to fairness: when modeling the GM protocol, if \(P_4\) sends \(T\) a recovery requirement \(P_4\), \(Recovery\_3\_3\_3\) in the condition that no one has contacted \(T\) before, \(T\) will absolutely agree with \(P_4\)’s requirement, setting \(T\) _Validated = 1. It is obvious to ensure that the fairness can be guaranteed for, in the recovery subprotocol, \(T\) cannot overturn the recovery decision. Therefore, this state can be ignored when we are searching for attacking trace.

(c) Combining transition relations which have the same next states: in the GM protocol, when \(T\) deals with the requirement \(P_1\), \(Recovery\_4\_A\_4\_4\) from \(P_1\), there are two transition relations: one of the current states is \(T\_F3 \land \neg T\_F2 \land \neg T\_F4\) and the other is \(!T\_F3\). However, those two states can reach the same next state; thus, they can be combined into one state \((T\_F3 \land T\_F2 \land \neg T\_F4) \lor \neg T\_F3\).

(d) Letting the dishonest signers send the highest promises to each other: when we model the dishonest signers \(P_2\) and \(P_3\) in the CKS protocol, some of
the intermediate states such as \( \text{Pr}_2.3.L = 3 \) and \( \text{Pr}_3.2.L = 2 \) can be deleted, letting \( P_1 \) and \( P_3 \) send the highest promise to each other directly, \( \text{Pr}_2.3.L = 4 \) and \( \text{Pr}_3.2.L = 4 \).

4.6. Analysis. One feature of the OMPCS protocol is that the number of participants can be varied and the structure is not symmetric. That is why the specification of \( P_i \) is different from that of \( P_j \) \((i \neq j)\). A number of CSP# models have been written in PAT for different signers for the GM and CKS protocols. We found that GM protocol cannot satisfy fairness for the case of \( n \geq 4 \), because we can verify that the protocol is not fair in case of four, and, in more signatories cases, if there is only one honest, then the dishonest can collaborate to cheat just as they do in the case of 4. The CKS protocol is not fair for the number of signers \( n \geq 5 \), the reason is the same for GM.

Table 3 shows one possible error trace for the GM protocol \((P_1 \) is the honest one and \( n = 4 \)). State 0 is the initialized state. In state 1, \( P_3 \) contacts \( T \) by sending an abort requirement. According to [14], \( T \) agrees with this requirement and sends abort message, setting \( T_S3 = \) true. But the dishonest \( P_3 \) continues to execute the main subprotocol. Then \( P_3 \) contracts \( T \) with a recovery requirement. \( T \) refuses \( P_3 \)'s requirement based on the abort subprotocol with \( T.F1 = \) true, \( T.S2 = \) true. However, the dishonest \( P_3 \) continues to execute the main subprotocol. In the next state \( P_4 \) launches a resolve request to \( T \), however \( T.F1 \) is true and it indicates that \( T \) will refuse \( P_1 \)'s requirement and update \( T.S1 = \) true. Then the honest signer \( P_1 \) quits the main subprotocol. At last, \( P_4 \) sends a recovery requirement. For \( T.S4 \) is false, \( T \) overturns the previous decision and agrees with \( P_3 \)'s requirement. Hence the fairness of \( P_1 \) are violated. We also found unfairness trace when \( P_3 \) or \( P_2 \) is the dishonest signer.

Table 4 shows one possible unfairness trace for the CKS protocol \( (P_1 \) is the honest one and \( n = 5 \)). State 0 is also the initialized state. In step 1, \( P_5 \) sends an abort requirement to \( T \). \( T \) agrees with this requirement and sends back an abort message with \( T.J5 = \) 1, \( T.S5 = \) true. But the dishonest \( P_3 \) continues to execute the main subprotocol. Then \( P_3 \) contract \( T \) with a recovery requirement. \( T \) refuses \( P_3 \)'s requirement and sets \( T.H1 = 4, T.S1 = \) true, the dishonest \( P_3 \) continues to execute the main subprotocol. Then \( P_4 \) launches a recovery request to \( T \). \( T \) refuses \( P_1 \)'s requirement with an
4.7. Experiment Comparison. In this paper, the fairness for four-party GM protocol and five-party CKS protocol has been verified. There are certain advantages of our method, such as less time and space complexity, more precise semantic, higher degree of automation, and more detailed results. The time and space consuming comparison between our method and the method in [5] is showed in Table 5; the unit for time is second and for memory is KB.

We use a common environment for all the tests discussed in this section; the hardware environment for the two methods is the same. For [5], the model checking platform is cmocha and the operating system is Ubuntu. For our method, the model checking platform is PAT3, and the operating system is Windows 7.

As Table 5 demonstrated, although the problem of protocol fairness verification is a NP problem and the states and time are expected to increase exponentially in theory, our method can get the verification result in certain times in the three and four participants cases of the CKS protocol. Therefore, we can get a conclusion that the reduction algorithm did a good performance as no error trace was found in such cases, and every state in the state space was visited. On the contrary, the method in [5] consumed more time and memory. Specifically, in the four-party occasion, in our experiment environment, this method cannot get verification result in certain hours (we had waited for more than 12 hours). Moreover, this method cannot give explicitly a number of visited states and fairness counterexamples. The main reason is that our method has less system states, higher states compression ratio, and the trace back-track algorithm.

5. Conclusion and Future Work

Based on modeling language CSP# and linear temporal logic, an efficient method which aims to analyze the fairness of optimistic multiparty contract signing protocols is presented in this paper. In order to demonstrate the feasibility of our method, two examples of the GM and CKS protocol have been described and several fairness attacking traces have been found. Comparisons also have been made in this paper between our method and other traditional methods. And the result shows that our method has certain strengths such as higher automation, less time and space complexity, visualized attacking trace, and better utility.

There is no explicit model for any cryptographic primitives in this paper and the attack model is weaker than the original paper. The main challenge is to verify the OMPCS protocols fairness in a more general condition which accounts for cryptographic and a more relaxed communication model. In our next work, we plan to extend the automatic cryptographic protocol verifier ProVerif [16] to suit our fairness verification method.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was supported in part by the National Science Foundation of China (nos. 91118003, 61272106, and 61003080).

References


Submit your manuscripts at
http://www.hindawi.com