Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 983204, 10 pages
http://dx.doi.org/10.1155/2014/983204

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

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 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 $P_1, \ldots, P_i, \ldots P_j, \ldots P_n$ 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. $\mathrm{PCS}_{P_1}(m, P_2, T)$ (T is short for TTP) denotes the promise signed by P_1 and sent to P_2, and [10]

(a) $\mathrm{PCS}_{P_1}(m, P_2, T)$ is generated by P_1;

(b) P_2 can forge $\mathrm{PCS}_{P_1}(m, P_2, T)$ but can be distinguished by P_1, P_2, and T;

(c) only P_1 and T can transfer $\mathrm{PCS}_{P_1}(m, P_2, T)$ into a digital signature $S_{P_1}(m)$ which can be universally verified.

Definition 2 (weak fairness). For a protocol Γ, 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 Γ 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 $S_{P_j}(m)$ and $S_{P_i}(m)$ from each other

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

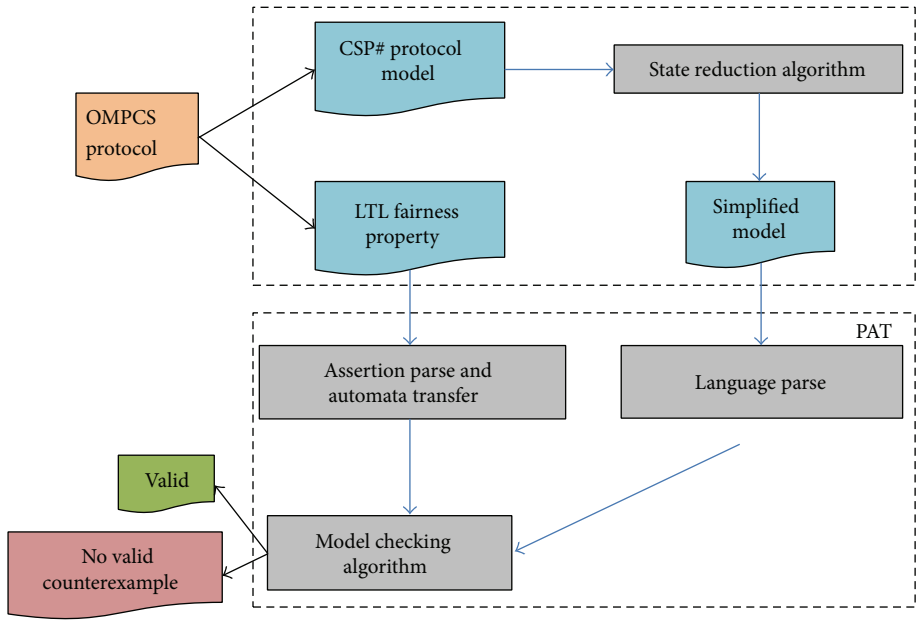

Figure 1: Structure of the method.

Table 1: Syntax of the CSP#.

Stop	Processes do nothing
Skip	Successful termination
$P[\ ]Q$	General choice
$P[*]$	External choice
$P\langle\ \rangle Q$	Internal choice
$P \mid\mid\mid Q$	Interleaving
$P \mid\mid Q$	Parallel composition
If (cond){P} else {Q}	Conditional choice
$e\{prog\} \rightarrow P$	Event prefixing
$P \setminus X$	Hiding all occurrences of event in X
$[cond]P$	Guarded process
$P;Q$	Sequential composition
$ch!exp \rightarrow P$	Channel output
$ch?x \rightarrow P$	Channel input

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.

3.3. 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.

(iii) $R : S \times \sum\tau \times S$ is the transition relation, $\sum\tau$ is a set of all events, $\sum *$ 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_iH_process, which describes how honest signer P_i behaves and the dishonest P_j has a corresponding process P_j_process. T_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:

$$\text{sys} = P_iH\_\text{Process} \mid\mid\mid P_j\_\text{process} \quad (i, j \in P). \tag{1}$$

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, ρ is a LTL formula;

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

(c) if φ, ϕ are two LTL formulae, then $\neg\varphi$, $\varphi \vee \phi$, $\varphi \wedge \phi$, $G\varphi$, and $F\ \varphi$ are all LTL formulae;

Table 2: System variables.

$P_r\_i\_j$	$P_r\_i\_j = k$, if P_i has successfully sent the k level promise to P_j
$P_i\_Recovery\_j$ $P_i\_Recovery\_j\_t$ $P_i\_Recovery\_j\_t\_m$	P_i sends recovery requirement message to TTP, j is the max level of message P_j has sent to P_i, t is the max level of message P_t has sent to P_i, and m is the max level of message P_t has sent to P_i, and so on
P_i_stop	P_i quits the protocol
$P_i\_contacted\_T$	P_i has sent recovery or abort requirement messages to TTP
P_i_Sj	P_i has successfully received the P_j signed contract
$P_i\_AbortTaken$	P_i sends abort requirement message to TTP
$T\_Respondi$	TTP has responded to the requirement of P_i
$T\_Recovery\_send\_P_i$	TTP sends recovery message to P_i
$T\_Abort\_send\_P_i$	TTP sends abort message to P_i
T_Fi	P_i has not sent abort message and TTP will force him/her to abort in certain situation
T_Si	P_i has sent abort message
T_hi	The highest level P_i has sent to higher signer before it contacts TTP
T_li	The lowest level P_i has sent to lower signer before it contacts TTP
T_ki	The highest level P_i has received from P_j $(i < j)$ The highest level P_i has received from all signers m, $(m, j < i)$

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

Here φ and ϕ are two formulae, G reads as "global" (also can be written as []), and $G\varphi$ means that event φ has to hold on the entire subsequent path. F reads as "finally" (also can be written as <>), $F\varphi$ means that event φ 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 τ in which a signer in $P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_n$ can receive P_i's digital signature $S_{P_i}(m)$, then there must exist τ' which is reachable from τ and can make P_i receive the digital signature of $P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_n$:

$$\begin{aligned}\langle\rangle\,[]\,\Big(\Big(&P_1 \cdot S_{P_i}(m) \vee \cdots \vee P_{i-1} \cdot S_{P_i}(m)\\ &\vee P_{i+1} \cdot S_{P_i}(m) \vee \cdots \vee P_n \cdot S_{P_i}(m)\Big)\\ &\longrightarrow []\,\langle\rangle\,\Big(P_i \cdot S_{P_1}(m) \wedge \cdots \wedge P_i \cdot S_{P_{i-1}}(m)\\ &\wedge P_i \cdot S_{P_{i+1}}(m) \wedge \cdots \wedge P_i \cdot S_{P_n}(m)\Big)\Big).\end{aligned} \tag{2}$$

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, $A\{R\}P$ and $B\{R\}P$, we can combine the two transition relations into $(A \vee B)\{R\}P$. Here A, B, and P are states and R 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.

trans is an array which stores the transition relations in the OMPCS model, and every transition relation in trans contains two states as well as a variable; state 1 is the current state, state 2 is the next state, and 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 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(trans[i]) = true means that the transition relation trans[i] 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 trans[i] and trans[j] have the same next state (state2), trans[i].state1 will be combined with trans[j].state1, and trans[i] will be deleted.

```
Procedure StateReduction(trans)
(1)  for i ← 1 to length(trans)
(2)    do if (vaildate(trans[i]) = 1 && exishon(i)))||(trans[i].mark = 0)
(3)          then delete(trans[i])
(4)         for j ← 1 to length(trans)
(5)             do if (trans[i].state2 = trans[j].state2) && (i! = j)
(6)                 then trans[j].state1 ← trans[i]state1 + trans[j].state1
(7)                       delete(trans[i])
(8)  for i ← 1 to n
(9)    do for j ← 1 to n
(10)           do if (pi, pj ∈ dis) && (i! = j)
(11)                then p_i_j ← n, p_j_i ← n
(12) end
```

Algorithm 1: The state reduction algorithm.

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

```
P1H_process()=
. . . . . .
//(1) honest P1 sends 1-level promises to P2
[][!P1_stop && !P1_contacted_T && Pr_1_4_L==0 && Pr_1_3_L==0 && Pr_1_2_L==0 && Pr_2_1_L==1 && Pr_3_1_L==1 &&
Pr_4_1_L==1]P1sendsp21{Pr_1_2_L=1;}->P1H_process()
//(2) honest P1 sends recovery requirement to T
[] [!P1_stop && !P1_contacted_T && Pr_1_4_L==0 && Pr_1_3_L==0 && Pr_1_2_L==1 && Pr_2_1_L==1 && Pr_4_1_L==1 &&
Pr_3_1_L==1]p1recovery111{P1_contacted_T=true; P1_Recovery_1_1_1=true;}->T_process()
```

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

```
P2_process()=
. . . . . .
//(3)dishonest P2 sends 1-level promise to P1
[] [!P2_stop && Pr_2_1_L<1] P2sendP32 {Pr_2_1_L=1;}->P2_process()
//(4)dishonest P2 sends recovery requirement to T
[] [!P2_stop && Pr_4_2_L==1 && Pr_3_2_L==1 && Pr_1_2_L==1]P2recovery111{P2_Recovery_1_1_1=true;}
->T_process()
```

System definition

```
sys1H|=P1H_process() ||| P2_process() ||| P3_process() |||P4_process();
```

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

```
T_process()=
```

Modeling of the four-party GM abort subprotocol

```
. . . . . .
//(1)T agrees with the abort requirement from P2
[][!T_Respond2 && P2_Abort_Send && !T_Validated && ( T_S4 || T_S3)]TabortP21{T_S2=true; T_Abort_Send_P2=true;
T_Respond2=true;}->P2H_process()
//(2)T refuses the abort requirement from P2
[][!T_Respond2 && P2_Abort_Send && T_Validated ]TabortP22{T_S2=true; T_Recovery_Send_P2=true; T_Respond2=true;}-
>P2H_process()
```

Modeling of the four-party GM recovery subprotocol

```
. . . . . .
//(3)T agrees with the recovery requirement from P3
[] [P3_Recovery_1_3_3 && !T_Respond4 && !T_Respond3 && !T_Respond2 && !T_Respond1]TreocveryP31
{T_Recovery_Send_P3=true; T_Respond3=true; T_Validated=true;}->P3_process()
//(4)T refuses the recovery requirement from P3
[] [P3_Recovery_1_3_3 && !T_Respond3 && (!T_Respond4 || !T_Respond3 || !T_Respond2 || !T_Respond1)&&
!T_Validated && T_S4]TreocveryP312{T_F1=true;T_F2=true;T_S3=true;T_Abort_Send_P3=true;
T_Respond3=true}->P3_process()
```

Algorithm 3: Modeling of four-party GM abort and recovery subprotocols.

```
sys1H= P1H_process() ||| P2_process() ||| P3_process() |||P4_process();
#define goals1 (P4_S1 || P3_S1 || P2_S1);
#define goals2 (P1_S2 && P1_S3 && P1_S4);
#assert sys1H |= G ((goals1) -> F(goals2));
```

Algorithm 4: Modeling fairness of P_1.

```
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_l2=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_Respond5 && !T_Respond4 && !T_Respond3 && !T_Respond2 && !
T_Respond1] 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||T_Respond2||
T_Respond1)&& !T_Validted && ((T_S5 && T_l5>0 ||T_S4 && T_l4>0 || T_S2 && T_h2>2||T_S1 &&
T_h1>2 ))]P3reco3{T_Respond3=true;T_S3=true;T_Abort_Send_P3=true;T_h3=3;T_l3=3}->P3_process()
```

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_1 sends i-level promise to P_2 which can be denoted as $\mathrm{PCS}_{P_1}((m,i),P_2,T)$. 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_i to abort, it will send the abort message to T; for P_i 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_3, and 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 1-level promise, in which we use boolean variables,

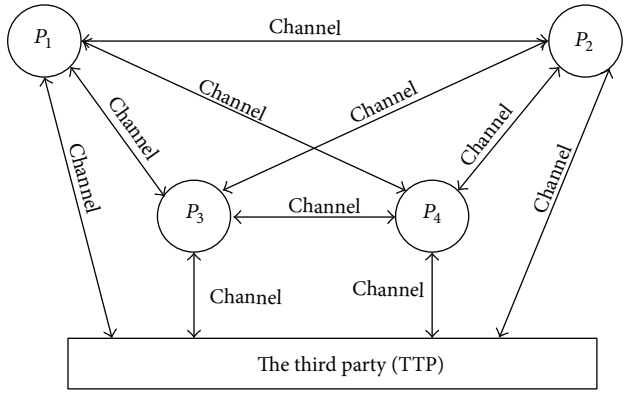

Figure 2: Communications between signers, third party, and channels.

such as Pr_1_3_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_i 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 sys1H| = G((goals 1) $\rightarrow$ F(goals 2)), which means that, for all the traces of GM modeling system, if there is one trace to make one of P_2, 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_2 abort request and P_3_Recovery_1_1_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_$S3$∧!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_$S3$∧!T_$S4$) ∧ (T_$S3$ ∨ 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_3_Recovery_4_4_4 from P_3there are two transition relations: one of the current states is T_$F3$∧!T_$F2$∧!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_$F3$∧!T_$F2$∧!T_$F4$)∨!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

Table 3: The unfairness trace of the four-party GM protocol.

GM protocol	State 0	State 1	State 2	State 3	State 4
P_4__Recovery_3_3_3	False	False	False	False	True
P_3__abort	False	True	True	True	True
P_2__Recovery_1_1_2	False	False	True	True	True
P_1__Recovery_1_3_3	False	False	False	True	True
T_Abort_Send_P_4	False	False	False	False	False
T_Recovery_Send_P_4	False	False	False	False	True
T_Abort_Send_P_3	False	True	True	True	True
T_Abort_Send_P_2	False	False	True	True	True
T_Abort_Send_P_1	False	False	False	True	True
goals1	False	False	False	False	True
goals2	False	False	False	False	False
Fairness	YES	YES	YES	YES	NO

Table 4: The unfairness trace of the five-party CKS protocol.

Revised GM protocol	State 0	State 1	State 2	State 3	State 4	State 5
P_5_abort	False	True	True	True	True	True
P_4__Recovery_5_4_4_4	False	False	False	True	True	True
P_3__Recovery_6_6_5_5	False	False	False	False	False	True
P_2__Recovery_5_5_5_5	False	False	False	False	True	True
P_1__Recovery_1_4_4_4	False	False	True	True	True	True
T_Abort_Send_P_5	False	True	True	True	True	True
T_Abort_Send_P_4	False	False	False	True	True	True
T_Abort_Send_P_3	False	False	False	False	False	True
T_Abort_Send_P_2	False	False	False	False	True	True
T_Abort_Send_P_1	False	False	True	True	True	True
goals1	False	False	False	False	False	True
goals2	False	False	False	False	False	False
Fairness	YES	YES	YES	YES	YES	NO

the intermediate states such as Pr_2_3_L = 3 and Pr_3_2_L = 2 can be deleted, letting P_2 and P_3 send the highest promise to each other directly, Pr_2_3_L = 4 and 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_2 contract T with a recovery requirement. T refuses P_2's requirement based on the abort subprotocol with T_F1 = true, T_S2 = true. However, the dishonest P_2 continues to execute the main subprotocol. In the next state P_1 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_4'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_3 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_l5 = 1, T_S5 = true. But the dishonest P_5 continues to execute the protocol. In state 2, P_1 contract T with a recovery requirement. T refuses P_1's requirement and sets T_h1 = 4, T_S1 = true, the dishonest P_1 continues to execute the main subprotocol. Then P_4 launches a recovery request to T. T refuses P_4's requirement with an

Table 5: Experiment comparison.

	This paper method		Reference [5] method	
	Time used	Memory used	Time used	Memory used
GM protocol				
$n = 2$	0.01	41179	0.050	51228
$n = 3$	10.64	72056	38.72	92595
$n = 4$	505.21	2479154	—	—
CKS protocol				
$n = 2$	0.01	39187	0.047	4568
$n = 3$	8.29	60145	19.54	86347
$n = 4$	552.31	2787336	—	—
$n = 5$	1674.05	3105836	—	—

abort message, setting $T\_l4 = T\_h4 = 5$, $T\_S4 =$ true. In the following states, P_2 send recovery messages to T and T will refuse it and update $T\_l2 = T\_h2 = 5$, $T\_S2 =$ true. P_3 sends its signed contract to P_4 and P_5, but the dishonest P_4 and P_5 quits the protocol and P_3 contracts T with recovery requirement.

However, according to [5] and the recovery subprotocol, T computes the value of T_hi and T_ki and finds $(T\_S2) \wedge (T\_k2 \leq T\_h2)$, and then it makes a decision that it will not overturn the previous decision. Finally, T sends an abort message to P_3; thus, the fairness of P_3 is not guaranteed. The similar error traces can also be found when P_4, P_2, or P_1 is the dishonest one.

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

[1] N. Asokan, M. Schunter, and M. Waidner, "Optimistic protocols for fair exchange," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 6–17, New York, NY, USA, April 1997.

[2] S. Even and Y. Yacobi, "Relations among public key signatures systems," Tech. Rep. 175, Technion, Haifa, Israel, 1980.

[3] S. Mauw, S. Radomirović, and M. T. Dashti, "Minimal message complexity of asynchronous multi-party contract signing," in *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF '09)*, pp. 13–25, IEEE Computer Society Press, July 2009.

[4] A. Mukhamedov and M. D. Ryan, "Resolve-impossibility for a contract-signing protocol," in *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW '06)*, pp. 167–173, July 2006.

[5] N. Asokan, *Fairness in electronic commerce [Ph.D. thesis]*, University of Waterloo, 1998.

[6] R. Chadha, S. Kremer, and A. Scedrov, "Formal analysis of multi-party contract signing," in *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW '04)*, pp. 266–279, IEEE Computer Society Press, Washington, DC, USA, June 2004.

[7] Y. Zhang, C. Zhang, J. Pang, and S. Mauw, "Game-based verification of contract signing protocols with minimal messages," *Innovations in Systems and Software Engineering*, vol. 8, no. 2, pp. 111–124, 2012.

[8] A. Mukhamedov and M. D. Ryan, "Fair multi-party contract signing using private contract signatures," *Information and Computation*, vol. 206, no. 2–4, pp. 272–290, 2008.

[9] S. Mauw, S. Radomirović, and M. T. Dashti, "Minimal message complexity of asynchronous multi-party contract signing," in *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF '09)*, pp. 13–25, IEEE CS, July 2009.

[10] A. Mukhamedov, S. Kremer, and E. Ritter, "Analysis of a multi-party fair exchange protocol and formal proof of correctness in the strand space model," in *Financial Cryptography and Data Security*, vol. 3570 of *Lecture Notes in Computer Science*, pp. 255–269, 2005.

[11] N. Zhang, X. Zhang, and Y. Wang, "Formal analysis of GM multi-party contract signing protocol," in *Proceedings of the 2nd International Conference on Convergent Information Technology (ICCIT '07)*, pp. 1316–1321, IEEE Computer Society Press, November 2007.

[12] J. Sun, Y. Liu, S. D. Jin, and C. Chen, "Integrating specification and programs for system modeling and verification," in *Proceedings of the 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE '09)*, pp. 127–135, July 2009.

[13] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS '77)*, pp. 46–57, 1977.

[14] J. A. Garay and P. Mackenzie, "Abuse-free multi-party contract signing," in *Distributed Computing*, vol. 1693 of *Lecture Notes in Computer Science*, pp. 151–165, 1999.

[15] Y. Liu, J. Sun, and J. S. Dong, "PAT 3: an extensible architecture for building multi-domain model checkers," in *Proceedings of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE '11)*, pp. 190–199, Hiroshima, Japan, December 2011.

[16] V. Cheval and B. Blanchet, "Proving more observational equivalences with ProVerif," in *Principles of Security and Trust*, vol. 7796 of *Lecture Notes in Computer Science*, pp. 226–246, Springer, 2013.