Front. Comput. Sci., 2012, 6(1): 57–75
DOI 10.1007/s11704-012-2903-3

RESEARCH ARTICLE

SeVe: automatic tool for verification of security protocols

Anh Tuan LUU[1] (✉), Jun SUN[2], Yang LIU[1], Jin Song DONG[1], Xiaohong LI[3], Thanh Tho QUAN[4]

1 School of Computing, National University of Singapore, Singapore 119077, Singapore
2 School of Computing, Singapore University of Technology and Design, Singapore 138682, Singapore
3 Department of Computer Science, Tianjin University, Tianjin 300222, China
4 Department of Computer Science, HoChiMinhCity University of Technology, HoChiMinh City 162903, Vietnam

Abstract Security protocols play more and more important roles with wide use in many applications nowadays. Currently, there are many tools for specifying and verifying security protocols such as Casper/FDR, ProVerif, or AVISPA. In these tools, the intruder's ability, which either needs to be specified explicitly or set by default, is not flexible in some circumstances. Moreover, whereas most of the existing tools focus on secrecy and authentication properties, few supports privacy properties like anonymity, receipt freeness, and coercion resistance, which are crucial in many applications such as in electronic voting systems or anonymous online transactions.

In this paper, we introduce a framework for specifying security protocols in the labeled transition system (LTS) semantics model, which embeds the knowledge of the participants and parameterizes the ability of an attacker. Using this model, we give the formal definitions for three types of privacy properties based on trace equivalence and knowledge reasoning. The formal definitions for some other security properties, such as secrecy and authentication, are introduced under this framework, and the verification algorithms are also given. The results of this paper are embodied in the implementation of a SeVe module in a process analysis toolkit (PAT) model checker, which supports specifying, simulating, and verifying security protocols. The experimental results show that a SeVe module is capable of verifying many types of security protocols and complements the state-of-the-art security verifiers in several aspects. Moreover, it also proves the ability in building an automatic verifier for security protocols related to privacy type, which are mostly verified by hand now.

Keywords security protocols, model checking, process analysis toolkit (PAT), authentication, secrecy, privacy

Received July 20, 2011; accepted September 30, 2011

E-mail: tuanluu@comp.nus.edu.sg

1 Introduction

With the explosion of the Internet, electronic transactions have become more and more common. The security for these transactions is very crucial to many applications, e.g., electronic commerce, digital contract signing, and electronic voting. However, these large open networks, where trusted and untrusted parties coexist and where messages transit through potentially dangerous environments, pose new challenges to the designers of communication protocols. Properties such as authenticity, confidentiality, proof of identity, proof of delivery, or privacy are difficult to assure in this scenario. Security protocols aim at solving this problem. By a suitable use of shared and public key cryptography, random numbers, hash functions, and encrypted and plain messages, a security protocol may assure security requirements for the participants.

The informal specification of security protocols is usually simple and easy to understand. However, it provides an incomplete description of the actions of principals engaged in the protocol execution. Namely, it describes only the actions taken in a complete protocol run between honest principals. In contrast, it does not describe what happens during unsuccessful runs, for example, with possibly untrusted par-

ticipants. Moreover, even in successful runs, certain checks must be taken and executions, will abort when the checks fail. Thus, the formal methods in specification and verification of security protocols have became the subject of intense research. For instance, methods based on belief logics [1,2], theorem proving with induction [3,4], and state exploration methods [5] have been successfully used to verify and debug security protocols. However, these approaches lack automation and therefore are hard to apply in practical use. In addition, the verification of security protocols is also a real challenge. With the complicated combination and parallellism of protocol sessions, the complexity of verification should be considered. With all of those requirements, there is a demand for automated tools which are able to help to specify and verify security protocols correctly and sufficiently. In this paper, we aim to develop an automatic security verifier based on a framework using the label transition system model and knowledge reasoning to solve this problem.

There has been much research on formally specifying and verifying security protocols such as Casper/FDR [6], ProVerif [7], or AVISPA [8]. Most of these focus on authentication and secrecy properties. Anonymity and privacy properties which take an important role in many protocols get less attention from researchers. The user may require anonymity and privacy guarantees for many transactions, such as anonymity of payment, privacy of shopping preference, or candidate choice in an election. Therefore, there is a demand on an effectively automatic tool to specify and verify security protocols related to those properties.

There are three types of anonymity and privacy-type properties in the literature: anonymity (or untraceability), receipt freeness, and coercion resistance.

- Anonymity: the intruder cannot determine which information was sent by a particular agent.
- Receipt freeness: the agent cannot gain any receipt which can be used to prove to the intruder that an agent sent particular information.
- Coercion resistance: an agent cannot cooperate with the intruder to prove to him that the agent sent particular information.

Of course, an agent can tell an intruder the information he sent but unless the agent provides convincing evidence, the intruder cannot believe him. Receipt freeness and coercion resistance guarantee that an agent cannot provide such evidence.

Recently, some research on verifying privacy properties using applied Pi-calculus [9,10] and epistemic logics [11,12] has been proposed. However, most of these studies require hand proving in the verification, especially in receipt-freeness and coercion-resistance properties. An effective automatic tool to verify security protocols related to privacy properties still poses a challenge. This is also a goal for our study.

The rest of the paper is organized as follows. In Section 2 we discuss related work Operational semantics and model semantics of security protocol are described in Section 3. Section 4 introduces the algorithms used in verification and the architecture of SeVe tool. Two case studies about security verification are demonstrated in Section 5. The experiments and comparison are given in Section 6. Section 7 finally concludes the paper.

2 Related work

There is much research on verification of security protocols. In this section, we will summarize some related works on security verifier tools as well as formalization and verification of privacy properties.

2.1 Security verifier tools

A method for analyzing security protocols using the process algebra CSP [13] has been developed in [6,14]. An advantage of using process algebra for modeling security protocols is that the model is easily extended. This technique has proved successful, and been used to discover a number of attacks upon protocols [15,16]. However, this technique requires the producing of a CSP description of the protocol by hand, which has proven tedious and error-prone. Developed originally by Gavin Lowe, the Casper/FDR tool set, as described in [6], automatically produces a CSP description from a more abstract description, thus greatly simplifying the modeling and analysis. The user specifies the protocol using a more abstract notation, similar to that appearing in academic literature, and Casper compiles this into CSP code suitable for checking using FDR. However, Casper only supplies a few forms of specification for protocols, mostly focusing on authentication and secrecy but not for other security properties such as anonymity and privacy. An intruder's ability is set by default, therefore, the user cannot have a choice when modeling system in different environments.

OFMC [17] and CL-Atse [18] are two tools developed in AVISPA project [8]. Both these tools take the same input language called HLPSL (High Level Protocol Specification Language) [19]. However, the declaration using HLPSL is quite complicated and the transition states for each step need to be

specified, so the specification job is quite tricky. With a large number of participants, OFMC and CL-Atse may not terminate. Like Casper, the user cannot vary the intruder abiliies in verification.

ProVerif [7] is an automatic cryptographic protocol verifier in the formal model (so called Dolev-Yao model). The input of this tool is pi-calculus description [20]. Then this description is translated into Horn clauses for verification. ProVerif can handle many different cryptographic primitives and an unbounded number of sessions of the protocol and message space. However, specifying security protocols using pi-calculus is not an easy task as the user needs to implicitly specify intruder behavior. Also intruder ability cannot be changed.

2.2 Research on privacy verification

The idea of formalizing anonymity as a kind of process equivalence in process algebra is first proposed in the work of Schneider and Sidiropoulos [21]. However, only anonymity checking is introduced in this paper. Fournet and Abadi [22] modeled security protocols using applied pi-calculus and observed the observational equivalence in process calculus to prove the anonymity. Similar idea has been used by Mauw et al. [10], Kremer and Ryan [23]. Again, only anonymity is investigated. The work of Delaune, Kremer, and Ryan [9] gave first formal methods definition of receipt-freeness and coercion-resistance, two other types of privacy properties, in applied pi-calculus. In this approach, the authors use forward channel and bisimulation to capture the condition for these two privacy properties, while in our approach, we use knowledge based reasoning and trace equivalence to define the condition for them. Reasoning about bisimulation in this approach is rather informal and mainly done by hand. Backes et al. [24] present a general technique for modeling remote voting protocols in the applied pi-calculus by giving a new definition of coercion-resistance in terms of observational equivalence. However, this approach requires human efforts to transform process specifications into biprocesses, which is not always straightforward. The applied pi-calculus approach is also used by Dong et al. [25]. While the anonymity checking is done automatically in this study, the receipt freeness property is still checked manually.

Halpern [11] and Jonker [12] proposed the formalizations of anonymity based on epistemic logic. The authors gave a logical characterization of the notion of the receipt in the electronic voting processes. However, these formalisms mainly focus on reasoning about the property and are less suited for modeling the protocol as well as attacker abilities. In addition, their logic aims to express properties rather than operational steps of a protocol. Thus, modeling protocols using epistemic-logic requires a high degree of expertise and is error prone. Pang and Zhang [26] also modeled security protocols using epistemic logic and used MCMAS model checker to automatically verify. However, only the anonymity property is investigated in their study.

The main results of our study are embodied in the design and implementation of a SeVe module, a self-contained framework for the automatic analysis of security protocols. Our contributions are threefold:

- First, we propose a framework for specifying and verifying security protocols using the label transition system model. Besides the behaviors of agents and intruders, the knowledge of participants, which prove effective to reason about security properties, are also included in the model. The abilities of intruders are parameterized for flexible choices in different environments.
- Second, we propose an approach for the integration of trace equivalence checking and knowledge reasoning to formally define and check privacy properties. By using this approach, we can automatically verify privacy properties, especially with receipt freeness and coercion resistance properties. Our tool is the first tool applying formal methods to automatically check these two properties.
- Third, we develop a SeVe module: an automatic tool to support specification, simulation, and verification of security protocols. SeVe is designed to support automatic verification for not only some common security properties such as secrecy and authentication but also for other properties such as anonymity and privacy.

3 System semantics

Security protocols describe the message terms sent between trusted participants during a session. A session is a single run of the protocol. Most protocols allow multiple concurrent sessions. Participants of the session are called agents. The environment in which the sender and receiver communicate is an unsecured environment. This unreliable environment is modeled by adding an intruder into the network, who is given special powers to tamper with the messages that pass around. Our approach follows the Dolev-Yao model [27] as in Fig. 1. The system is the interleaving of agents and intruder

activities:

$$System = (|||_{X\in\{Agent\}^*} Agent_X) \;|||\; Intruder,$$

where ||| denotes for interleaving.

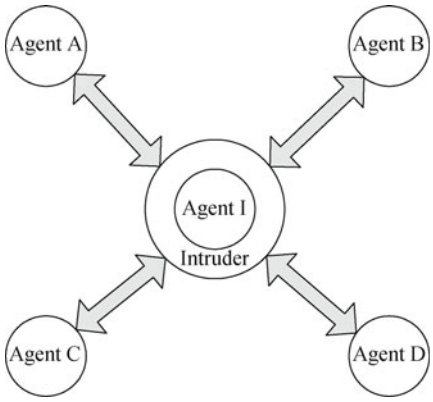

Fig. 1 Dolev-Yao model

• ***Basic sets*** We start with the basic sets: C is the set of constants, such as the nonce and session key; $\mathcal{F}$ is the set of function names, such as hash function and bit scheme schema; $\mathcal{A}$ is the set of participants, including $\mathcal{A}_\mathcal{T}$ denoting trusted agents and $\mathcal{A}_\mathcal{U}$ denoting untrusted agents (intruders). In Table 1 we show some typical elements of these sets, as used throughout this paper.

Table 1 Basic sets and some typical elements

Description	Set	Typical elements
Constants	C	na, nb, session key
Trusted agents	$\mathcal{A}_\mathcal{T}$	Alice, Bob, Carol
Untrusted agents	$\mathcal{A}_\mathcal{U}$	Jeeves
Functions	$\mathcal{F}$	kas, kbs, hash

• ***Term*** We define the set of $Term$ as the basic term sets, extended with constructors for pairing and encryption, and we assume that pairing is right-associative.

$$Term ::= \mathcal{A} \mid C \mid \mathcal{F}(Term) \mid (Term, Term) \mid \{Term\}_{Term}$$

Terms that have been encrypted with a term, can only be decrypted by either the same term (for symmetric encryption) or the inverse key (for asymmetric encryption). To determine which term needs to be known to decrypt a term, we introduce a function that yields the inverse for any term: $\_^{-1} : Term \to Term$. For example, if pk and sk are public key and private key of an agent correspondingly, we have: $pk^{-1} = sk$. Similarly, if s is a session key, we have: $s^{-1} = s$.

We require that $\_^{-1}$ is its own inverse, i.e., $(t^{-1})^{-1} = t$. Terms are reduced according to $\{\{s\}_t\}_{t^{-1}} = s$.

Definition 1 Message Amessage used in the security model has the form: $Message ::= sender \times receiver \times term$, where $sender \in \mathcal{A}$, $receiver \in \mathcal{A}$, and $term \in Term$.

We now turn to describe the protocol behaviors. The protocol is described as set of events, i.e., the send and read events, between agents. We also have other behaviors of intruders such as deflect, inject, eavesdrop, and jam. Their semantics are given in Section 1.

Definition 2 Security events The set of events used to describe the security protocol behaviors are defined as:

$$\begin{aligned} RunEvent = \{&send(m),\ read(m),\ asend(m),\ aread(m),\\ &usend(m),\ uread(m),\ deflect(m),\ inject(m),\\ &eavesdrop(m),\ jam(m),\ start(s,r),\ commit(s,r)\\ &\mid m \in Message,\ s, r \in \mathcal{A}\}. \end{aligned}$$

Definition 3 Security process A security process is defined as the following BNF, where P and Q range over processes, e is a security event.

$$\begin{aligned} P = {} & Stop \mid Skip && \text{– primitives}\\ \mid {} & e \to P && \text{– event prefixing}\\ \mid {} & P[\,]Q && \text{– general choice}\\ \mid {} & P \;|||\; Q && \text{– interleave composition}\\ \mid {} & P \parallel Q && \text{– parallel composition}\\ \mid {} & P; Q && \text{– sequential composition} \end{aligned}$$

Each agent or intruder has his own knowledge which is defined at the beginning of the protocol. During a protocol run, the participants can learn and enhance their knowledge. We denote V is the knowledge of participants upon the running of the system. $V : \mathcal{A} \to \mathcal{S}_T$ is the function mapped from set of agents A to set of terms $\mathcal{S}_T$. The messages are sent and read via an unsecured environment. We use the set B of messages as the buffer representing this environment.

Definition 4 System configuration A system configuration is a 3-element state $s = \langle V, P, B\rangle$, where V is the knowledge of agents, P is the running process, and B is the buffer of the messages. At the initial state of the system, the buffer is empty and the program starts at the System root, thus the initial state of the system is given by $init_{im} = \langle V_0, System, \phi\rangle$, where V_0 refers to the initial knowledge.

3.1 Operational semantics

In this part, we will introduce the semantics of agent and intruder rules respectively.

• **Agent rules**

The agent can compose and decompose pair terms in a message. A term can be encrypted if the agent knows the encryption key, and an encrypted term can be decrypted if the agent knows the corresponding decryption key. This is expressed by the knowledge inference operator, which is defined inductively as follows (M is the set of terms):

$$\begin{array}{lll} t \in M & \Rightarrow & M \vdash t \\ M \vdash t1 \ \bigwedge\ t1 = t2 & \Rightarrow & M \vdash t2 \\ M \vdash t1 \ \bigwedge\ t1 \vdash t2 & \Rightarrow & M \vdash (t1, t2) \\ M \vdash t \ \bigwedge\ M \vdash k & \Rightarrow & M \vdash \{t\}_k \\ M \vdash \{t\}_k \ \bigwedge\ M \vdash k^{-1} & \Rightarrow & M \vdash t \\ M \vdash \mathcal{F}(t) \ \bigwedge\ M \vdash \mathcal{F}^{-1} & \Rightarrow & M \vdash t \end{array}$$

The process of learning information from a message is described as a function $Learn : V \times \mathcal{A} \times Message \rightarrow \mathcal{S}_T$, where $\mathcal{S}_T$ is set of terms and function $\_^{-1} : Term \rightarrow Term$ yields the reverse for a term as described before.

procedure $Learn(V, receiver, m)$
1 $if\ (m \in C \bigcup \mathcal{A})$;
2 $return\ \{m\}$;
3 $if\ (m\ is\ (t1, t2))$
4 $return\ Learn(t1) \bigcup Learn(t2)$;
5 $if\ (m\ is\ (t1)_{t2})\ \&\&\ V(receiver) \vdash t2^{-1}$
6 $return\ Learn(t1)$;
7 $if\ (m\ is\ \mathcal{F}(t1))\ \&\&\ V(receiver) \vdash \mathcal{F}^{-1}$
8 $return\ Learn(t1)$;

Whenever a message is received, the receiver will decrypt the message (if he can), get the information, and update his knowledge. We implement this behavior as a procedure $Update : V \times Message \times \mathcal{A}$.

procedure $Update(V, message, agent)$
1 $S = V(agent)$;
2 $T = Learn(V, agent, message)$;
3 $foreach\ (i\ in\ T)$
4 $if\ (S \nvdash i)$
5 $S = S \bigcup \{i\}$

The public send rule states that if a run executes a send event, the sent message is added to the buffer and the executing run proceeds to the next event. The public read rule requires that the message pattern specified in the read event should match any of the messages from the buffer. Upon execution of the read event, this message is removed from the buffer, the knowledge of the trusted agents is updated and the executing run advances to the next event.

$$\frac{p = send(m), V(m.sender) \vdash m.term}{(V, p \rightarrow Q, B) \xrightarrow{p} (V, Q, B \bigcup \{m\})}\ [public\ send]$$

$$\frac{p = read(m), m \in B}{\begin{array}{l}(V, p \rightarrow Q, B) \xrightarrow{p} (V', Q, B \setminus \{m\}), \\ V' = Update(V, m, m.receiver)\end{array}}\ [public\ read]$$

The anonymous send and read rules are semantically similar with public send and read rules, except that the information about sender is omitted.

$$\frac{p = asend(m), V(m.sender) \vdash m.term}{\begin{array}{l}(V, p \rightarrow Q, B) \xrightarrow{p} (V, Q, B'), \\ B' = B \bigcup \{(\_, m.receiver, m.term)\}\end{array}}\ [ano\ send]$$

$$\frac{p = aread(m), m \in B, m.receiver = \_}{\begin{array}{l}(V, p \rightarrow Q, B) \xrightarrow{p} (V', Q, B \setminus \{m\}), \\ V' = Update(V, m, m.receiver)\end{array}}\ [ano\ read]$$

The untappable send and read rules state the events which happen outside intruder control or even intruder awareness. It thus limits the power of an intruder. In this case, the intruder does not learn the communicated term for untappable send and read events. We do this in model generation stage: the intruder behaviors are not generated corresponding to untappable messages.

$$\frac{p = usend(m), V(m.sender) \vdash m.term}{(V, p \rightarrow Q, B) \xrightarrow{p} (V, Q, B \bigcup \{m\})}\ [untap\ send]$$

$$\frac{p = uread(m), m \in B}{\begin{array}{l}(V, p \rightarrow Q, B) \xrightarrow{p} (V', Q, B \setminus \{m\}), \\ V' = Update(V, m, m.receiver)\end{array}}\ [untap\ read]$$

The $Start$ and $Commit$ events are used to mark the start and finish signal of the session. They do not change the knowledge of the participants V and buffer B.

$$\frac{a1 \in \mathcal{A}, a2 \in \mathcal{A},\ p = start(a1, a2)}{(V,\ p \rightarrow P,\ B) \xrightarrow{p} (V,\ P,\ B)}\ [start]$$

$$\frac{a1 \in \mathcal{A}, a2 \in \mathcal{A},\ p = commit(a1, a2)}{(V,\ p \rightarrow P,\ B) \xrightarrow{p} (V,\ P,\ B)}\ [commit]$$

• Intruder rules

To determine which message that the intruder can observe in the buffer B, we define a function $In : Message \times B \rightarrow Boolean$ as

procedure $In(m, B)$
1 $foreach\ (m1 \in B)$;

```
2   if (m == m1)
3       return true;
4   else if(m1.sender == _ && m1.term == m.term)
5       return true;
6   else
7       return false;
```

Line 4 and line 5 in the above procedure represent the case that the message is sent and read via an anonymous channel.

If an intruder has eavesdropping capabilities, as stated in the eavesdrop rule, he can learn the message during transmission. The jam rule states that an intruder with action capabilities can delete any message from the output buffer. The difference of the deflect rule is that the intruder can read the message and add it to its knowledge. The inject rule describes the injection of any message inferable from the intruder knowledge into the input buffer.

$$\frac{In(m,B), p = deflect(m)}{(V, p \to P, B) \xrightarrow{p} (V', P, B \setminus m),\ V' = Update(V, m, intruder)}\ [deflect]$$

$$\frac{V(intruder) \vdash m.term, p = inject(m)}{(V, p \to P, B) \xrightarrow{p} (V, P, B \bigcup m)}\ [inject]$$

$$\frac{In(m,B), p = eavesdrop(m)}{(V, p \to P, B) \xrightarrow{p} (V', P, B'),\ V' = Update(V, m, intruder)}\ [eavesdrop]$$

$$\frac{In(m,B), p = jam(m)}{(V, p \to P, B) \xrightarrow{p} (V, P, B \setminus m)}\ [jam]$$

The operational semantics for other rules such as interleaving or choice are described in Appendix B.

3.2 Model semantics

In this part, we will investigate the formalization of privacy-type properties. The semantics of a model are defined by a labeled transition system (LTS). Let Σ_τ denote the set of all events. Let Σ^* be the set of finite traces.

Definition 5 LTS A LTS is a 3-tuple $L = (S, init, T)$ where S is a set of states, $init \in S$ is the initial state, and $T \subseteq S \times \Sigma_\tau \times S$ is a labeled transition relation.

For states $s, s' \in S$ and $e \in \Sigma_\tau$, we write $s \xrightarrow{e} s'$ to denote $(s, e, s') \in T$. The set of enabled events at s is $enabled(s) = \{e : \Sigma_\tau | \exists s' \in S, s \xrightarrow{e} s'\}$. We write $s \xrightarrow{e_1, e_2, \ldots, e_n} s'$ iff there exist $s_1, s_2, \ldots, s_{n+1} \in S$ such that $s_i \xrightarrow{e_i} s_{i+1}$ for all $1 \leqslant i \leqslant n$, $s_1 = s$ and $s_{n+1} = s'$. Let $tr : \Sigma^*$ be a sequence of events. $s \overset{tr}{\Rightarrow} s'$ if and only if there exist $e_1, e_2, \ldots, e_n \in \Sigma_\tau$ such that $s \xrightarrow{e_1, e_2, \ldots, e_n} s'$. The set of traces of L is $traces(L) = \{tr : \Sigma^* | \exists s' \in S, init \xrightarrow{tr} s'\}$.

Given a model composed of a process P and a set of knowledge V and the buffer B, we may construct an LTS $(S, init, T)$ where $S = \{s | (V, P, B) \to^* s\}$, $init = (V, P, B)$ and $T = \{(s_1, e, s_2) : S \times \Sigma_\tau \times S | s_1 \xrightarrow{e} s_2\}$ using the operational semantics. The following definition defines refinement and equivalence relations.

Definition 6 Refinement and Equivalence Let $L_{im} = (S_{im}, init_{im}, T_{im})$ be an LTS for an implementation. Let $L_{sp} = (S_{sp}, init_{sp}, T_{sp})$ be an LTS for a specification. L_{im} refines L_{sp}, written as $L_{im} \sqsupseteq_T L_{sp}$, iff $traces(L_{im}) \subseteq traces(L_{sp})$. L_{im} equals L_{sp} in the trace semantics, written as $L_{im} \equiv L_{sp}$ iff they refine each other.

In the LTS model of a security protocol, we also need to apply knowledge reasoning in verification: during a protocol run, an agent does not have knowledge about specific information. We capture this semantically using the following definition of *Unknown knowledge*.

Definition 7 Unknown knowledge Let $L_{im} = (S_{im}, init_{im}, T_{im})$ be an LTS for an implementation. Given an agent a and term t. t is unknown knowledge of agent a in the implementation, written as $UnknownKnowledge(L_{im}, a, t) == true$ iff $\forall tr = \langle e_1, e_2, \ldots, e_n \rangle \in traces(L_{im}), \forall i \in \{1, 2, \ldots, n\}\ \mathcal{M}^{tr}_{e_i}(a) \nvdash t1$, where $M^{tr}_{e_i}(a)$ is the knowledge of the agent a of the system following trace tr before executing event e_i.

Given an event e, process P, term x and $x1$, we denote $e[x1/x]$ as a new event which replaces all occurrences of x in e with $x1$, and $P[x1/x]$ is the new process that replaces all occurrences of x in P by $x1$. The function $In(x, e)$ is defined as: $In(x, e) = true$ if x occurs in event e, otherwise $In(x, e) = false$.

Definition 8 Event renaming function Let A be a set of terms, an event renaming function $f_A : \Sigma \to \Sigma$ is the function that satisfies:

- $f_A(e) = e[\alpha/x]$ if $\exists x \in A, In(x, e) == true$
- $f_A(e) = e$ if $\forall x \in A, In(x, e) == false$

where α is an anonymous term and $\alpha \notin A$.

Process $f_A(P)$ performs event $f_A(e)$ whenever P performs e. We also have the notion of the reverse renaming function

f_A^{-1}, where [] denotes choice.

- $f_A^{-1}(e) = [\,]_{i=1..n} e[x_i/\alpha]$, where $A = \{x_1, x_2, ..., x_n\}$, if $In(x, \alpha) == true$
- $f_A^{-1}(e) = e$, if $In(x, \alpha) == false$

Process $f_A^{-1}(P)$ performs any event from set $f_A^{-1}(e)$ whenever P performs e.

As an illustration of the definition, we consider a simple voting protocol: there are two choices of candidates corresponding two values of a vote v named: $v1$ and $v2$. The voter V sends collector C his vote which is encrypted by the public key of collector PkC. If the voter votes $v1$, the collector will send the voter receipt $r1$, otherwise the collector sends receipt $r2$. The process representing the voter will be:

$$\begin{aligned} Vote() &= Vote[v1/v] \,[\,]\, Vote[v2/v] \\ &= Send(V, C, \{v1\}_{PkC}) \rightarrow Read(C, V, r1) \rightarrow Skip; \\ &\quad [\,]\, Send(V, C, \{v2\}_{PkC}) \rightarrow Read(C, V, r2) \rightarrow Skip; \end{aligned}$$

Let $A = \{v1, v2\}$, we have:

$$\begin{aligned} &f_A(Vote()) \\ &\quad = Send(V, C, \{\alpha\}_{PkC}) \rightarrow Read(C, V, r1) \rightarrow Skip; \\ &\quad [\,]\, Send(V, C, \{\alpha\}_{PkC}) \rightarrow Read(C, V, r2) \rightarrow Skip; \end{aligned}$$

$$\begin{aligned} &f_A^{-1}(f_A(Vote())) \\ &\quad = Send(V, C, \{v1\}_{PkC}) \rightarrow Read(C, V, r1) \rightarrow Skip; \\ &\quad [\,]\, Send(V, C, \{v1\}_{PkC}) \rightarrow Read(C, V, r2) \rightarrow Skip; \\ &\quad [\,]\, Send(V, C, \{v2\}_{PkC}) \rightarrow Read(C, V, r1) \rightarrow Skip; \\ &\quad [\,]\, Send(V, C, \{v2\}_{PkC}) \rightarrow Read(C, V, r2) \rightarrow Skip; \end{aligned}$$

3.3 Formalizing security properties

This part is devoted to introducing the formalization of security properties in the LTS model. The properties examined in this study are: secrecy, authentication, anonymity, receipt freeness, and coercion resistance.

• Secrecy property

The secrecy property concerns a message used by the protocol. The requirement is that an adversary should not be able to derive the whole message or a part of it we want to keep secret. That information may be leaked during the learning of an intruder: he gets the message, understands the contents of an messages directly or using his private key to decrypt the message encrypted with his public key.

For $t \in Term$, $r \in \mathcal{A}$, we define $secret(t, r)$ as the goal that information t is kept secret by agent r during security protocols run.

Theorem 1 **Secrecy** Let $L_{im} = (S_{im}, init_{im}, T_{im})$ be an LTS for the implementation: $secret(t, r) = true \Leftrightarrow UnknownKnowledge(L_{im}, r, t) == true$.

Proof The proof is quite trivial, the information t is kept secret by agent r if and only if at every state of the traces, r cannot have knowledge about t.

• Authentication property

For $r \in \mathcal{AT}$, $s \in \mathcal{AT}$, we define $authentication(r, s)$ as the goal that after execution of the protocols between r and s, r is assured that he actually finished the communication with s. Consider the following process: $P1 = start(s, r) \rightarrow commit(r, s)$ and LTS $L_{au} = (S, init, T)$ where $S = \{s | (\phi, P1, \phi) \rightarrow^* s\}$, $init = (\phi, P1, \phi)$ and $T = \{(s_1, e, s_2) : S \times \Sigma_\tau \times S | s_1 \xrightarrow{e} s_2\}$.

Theorem 2 **Authentication** Let $L_{sp} = (S_{sp}, init_{sp}, T_{sp})$ be an LTS for the specification, $authentication(r, s) = true \Leftrightarrow L_{au} \sqsupseteq_T L_{sp}$.

Proof. A message-oriented approach to authentication is discussed in [28]. Authentication will be captured in terms of events (generally transmission and receipt of particular messages) whose occurrence guarantees the prior occurrence of other messages. r is assured that he just finished the communication with s if and only if whenever we get the *commit* signal from r, we must previously have received the *start* signal from s. This authentication is illustrated by Fig. 2.

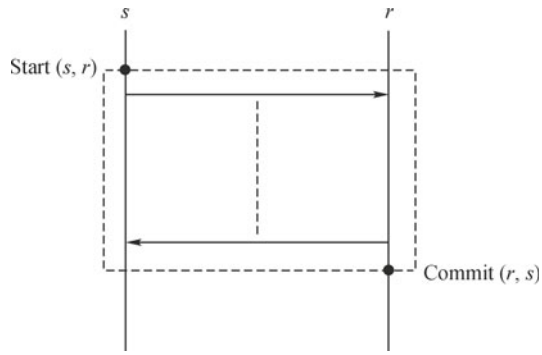

Fig. 2 Authentication checking

• Privacy-type properties

Our formal definitions of anonymity, receipt freeness, and coercion resistance can be compared with the work of Delaune et al. [9]. While the authors in [9] give the definitions of these privacy properties in applied Pi-calculus using bisimulation checking, we define them relying on observational equivalence and knowledge inference. In both our work and theirs, the ideas in anonymity checking are similar: if the intruder cannot detect any differences when we swap the possible val-

ues of a term in protocol runs, the anonymity of this term is satisfied. However, the receipt freeness and coercion resistance checking in our work and [9] are different. In [9], the authors modeled the fact that the agent and intruder exchange information by giving additional extended processes and contexts. However, their definitions are not amenable to automation because they require universal quantification over an infinite set of contexts. In our definitions, we come up with a more natural way: the agents and intruder have their own knowledge, and they can exchange their knowledge at the initial stage as well as during protocol runs. The privacy checking turns out to be observational equivalence and knowledge inference, which can be automatically checked using trace equivalence and knowledge reasoning algorithms. In this section, we will give the formal definitions for three kinds of privacy. The checking algorithms are given in Section 4.

• **Anonymity (Untraceability)**

Let t be a term that the protocol will ensure the anonymity on it. Assuming $t1$ and $t2$ are two terms representing different values of t. Denote $V0$ as the initial knowledge of the protocol. Consider two following contexts:

- The first context represents the choice situation in term t: $System_{im} = System[t1/t]\ []\ System[t2/t]$. Let LTS $L_{im} = (S1, init_{im}, T1)$ where $init_{im} = (V0, System_{im}, \phi)$, $S1 = \{s | init_{im} \rightarrow^* s\}$, and $T1 = \{(s_1, e, s_2) : S1 \times \Sigma_\tau \times S1 | s_1 \xrightarrow{e} s_2\}$.
- The second context applies renaming function: $System_{sp} = f^{-1}_{\{t1,t2\}}(f_{\{t1,t2\}}(System_{im}))$ and LTS $L_{au} = (S2, init_{au}, T2)$ where $init_{au} = (V0, System_{au}, \phi)$, $S2 = \{s | init_{au} \rightarrow^* s\}$, and $T2 = \{(s_1, e, s_2) : S2 \times \Sigma_\tau \times S2 | s_1 \xrightarrow{e} s_2\}$.

Definition 9 Anonymity (Untraceability) The protocol assures the anonymity (or untraceability) of term t if and only if $L_{im} \equiv L_{au}$.

This definition states that if every occurrence of $t1$ or $t2$ were renamed to a new dummy value α (which is the situation in the process $f_{\{t1,t2\}}(System_{im})$), then whenever an event containing α is possible in this renamed process, any possible corresponding event containing $t1$ or $t2$ should have been possible in the original process (this is assured by using reverse renaming function $f^{-1}_{\{t1,t2\}}(f_{\{t1,t2\}}(System_{im}))$. The equation in traces means that at any time t is replaced by $t1$ or $t2$, the intruder cannot observe any difference in traces and therefore cannot infer anything about t.

Consider the illustration example of the event renaming function given before. The traces of process $f_A^{-1}(f_A(Vote()))$ are different the traces of $Vote()$. One of the traces it has is $< Send(V, C, \{v1\}_{PkC}), Read(C, V, r2) >$ which is not possible for $Vote()$. This indicates that the occurrence of the event $Read(C, V, r2)$ allows a distinction to be made between different events containing $v1, v2$ and so this vote process does not provide anonymity.

• **Receipt freeness**

Similar to anonymity, receipt freeness can be formalized using the event renaming function. In addition, we need to model the fact that an agent is willing to share reliable secret information with an intruder. We do it by changing the initial knowledge of the intruder: the intruder's knowledge will be added to the initial knowledge of agents, except for some privilege knowledge (such as private key) or unreliable knowledge (such as the key of a trap-door commitment scheme which the user can fake but the intruder cannot detect). We call that initial knowledge $V1$. Let t be a term that the protocol ensures the receipt freeness on it. Assuming $t1$ and $t2$ are two terms representing different values of t. Consider three following contexts:

- The first context represents the choice situation in term t: $System_{im} = System[t1/t]\ [\ ]\ System[t2/t]$. Let LTS $L_{im} = (S1, init_{im}, T1)$ where $init_{im} = (V1, System_{im}, \phi)$, $S1 = \{s | init_{im} \rightarrow^* s\}$, and $T1 = \{(s_1, e, s_2) : S1 \times \Sigma_\tau \times S1 | s_1 \xrightarrow{e} s_2\}$.
- The second context applies renaming function: $System_{sp} = f^{-1}_{\{t1,t2\}}(f_{\{t1,t2\}}(System_{im}))$ and LTS $L_{au} = (S2, init_{au}, T2)$ where $init_{au} = (V1, System_{au}, \phi)$, $S2 = \{s | init_{au} \rightarrow^* s\}$, and $T2 = \{(s_1, e, s_2) : S2 \times \Sigma_\tau \times S2 | s_1 \xrightarrow{e} s_2\}$.
- The third context represents the situation in which the agent sends $t1$: $System_{t1} = System[t1/t]$ and LTS $L_{t1} = (S3, init_{t1}, T3)$ where $init_{t1} = (V1, System_{t1}, \phi)$, $S3 = \{s | init_{t1} \rightarrow^* s\}$, and $T3 = \{(s_1, e, s_2) : S3 \times \Sigma_\tau \times S3 | s_1 \xrightarrow{e} s_2\}$.

Definition 10 Receipt freeness The protocol assures receipt freeness of term t if and only if the following two conditions hold:

1. $L_{im} \equiv L_{au}$.
2. $UnknownKnowledge(L_{t1}, intruder, t1) == true$.

The first condition is similar to the anonymity condition: the intruder cannot tell the difference in traces when t is replaced by $t1$ and $t2$. The second condition gives the situation when the agent wants to fool the intruder: while the agent tells the intruder that he sent $t2$, he actually sent $t1$. The receipt freeness property holds if the intruder, with all reliable

knowledge the agent supplies, cannot detect that the agent is sending $t1$.

• **Coercion resistance**

Coercion resistance is a strongest property as we give intruders the ability to communicate interactively with the agent. In this model, an intruder can prepare the messages that they want the agent to send.

Let t be a term that the protocol ensures the coercion resistance on. Assuming $t1$ and $t2$ are two terms representing different values of t, we specify the initial knowledge of the agent and intruder in this case as: intruder knowledge is all the terms needed to generate all messages in the sessions, whereas agent knowledge is only messages corresponding to the session which t is replaced by $t1$ and $t2$ as supplied by intruder. We call the initial system knowledge $V2$. Consider another case: the intruder knows and wants to send $t2$, so they supply all the necessary messages for an agent to send in this way; however, they do not know about $t1$. However, the agent, who based on information supplied by intruder, knows $t1$ but may not know how to construct the protocol messages to send $t1$. We call that initial system knowledge $V3$. Consider the three following contexts:

- The first context represents the choice situation in term t: $System_{im} = System[t1/t]\ [\ ]\ System[t2/t]$ and LTS $L_{im} = (S1, init_{im}, T1)$ where $init_{im} = (V2, System_{im}, \phi)$, $S1 = \{s | init_{im} \rightarrow^* s\}$, and $T1 = \{(s_1, e, s_2) : S1 \times \Sigma_\tau \times S1 | s_1 \xrightarrow{e} s_2\}$.
- The second context applies renaming function: $System_{sp} = f^{-1}_{\{t1,t2\}}(f_{\{t1,t2\}}\ (System_{im}))$ and LTS $L_{au} = (S2, init_{au}, T2)$ where $init_{au} = (V2, System_{au}, \phi)$, $S2 = \{s | init_{au} \rightarrow^* s\}$, and $T2 = \{(s_1, e, s_2) : S2 \times \Sigma_\tau \times S2 | s_1 \xrightarrow{e} s_2\}$.
- The third context represents the situation in which the agent sends $t1$: $System_{t1} = System[t1/t]$ and LTS $L_{t1} = (S3, init_{t1}, T3)$ where $init_{t1} = (V3, System_{t1}, \phi)$, $S3 = \{s | init_{t1} \rightarrow^* s\}$, and $T3 = \{(s_1, e, s_2) : S3 \times \Sigma_\tau \times S3 | s_1 \xrightarrow{e} s_2\}$.

Definition 11 Coercion resistance The protocol assures coercion resistance on term t if and only if two following conditions are hold:

1. $L_{im} \equiv L_{au}$.
2. $UnknownKnowledge(L_{t1}, intruder, t1) == true$.

The first condition is similar to the anonymity condition: the intruder cannot tell the difference in traces when t is replaced by $t1$ and $t2$. The second condition enables us to reason about the coercer's choice of t: while intruder forces the agent to send $t2$ by supplying all the necessary messages, the agent successfully fools the intruder by sending $t1$.

These two conditions look similar with the condition of receipt freeness. However, there are differences in the initial knowledge of participants. In the case of coercion resistance, the agent's knowledge is constituted from all the messages supplied by the intruder; moreover, the intruder knows how to generate them whereas the agent may not. In the case of receipt freeness, the agent knows how to generate those messages and the intruder's knowledge is only supplied reliable information carried out by agents.

4 Algorithm and implementation

In this part, we will present the algorithms used in the verification of security properties. The architecture and implementation of the SeVe module are also briefly introduced.

4.1 Verification algorithms

In privacy checking, we introduce an algorithm for checking the trace equivalence. Let $Spec = (S_{sp}, init_{sp}, T_{sp})$ be a specification and $Impl = (S_{im}, init_{im}, T_{im})$ be an implementation, the checking of trace equivalence is defined in Fig. 3, where $refine$ is denoted as the refinement checking function. In this paper, we follow the on-the-fly refinement checking algorithm from [29], which is based on the refinement checking algorithms in FDR [30] but applies partial order reduction. This algorithm performs a reachability analysis on the product of the implementation and normalized specification, while the normalization is also applied on-the-fly. Because normalization is applied on-the-fly, it is sometimes possible to find a counterexample before the specification is completely normalized.

To check the *Unknown Knowledge* relation, we apply a depth first search (DFS) algorithm. The algorithm for *Unknown Knowledge* checking is presented in Fig. 3. In line 6, $s2.V(agent)$ is the knowledge of *agent* at state $s2$. Note that the operator $\vdash$ is recursively calculated as in Section 1. In this algorithm, we record all the visited states to detect the loop in traces (line 5). The worst complexity of this algorithm is $|n_s|$, the number of states in the state space.

4.2 Implementation: SeVe module

Model checker PAT[1)] is designed to apply state-of-the-art

[1)] http://www.patroot.com

```
procedure Equivalence(Impl, Spec)
1. if (refine(Impl, Spec) == true
2.     && refine(Spec, Impl) == true)
3.       return true;
4. else
5.       return false;

procedure Unknown_Knowledge(Impl, agent, term)
1. visited.push(init_im);
2. while visited ≠ ∅ do
3.       s1 := visited.pop();
4.       foreach (s2 ∈ enabled(s1))
5.           if (s2 ∉ visited)
6.               if (s2.V(agent) ⊢ term)
7.                   return false;
8.               else
9.                   visited.push(s2);
10.              end if
11.          end if
12.      end for
13. end while
14. return true;
```

Fig. 3 Algorithm: Equivalent and Unknown_Knowledge functions

model checking techniques for system analysis. PAT [31,32] supports a wide range of modeling languages including communicating sequential programs (CSP), which shares similar design principle with integrated specification languages like TCOZ [33,34] or SOFL [35,36]. The verification features of PAT are abundant in that an on-the-fly refinement checking algorithm is used to implement linear temporal logic (LTL) based verification, partial order reduction is used to improve verification efficiency, and LTL based verification supports both event and state checking. Furthermore, PAT has been enhanced for verifying properties such as deadlock freeness, divergence freeness, timed refinement, and temporal behaviors.

With all of these advantages, we have implemented the ideas of privacy checking into the SeVe module of the PAT model checker. Figure 4 shows the architecture design of the SeVe module with five components. The security protocols are specified using SeVe language. The *Intruder behavior generator* will automatically generate all possible attacks of the intruder using the Dolev-Yao model. The *SeVe compiler* will compile all those behaviors of protocols into SeVe processes, of which operational semantics are defined in Section 1. These processes are then transmitted to the *LTS Generator*, which generates the LTS semantic model. This model can be passed to the *Simulator* or *Model checker* for simulating behaviors or verifying security properties, respectively. The counter example, if it exists, will be showed visually by the *Simulator*.

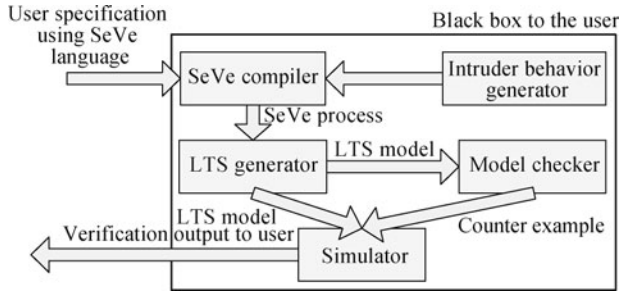

Fig. 4 SeVe architecture

The SeVe language can be considered an extension of Casper languages [6], however, we have some amelioration. Firstly, we support many kinds of property checking for different security purposes (anonymity, receipt freeness, coercion resistance, etc.). The user also does not need to specify the behavior of the intruder, which may be very complicated. By observing the general intruder behavior as in the Dolev-Yao model, the generator will automatically generate all possible attacks of the intruder. The intruder ability is also parameterized, so the user can describe the capability of an intruder corresponding to different situations. The full grammar of the SeVe language is in Appendix A. To illustrate the language, the SeVe specification of Needham Schroeder public key protocol and Fujioka et al. protocol, which we examine in Section 5, are given in Appendix C.

5 Case study

In this section, we investigate two case studies on security verification using our SeVe approach: the Needham Schroeder public key protocol with authentication property, and Fujioka et al.'s. electronic voting protocol with privacy property.

5.1 Needham Schroeder public key protocol

• Description and the model in LTS

In this part, we will examine the Needham Schroeder public key protocol [37]. This protocol involves communication between two agents A and B:

$$A \to B : \{A, N_A\}_{pk_B}$$
$$B \to A : \{N_A, N_B\}_{pk_A}$$
$$A \to B : \{N_B\}_{pk_B}.$$

The goal of this protocol is to provide authentication between A and B. When the session ends, agent A is sure that

she was talking to B and agent B is sure that he was talking to A. To ensure this property, when A decodes the second message, she verifies that the person she is talking to correctly inserted N_A in it. As N_A was encrypted by the public key of B in the first message, only B could determine the value of N_A. When B decodes the third message, he verifies that the nonce is N_B. As N_B encrypted the message with the public key of A, A is the only agent that could have deduced N_B. For these two reasons, A thinks that she is talking to B and B thinks that she is talking to A. The SeVe description for this protocol is given in Appendix C. In this section, we examine the internal processes of the model.

Each of the three messages in the Needham Schroeder protocol is sent by one agent and received by another. The view that each process has the running protocol is the sequences of sent and received messages as follows:

A's view:
Message 1: A sends to B: $\{A, N_A\}_{pk_B}$
Message 2: A receives from B: $\{N_A, N_B\}_{pk_A}$
Message 3: A sends to B: $\{N_B\}_{pk_B}$
B's view:
Message 1: B receives from A: $\{A, N_A\}_{pk_B}$
Message 2: B sends to A: $\{N_A, N_B\}_{pk_A}$
Message 3: B receives from A: $\{N_B\}_{pk_B}$

SeVe tool can automatically translate a protocol description in SeVe language to an LTS model, which is suitable for the model checker. For example, below are the processes representing the communication between A and B:

$$\begin{aligned} AgentA(B) = {} & Start(A, B) \\ & \rightarrow send(A, B, \{A, N_A\}_{pk_B}) \\ & \rightarrow read(B, A, \{N_A, N_B\}_{pk_A}) \\ & \rightarrow send(A, B, \{N_B\}_{pk_B}) \\ & \rightarrow commit(A, B) \rightarrow Skip(); \end{aligned}$$

$$\begin{aligned} AgentB(A) = {} & Start(B, A) \\ & \rightarrow read(A, B, \{A, N_A\}_{pk_B}) \\ & \rightarrow send(B, A, \{N_A, N_B\}_{pk_A}) \\ & \rightarrow read(A, B, \{N_B\}_{pk_B}) \\ & \rightarrow commit(B, A) \rightarrow Skip(); \end{aligned}$$

In the Dolev-Yao model, the intruder I is also an agent; therefore, we have other processes representing the communication between A, B, and I. However, for brevity, they are not given here.

To automatically generate the intruder behaviors, we notice that: although the intruder can fake or learn anything he can, it is only useful if the intruder manipulates information in the communication between trusted agents. He only needs to fake messages which can be used to deceive the agents, i.e., messages from trusted agents communication. Hence, we generate the intruder behaviors based on messages sent between trusted agents. For example, for the first message in the Needham Schroeder protocol, we can generates some behaviors for the intruder with *deflect* and *inject* ability as

$$\begin{aligned} Intruder() = {} & deflect(A, B, \{A, N_A\}_{pk_B}) \rightarrow Intruder \\ & [\,]\ inject(A, B, \{A, N_A\}_{pk_B}) \rightarrow Intruder \\ & \ldots \end{aligned}$$

where [ ] denotes *choice* (see Appendix B). Each of above behaviors presents one ability of an intruder, which is a sequential execution of rules.

The system is the interleaving of behaviors of agent A, agent B, and intruder I:

$System(vt) = AgentA(B) \,|||\, AgentB(A) \,|||\, AgentA(I) \,|||\, ... \,|||\, Intruder()$;

• Analysis

Before analyzing the protocols, we need to specify the initial knowledge of the participants. The initial knowledge of each participant in this protocol includes: the identifier and public key of all participants, his own private key, and the nonce.

- A's knowledge:
  $V(A) = \{A, B, I, N_A, pk_A, pk_A^{-1}, pk_B, pk_I\}$.
- B's knowledge:
  $V(B) = \{A, B, I, N_B, pk_B, pk_B^{-1}, pk_A, pk_I\}$.
- Intruder's knowledge:
  $V(I) = \{A, B, I, N_I, pk_I, pk_I^{-1}, pk_A, pk_B\}$.

Authentication of A by B. To verify the authentication of A by B, we check the fact that: whenever the event $commit(B, A)$ happens, the event $start(A, B)$ has occurred beforehand. The SeVe tool returns a counter example for this assertion:

$$\begin{aligned} & < send(A, I, \{A, N_A\}_{pk_I}) \rightarrow deflect(A, I, \{A, N_A\}_{pk_I}) \rightarrow \\ & inject(A, B, \{A, N_A\}_{pk_B}) \rightarrow read(A, B, \{A, N_A\}_{pk_B}) \rightarrow \\ & send(B, A, \{N_A, N_B\}_{pk_B}) \rightarrow deflect(B, A, \{N_A, N_B\}_{pk_B}) \rightarrow \\ & inject(I, A, \{N_A, N_B\}_{pk_A}) \rightarrow read(I, A, \{N_A, N_B\}_{pk_A}) \rightarrow \\ & send(A, I, \{N_B\}pk_I) \rightarrow deflect(A, I, \{N_B\}pk_I) \rightarrow \\ & inject(A, B, \{N_B\}pk_B) \rightarrow read(A, B, \{N_B\}pk_B) \rightarrow \\ & commit(B, A) > \end{aligned}$$

This counter example is represented as in Fig. 5. It shows that the intruder can successfully delude B by pretending that he is A.

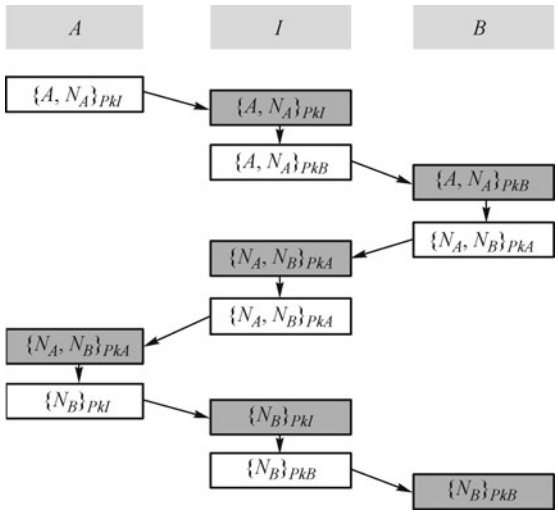

Fig. 5 Counter example of authentication checking

Authentication of B by A. Similarly, we can verify the authentication of B by A by checking the fact that: whenever the event $commit(A, B)$ happens, the event $start(B, A)$ has occurred beforehand. The SeVe tool returns a valid result, meaning that the protocol satisfies the authentication of B by A.

5.2 Electronic voting protocol of Fujioka et al.

• Description

In this part, we study a protocol due to Fujioka, Okamoto, and Ohta [38]. The protocol involves voters, an administrator, verifying that only eligible voters can cast votes, and a collector, collecting and publishing the votes. The protocol uses some unusual cryptographic primitives such as blind signatures. In a first phase, the voter gets a signature on a commitment to his vote from the administrator.

1. Voter V selects a vote vt and computes the commitment $x = \{vt\}_r$ using a random key r;
2. V computes the message $e = \Psi(x, b)$ using a blinding function Ψ and a random blinding factor b;
3. V digitally signs e and sends her signature $sV(e)$ to the administrator A;
4. A verifies that V has the right to vote and the signature is valid; if all these tests hold, A digitally signs e and sends his signature $sA(e)$ to V;
5. V now unblinds $sA(e)$ and obtains $y = sA(x)$, a signed commitment to $V's$ vote.

The second phase of the protocol is the actual voting phase.

1. V sends y, A's signature on the commitment to V's vote, to the collector C;
2. C checks the correctness of the signature y and, if the test succeeds, records x and y;
3. V sends r to C via an untappable channel;
4. C decrypts x using r to get the vote vt.

The SeVe description for Lee et al. protocol is given in Appendix C. In next part, we will examine the LTS model of this protocol.

• The model in LTS

To model blind signature, we consider the following equation:

$$\Psi^{-1}(\{\Psi(x, b)\}sA, b) = \{x\}sV$$

Voter. The voter interacts with administrator and counter during protocol runs by sending and receiving messages. The digital signature of administrator is checked in read event: the message the voter read must be matched with the message he is waiting. Moreover, because voter's knowledge contains the public key of administrator, he can only decrypt and get information from message signed by administrator's signature key.

$$\begin{aligned} Voter(vt) \;=\; & send(V, A, \{\Psi(\{vt\}r, b)\}sV) \\ & \rightarrow read(A, V, \{\Psi(\{vt\}r, b)\}sA) \\ & \rightarrow send(V, C, \{\{vt\}r\}sA) \\ & \rightarrow usend(V, C, r) \rightarrow Skip(); \end{aligned}$$

Administrator. Similarly, based on administrator knowledge of voter public keys, the administrator can check the signature key of a voter via read event.

$$\begin{aligned} Admin(vt) = \; & read(V, A, \{\Psi(\{vt\}r, b)\}sV) \\ & \rightarrow send(A, V, \{\Psi(\{vt\}r, b)\}sA) \rightarrow Skip(); \end{aligned}$$

Counter. The counter gets the messages from voter, checks the correctness of the signature, and decrypts messages to get the vote.

$$\begin{aligned} Counter(vt) = \; & read(V, C, \{\{vt\}r\}sA) \\ & \rightarrow uread(V, C, r) \rightarrow Skip(); \end{aligned}$$

Intruder. For privacy checking of this protocol, the intruder's ability is eavesdropping. That is, he can observe the passing messages and tries to get the information using his knowledge (except the final message of the protocol as it is sent via an untappable channel).

$$\begin{aligned} & Intruder(vt) \\ & \quad = eavesdrop(V, A, \{\Psi(\{vt\}r, b)\}sV) \rightarrow Skip() \\ & \quad [\,] \; eavesdrop(A, V, \{\Psi(\{vt\}r, b)\}sA) \rightarrow Skip() \\ & \quad [\,] \; eavesdrop(V, C, \{\{vt\}r\}sA) \rightarrow Skip() \end{aligned}$$

System. The system is the interleaving of agent, administrator, counter, and intruder processes:

$System(vt) = Voter(vt) \;|||\; Admin(vt) \;|||\; Counter(vt) \;|||\; Intruder(vt)$;

• **Analysis**
Let $vt1, vt2$ be two values of vote vt.

Anonymity. In anonymity checking, we model the initial knowledge of participants as: the voter, administrator, and counter have all the knowledge to generate and decrypt the messages, whereas the intruder only knows public information such as participant's names, public keys. Note that sA^{-1}, the reverse key of the administrator's signature key sA, is the public key of the administrator and is published to everyone. It happens similarly with sA^{-1} and sA.

- Voter's knowledge:
  $V(V) = \{V, A, C, sV, sA^{-1}, vt1, vt2, \Psi, \Psi^{-1}, b, r\}$.
- Administrator's knowledge:
  $V(A) = \{V, A, C, sA, sV^{-1}\}$.
- Counter's knowledge:
  $V(C) = \{V, A, C, sA^{-1}, sV^{-1}\}$.
- Intruder's knowledge:
  $V(Intruder) = \{V, A, C, sA^{-1}, sV^{-1}\}$.

The SeVe tool returns true in checking the equality of two traces in the anonymity condition. It means that the protocol ensures the anonymity property.

Receipt freeness. In this case, besides the public information, the intruder has some information which the voter shares. In this protocol, that information is Ψ^{-1}, b, and r.

- Voter's knowledge:
  $V(V) = \{V, A, C, sV, sA^{-1}, vt1, vt2, \Psi, \Psi^{-1}, b, r\}$.
- Administrator's knowledge:
  $V(A) = \{V, A, C, sA, sV^{-1}\}$.
- Counter's knowledge:
  $V(C) = \{V, A, C, sA^{-1}, sV^{-1}\}$.
- Intruder's knowledge:
  $V(Intruder) = \{V, A, C, sA^{-1}, sV^{-1}, \Psi, \Psi^{-1}, b, r\}$.

Using the SeVe tool to check the conditions of receipt freeness, we have a counter example. By observing it, we find that the second condition of receipt freeness is not satisfied: by unblinding the first message, the intruder can trace which vote corresponds to this particular voter. Therefore, if the voter tells the intruder he votes $v2$ (whereas he actually votes $v1$), the intruder can decrypt the first message and detect $v1$. Moreover, the voter cannot lie about the values of r and b as this will be detected immediately by the intruder.

Coercion resistance. Because the first condition in coercion resistance is checked similarly to anonymity, we omit it here. Consider the second condition of coercion resistance, which corresponds to the third context. In this context, the intruder has all the information to generate messages, which are used to vote $vt2$. He supplies all those messages to the voter. The voter, on the other hand, based on that information, tries to vote $vt1$ and deceives the intruder.

- Voter's knowledge:
  $V(V) = \{V, A, C, sA^{-1}, sV, vt1, \Psi, \Psi^{-1},$
  $\{\Psi(\{vt2\}r, b)\}sV, \{\{vt2\}r\}sA, r\}$.
- Administrator's knowledge:
  $V(A) = \{V, A, C, sA, sV^{-1}\}$.
- Counter's knowledge:
  $V(C) = \{V, A, C, sA^{-1}, sV^{-1}\}$.
- Intruder's knowledge:
  $V(Intruder) = \{V, A, C, sV, sA^{-1}, vt2, \Psi, \Psi^{-1}, b, r\}$.

The protocol does not satisfy the coercion resistance as the SeVe tool returns a counter example when checking the second condition. The reason is that at the initial state, the intruder knows about b and r, so he can trace whatever the voter votes.

6 Experiments and comparison

In this section, we evaluate the performance of our SeVe tool using some classical security protocols from the literature. Table 2 shows the comparison of our SeVe tool with other three recent popular tools: OFMC, Casper/FDR, and ProVerif. The first column is the name of the protocol. The last four columns are the time needed to check the protocol. The experiments are running on a computer with a Core 2 Duo CPU E6550 2.33Ghz and 4Gb RAM.

From Table 2, we can see that the SeVe tool has better performance than OFMC and ProVerif in protocols not related to algebra operators. SeVe's performance is not as good as Casper/FDR in small size protocols, however, SeVe outperforms Casper/FDR in larger size protocols. This can be explained as in PAT tools and SeVe module, we applied many optimization techniques such as partial order reduction and lazy intruder model, in both generation and verification phases. Nevertheless, SeVe tool underperforms compared to OFMC, ProVerif, and Casper/FDR in protocols related algebra operators. This is because in SeVe, we do not support these operators in language semantics. Therefore, we

Table 2 Experiment results on some security protocols

Protocol	SeVe/s	OFMC/s	Casper/s	ProVerif/s
Needham Pub.	0.015	0.031	0.008	0.019
Andrew RPC	0.227	0.376	0.313	0.293
Denning Saco	0.032	0.101	0.083	0.041
Lowe modified	0.046	0.146	0.11	0.052
Otway-Rees	0.037	0.12	0.040	0.041
Wide-Mouthed	0.011	0.032	0.013	0.025
Yahalom	0.027	0.081	0.019	0.032
Woo-Lam	0.019	0.038	0.016	0.022
Woo-Lam Mut.	0.085	0.137	0.091	0.103
Needham Con.	0.041	0.105	0.036	0.067
Kao Chow Auth.	0.073	0.159	0.104	0.116
Langendor	0.488	0.223	0.140	0.198
TMN	0.351	0.177	0.236	0.228
Gong's Mutual	0.690	0.204	0.311	0.294

need to encode them as defined functions, which take time in verification. SeVe tool also successfully checks other well-known protocols such as "A Fair Non-Repudiation Protocol" [39], "Computer-assisted verification of a protocol for certified email" [40] (these protocols require the checking of participant's knowledge, which is not supported by other tools).

We also demonstrate our tools with three test cases of electronic voting protocols from the literature: the protocol of Fujioka et al. [38], the protocol of Okamoto et al. [41] and the protocol by Lee et al. [42] as shown in Table 3. These protocols were demonstrated in [9]. However, the reasoning task is mainly carried out by hand in that paper. We prove that we can verify these protocols in a fully automatic way. The user can easily specify these protocols using SeVe language and verify privacy type properties by just one click, without any hand-proving step. These protocols specifications and the SeVe tool can be downloaded from the link in [43]. The last column represents the automatic verifier ability of the approach in [9] dealing with the corresponding protocol.

7 Conclusion

Along with the rapid development of network transaction, the verification of security protocol becomes more and more important. Different from other security verification tools which mainly focus on secrecy and authentication properties, we have developed an automatic tool that can verify many types of security goals.

In this paper, we have introduced a framework for modeling security protocols using LTS semantics model. To enhance the ability of verification, we embed the knowledge of participants inside the model, as well as parameterizing the intruder abilities for different specification purposes. Within this framework, we have formally defined the secrecy property, authentication property, and three kinds of privacy-types properties: anonymity, receipt freeness, and coercion resistance, based on observational equivalence and knowledge inference. We also provide some verification algorithms for these properties.

Whereas other previous studies about privacy checking based mainly on hand-proving, our approach is automatic in receipt freeness and coercion resistance checking: the user only need specify the protocol in SeVe language, the verification will be automatically run and return the counter examples, if they exist. This makes privacy verification more reliable and avoids errors. Moreover, by using the automatic approach, we can verify a larger system, which is crucial in practical use.

The results of this study are embodied in the implementation of a SeVe module in the PAT model checker, which supports specifying, simulating, and verifying security protocols for different type of security goals. The experimental results on some classical security protocols prove that our tool is comparasle with other current tools in some aspects.

Table 3 Experimental results on three electronic voting protocols

Protocol	Property	#States	#Transitions	time/s	#Result	Auto in [9]
Fujioka et al.	Anonymity	356	360	0.117	true	yes
Fujioka et al.	Receipt freeness	138	141	0.053	false	no
Fujioka et al.	Coercion resistance	97	104	0.040	false	no
Okamoto et al.	Anonymity	484	492	0.133	true	yes
Okamoto et al.	Receipt freeness	606	611	0.158	true	no
Okamoto et al.	Coercion resistance	131	136	0.068	false	no
Lee et al.	Anonymity	1744	1810	0.715	true	yes
Lee et al.	Receipt freeness	2107	2265	0.849	true	no
Lee et al.	Coercion resistance	2594	2673	0.930	true	no

However, our approach also has some limitations. Firstly, the SeVe tool currently cannot verify an unbounded number of sessions or an unbounded number of voters. To achieve this, we need a technique to abstract those parameters before verification. Secondly, some cryptography terms and algebra properties such as Diffie-Hellman and Exclusive-or operators are also out of scope of current research. In future work, we will extend the SeVe language and verifier to be able to verify protocols related to these problems. We also try to apply other optimization techniques to enhance the ability of our tool in verifying large systems. Another challenge is to extend the SeVe framework to verify other security protocols such as integrity, non-repudiation, and fairness.

Acknowledgements This research wa supported in part by research grant MOE2009-T2-1-072 (Advanced Model Checking Systems) from the ministry of Education, Singapore.

Appendixes

Appendix A: SeVe syntax

In this part, we present the formal syntax of SeVe language. We use EBNF, key words in bold term; terms within square brackets are optional.

Program section

Program ::= **#Variables**
Variables_declare
[**#Function_declare**
Function_declare]
#Initial
Initial_declare
#Protocol_description
Protocol_declare
#System
System_declare
[**#Intruder**
Intruder_declare]
[**#Verification**
Verification_declare]

Declaration section

Variables_declare ::= [**Timestamps:** *List_Id*]
[**Time allow:** *Number*]
Agents: *List_Id*
[**Nonces:** *List_Id*]
[**Public_keys:** *List_Id*]
[**Server_keys:** *List_ServerKey*]
[**Signature_keys:** *List_Id*]
[**Session_keys:** *List_Id*]
[**Constants:** *List_Id*]
[**Functions:** *List_Id*]

Function_declare ::= *Id* **=** *Id*
| *Id* **=** *Id*, *Function_declare*

List_Id ::= *Id*
| *Id*, *List_Id*

List_ServerKey ::= {*List_Id*} **of** *Id*
| {*List_Id*} **of** *Id*, *List_ServerKey*

Initial knowledge section

Initial_declare ::= *Id* **knows** {*msg*}
| *Id* **knows** {*msg*}, *Initial_declare*

Protocol description section

Protocol_declare ::= *Id* → *Id* : *message*
| *Id* → *Id* : *message*,
Protocol_declare

message ::= *msg*[**within**[*number*]][**anonymous**]
[**untappable**]

msg ::= *Id*
| *Id*, *msg*
| {*msg*}*Id*
| *Id*(*msg*) *%function declare*

Actual system section

System_declare ::= *ListAgent*
[**Repeat:** *number*]
ListAgent ::= *Id* : *List_Id*
Id : *List_Id*, *ListAgent*

Intruder section

Intruder_declare ::= **Intruder:** *Id*
[**Intruder_knowledge:** *msg*]
[**Intruder_prepare:** *msg*]
[**Intruder_ability:** *List_ability*]

List_ability ::= [**Inject**], [**Deflect**], [**Eavesdrop**], [**Jam**]

Verification section

Verification_declare ::= [**Data_secrecy:** *list_secrecy*]
[**Authentication:** *list_auth*]
[**Anonymity:** *Id*]
[**Receipt_freeness:** *Id*]
[**Coercion_resistance:** *Id*]

list_secrecy ::= *msg*1 **of** *Id*
| *msg*1 **of** *Id*, *list_secrecy*

list_auth ::= *Id* **is authenticated with** *Id* [**using**{*Id*}]
| *Id* **is authenticated with** *Id* [**using**{*msg*}], *list_auth*

Basic definition

Identifier ::= *letter*{*letter*|*digit*}*
Number ::= '1'..'9' *digit**

letter ::= 'a'..'z'|'A'..'Z'|'_'
digit ::= '0'..'9'

Appendix B: Other semantics

In this part, we introduce the semantics of other operational rules: choice, interleaving, and sequence.

$$\frac{}{(V,\ Skip,\ B)\xrightarrow{\checkmark}(V,\ Stop,\ B)}\ [Skip]$$

$$\frac{(V,\ P,\ B)\xrightarrow{p}(V',\ P',\ B')}{(V,\ P\ []\ Q,\ B)\xrightarrow{p}(V',\ P'\ []\ Q,\ B')}\ [choice1]$$

$$\frac{(V,\ Q,\ B)\xrightarrow{p}(V',\ Q',\ B')}{(V,\ P\ []\ Q,\ B)\xrightarrow{p}(V',\ P\ []\ Q',\ B')}\ [choice2]$$

$$\frac{(V,\ P,\ B)\xrightarrow{p}(V',\ P',\ B')}{(V,\ P\ |||\ Q,\ B)\xrightarrow{p}(V',\ P'\ |||\ Q,\ B')}\ [interleave1]$$

$$\frac{(V,\ Q,\ B)\xrightarrow{p}(V',\ Q',\ B')}{(V,\ P\ |||\ Q,\ B)\xrightarrow{p}(V',\ P\ |||\ Q',\ B')}\ [interleave2]$$

$$\frac{}{(V,\ Skip\ |||\ Skip,\ B)\xrightarrow{p}(V,\ Stop,\ B)}\ [interleave3]$$

$$\frac{(V,\ P,\ B)\xrightarrow{p}(V',\ P',\ B')}{(V,\ P;Q,\ B)\xrightarrow{p}(V',\ P';Q,\ B')}\ [sequence1]$$

$$\frac{(V,\ P,\ B)\xrightarrow{\checkmark}(V',\ P',\ B')}{(V,\ P;Q,\ B)\xrightarrow{\tau}(V',\ P';Q,\ B')}\ [sequence2]$$

Appendix C: Specification of security protocols

Needham Schroeder public key protocol

#Variables
Agents : *a*, *b*;
Nonces : *na*, *nb*;
Public_keys : *ka*, *kb*;

#Initial
a knows {*na*, *ka*};
b knows {*nb*, *kb*};

#Protocol_description
$a \rightarrow b$: {*a*, *na*}*kb*;
$b \rightarrow a$: {*na*, *nb*}*ka*;
$a \rightarrow b$: {*nb*}*kb*;

#System
a : *Alice*;
b : *Bob*;

#Intruder
Intruder : *I*;

#Verification
Data_secrecy : {*na*} *of Alice*;
Agent_authentication :
Alice is_authenticated_with Bob using {*a*},
Bob is_authenticated_with Alice using {*b*};

Electronic voting protocol of Fujioka et al.

Anonymity checking

#Variables
Agents : *V*, *A*, *C*;
Signature_keys : *sV*, *sA*;
Constants : *vt*, *r*, *b*;
Function : Ψ;

#Initial
V knows {*V*, *A*, *C*, *sV*, sA^{-1}, *vt*1, *vt*2, Ψ, Ψ^{-1}, *b*, *r*};
A knows {*V*, *A*, *C*, *sA*, sV^{-1}};
C knows {*V*, *A*, *C*, sA^{-1}, sV^{-1}};

#Protocol_description
$V \rightarrow A$: {Ψ({*vt*}*r*, *b*)}*sV*;
$A \rightarrow V$: {Ψ({*vt*}*r*, *b*)}*sA*;
$V \rightarrow C$: {{*vt*}*r*}*sA*;
$V \rightarrow C$: *r*;

#Intruder
Intruder_knowledge : *V*, *A*, *C*, sA^{-1}, sV^{-1}, Ψ, Ψ^{-1};

$\#Function\_declare$
$\Psi^{-1}(\{\Psi(x,b)\}sA,b) = \{x\}sV;$

$\#Verification$
$Anonymity : vt;$

Receipt freeness checking
$\#Variables$
$Agents : V, A, C;$
$Signature\_keys : sV, sA;$
$Constants : vt, r, b;$
$Function : \Psi;$

$\#Initial$
$V\ knows\ \{V, A, C, sV, sA^{-1}, vt1, vt2, \Psi, \Psi^{-1}, b, r\};$
$A\ knows\ \{V, A, C, sA, sV^{-1}\};$
$C\ knows\ \{V, A, C, sA^{-1}, sV^{-1}\};$

$\#Protocol\_description$
$V \rightarrow A : \{\Psi(\{vt\}r, b)\}sV;$
$A \rightarrow V : \{\Psi(\{vt\}r, b)\}sA;$
$V \rightarrow C : \{\{vt\}r\}sA;$
$V \rightarrow C : r;$

$\#Intruder$
$Intruder\_knowledge : V, A, C, sA^{-1}, sV^{-1}, \Psi, \Psi^{-1}, b, r;$

$\#Function\_declare$
$\Psi^{-1}(\{\Psi(x,b)\}sA,b) = \{x\}sV;$

$\#Verification$
$Receipt\_freeness : vt;$

Coercion resistance checking
$\#Variables$
$Agents : V, A, C;$
$Signature\_keys : sV, sA;$
$Constants : vt, r, b;$
$Function : \Psi;$

$\#Initial$
$V\ knows\ \{V, A, C, sV, sA^{-1}, vt1, vt2, \Psi, \Psi^{-1}, b, r\};$
$A\ knows\ \{V, A, C, sA, sV^{-1}\};$
$C\ knows\ \{V, A, C, sA^{-1}, sV^{-1}\};$

$\#Protocol\_description$
$V \rightarrow A : \{\Psi(\{vt\}r, b)\}sV;$
$A \rightarrow V : \{\Psi(\{vt\}r, b)\}sA;$
$V \rightarrow C : \{\{vt\}r\}sA;$
$V \rightarrow C : r;$

$\#Intruder$
$Intruder\_knowledge : V, A, C, sA^{-1}, sV^{-1}, \Psi, \Psi^{-1}, b, r;$

$\#Function\_declare$
$\Psi^{-1}(\{\Psi(x,b)\}sA,b) = \{x\}sV;$

$\#Verification$
$Coercion\_resistance : vt;$

References

1. Burrows M, Abadi M, Needham R. A logic of authentication. ACM Transactions on Computer Systems, 1990, 8(1): 18–36
2. Syverson P F, van Oorschot P C. On unifying some cryptographic protocol logics. In: Proceedings of 1994 IEEE Symposium on Security and Privacy. 1994, 14–28
3. Paulson L C. The inductive approach to verifying cryptographic protocols. Journal of Computer Security, 1998, 6(1–2): 85–128
4. Bella G, Paulson L C. Kerberos version IV: inductive analysis of the secrecy goals. In: Proceedings of 5th European Symposium on Research in Computer Security. 1999, 361–375
5. Mitchell J C, Mitchell M, Stern U. Automated analysis of cryptographic protocols using Murphi. In: Proceedings of 1997 IEEE Symposium on Security and Privacy. 1997, 141–151
6. Lowe G. Casper: a compiler for the analysis of security protocols. Journal of Computer Security, 1998, 6(1–2): 53–84
7. Blanchet B. Automatic verification of correspondences for security protocols. Journal of Computer Security, 2009, 17(4): 363–434
8. Armando A, Basin D A, Boichut Y, Chevalier Y, Compagna L, Cuéllar J, Drielsma P H, Héam P, Kouchnarenko O, Mantovani J, Mödersheim S, von Oheimb D, Rusinowitch M, Santiago J, Turuani M, Viganò L, Vigneron L. The AVISPA tool for the automated validation of Internet security protocols and applications. In: Proceedings of 17th International Conference on Computer Aided Verification. 2005, 281–285
9. Delaune S, Kremer S, Ryan M. Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security, 2009, 17(4): 435–487
10. Mauw S, Verschuren J, de Vink E P. A formalization of anonymity and onion routing. In: Proceedings of 9th European Symposium on Research Computer Security. 2004, 109–124
11. Halpern J Y, O'Neil K R. Anonymity and information hiding in multiagent systems. Journal of Computer Security, 2005, 13(3): 483–512
12. Jonker H J, de Vink E P. Formalising receipt-freeness. In: Proceedings of 9th International Conference on Information Security. 2006, 476–488
13. Hoare C A R. Communicating Sequential Processes. Upper Saddle River: Prentice-Hall, 1985
14. Schneider S. Verifying authentication protocols in CSP. IEEE Transactions on Software Engineering, 1998, 24(9): 741–758

15. Shahriari H R, Jalili R. Using CSP to model and analyze transmission control protocol vulnerabilities within the broadcast network. In: Proceedings of 2004 International Networking and Communication Conference. 2004, 42–47
16. Schneider S, Delicata R. Verifying security protocols: an application of CSP. In: Proceedings of Symposium on the Occasion of 25 Years of CSP. 2004, 246–263
17. Basin D A, Mödersheim S, Viganò L. An on-the-fly model-checker for security protocol analysis. In: Proceedings of 8th European Symposium on Research in Computer Security. 2003, 253–270
18. Turuani M. The CL-Atse protocol analyser. In: Proceedings of 17th International Conference Term Rewriting and Applications. 2006, 277–286
19. AVISPA project. HLPSL tutorial. http://www.avispa-project.org/package/tutorial.pdf
20. Abadi M, Fournet C. Mobile values, new names, and secure Communication. In: Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 2001, 104–115
21. Schneider S, Sidiropoulos A. CSP and anonymity. In: Proceedings of 4th European Symposium on Research in Computer Security. 1996, 198–218
22. Fournet C, Abadi M. Hiding names: private authentication in the applied Pi calculus. In: Proceedings of 2002 International Symposium on Software Security. 2002, 317–338
23. Kremer S, Ryan M. Analysis of an electronic voting protocol in the applied Pi calculus. In: Proceedings of 14th European Symposium on Programming. 2005, 186–200
24. Backes M, Hritcu C, Maffei M. Automated verification of remote electronic voting protocols in the applied Pi-calculus. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium. 2008, 195–209
25. Dong N, Jonker H L, Pang J. Analysis of a receipt-free auction protocol in the applied Pi calculus. In: Proceedings of 7th International Workshop on Formal Aspects of Security and Trust. 2010, 223–238
26. Pang J, Zhang C. How to work with honest but curious judges? (preliminary report). In: Proceedings of 7th International Workshop on Security Issues in Concurrency. 2009, 31–45
27. Dolev D, Yao A. On the security of public key protocols. IEEE Transactions on Information Theory, 1983, 29(2): 198–208
28. Ryan P, Schneider S. The Modelling and Analysis of Security Protocols: The CSP Approach. New York: Addison-Wesley, 2000
29. Liu Y, Chen W, Liu Y, Sun J. Model checking lineariability via refinement. In: Proceedings of 2nd World Congress on Formal Methods. 2009, 321–337
30. Roscoe A W. Model-checking CSP. In: Roscoe A W, eds. A Classical Mind: Essays in Honour of C. A. R. Hoare. Hertfordshire: Prentice Hall International (UK) Ltd, 1994, 353–378
31. Sun J, Liu Y, Dong J S, Wang H. Specifying and verifying event-based Fairness enhanced systems. In: Proceedings of 10th International Conference on Formal Engineering Methods and Software Engineering. 2008, 5–24
32. Sun J, Liu Y, Dong J S, Pang J. PAT: towards flexible verification under Fairness. In: Proceedings of 21st International Conference on Computer Aided Verification. 2009, 709–714
33. Mahony B, Dong J S. Blending object-Z and timed CSP: an introduction to TCOZ. In: Proceedings of 20th International Conference on Software Engineering. 1998, 95–104
34. Mahony B, Dong J S. Timed communicating object Z. IEEE Transactions on Software Engineering, 2000, 26(2): 150–177
35. Liu S, Offutt A J, Ho-Stuart C, Sun Y, Ohba M. SOFL: a formal engineering methodology for industrial applications. IEEE Transactions on Software Engineering, 1998, 24(1): 24–45
36. Dong J S, Liu S. An object semantic model of SOFL. In: Proceedings of 1st International Conference on Integrated Formal Methods. 1999, 189–208
37. Needham R M, Schroeder M D. Using encryption for authentication in large networks of computers. Communications of the ACM, 1978, 21(12): 993–999
38. Fujioka A, Okamoto T, Ohta K. A practical secret voting scheme for large scale elections. In: Proceedings of 1992 Workshop on the Theory and Application of Cryptographic Techniques. 1992, 244–251
39. Zhou J, Gollmann D. A fair non-repudiation protocol. In: Proceedings of 15th IEEE Symposium on Security and Privacy. 1996, 55–61
40. Abadi M, Blanchet B. Computer-assisted verification of a protocol for certified email. In: Proceedings of 10th International Symposium on Static Analysis. 2005, 316–335
41. Okamoto T. An electronic voting scheme. In: Proceedings of IFIP World Conference on IT Tools. 1996, 21–30
42. Lee B, Boyd C, Dawson E, Kim K, Yang J, Yoo S. Providing receipt-freeness in mixnet-based voting protocols. In: Proceedings of 6th International Conference on Information Security and Cryptology. 2003, 245–258
43. Luu A T. Formal modeling and verifying privacy types properties of security protocols. Technical report, National University of Singapore, 2010, http://www.comp.nus.edu.sg/~ pat/fm/security/

Mr. Anh Tuan LUU received his MSc in Computer Science from the National University of Singapore. Since 2011, he has been a research associate in Centre for Advanced Information Systems at Nanyang Technological University, Singapore. His research interests include formal methods, security protocol verification, semantic web, and bioinformation retrieval.

Dr. Jun SUN received Bachelor and PhD degrees in Computing Science from National University of Singapore (NUS) in 2002 and 2006. In June 2007, he received the LEE KUAN YEW post-doctoral fellowship in the Computer Science Department at the National University of Singapore. In Septemper 2010, he joined Singapore University of Technology and Design as an Assistant Professor. He is a visiting

scientist at MIT (2011–2012). Jun's research is in areas of software engineering and formal methods, in particular, formal specification and verification. He is the co-founder of the PAT model checker.

Dr. Yang LIU graduated in 2005 with a Bachelor of Computing in the National University of Singapore (NUS). In 2010, he obtained his PhD and continued with his post doctoral work in NUS. Dr. Liu specializes in software verification using model checking techniques. This work led to the development of a state-of-the-art model checker, Process Analysis Toolkit. In 2011, Dr. Liu is awarded the Temasek Research Fellowship at NUS as the Principal Investigator in the area of Cyber Security. He continnes to pursue research in the verification of security related software using formal methods such as model checking and theorem proving.

Dr. Jin Song DONG received Bachelor and PhD degree in Computing from University of Queensland in 1992 and 1996. From 1995–1998, he was a Research Scientist at the Commonwealth Scientific and Industrial Research Organisation in Australia. Since 1998 he has been in the School of Computing at the National University of Singapore (NUS) where he is currently Associate Professor. He is co-founder of the PAT model checking system. He is a steering committee member of the International Conference on Formal Engineering Methods (ICFEM) and the Asia Pacific Software engineering Conference (APSEC) series. He was a Chair for many conferences and also on the editorial board of *Formal Aspects of Computing* and *Innovations in Systems and Software Engineering*, A NASA Journal.

Dr. Xiaohong LI received master of engineering degree from Tianjin University, China in 1999, and she received her Ph D in Engineering, major of Computer Science and Applications, from the Department of Computer Science, Tianjin University in Aug., 2005. Since Feb., 2001, she has been working as teacher and researcher in the Department of Computer Science, Tianjin University. Dr. Li is specialized in agent-oriented software engineering, repaid development base on multi-agent system, agent architecture research, agent security, and ambient intelligence.

Dr. Thanh Tho QUAN is a lecturer in the Faculty of Computer Science and Engineering, Hochiminh City University of Technology (HCMUT), Vietnam. He received his BEng degree in Information Technology from HCMUT in 1998 and received his PhD in 2006 from Nanyang Technological University, Singapore. His current research interests include formal methods, program analysis/verification, the Semantic Web, machine learning/data mining and intelligent systems.