Automated Verifying Anonymity and Privacy Properties of Security Protocols

Luu Anh Tuan[1], Jun Sun[2], Yang Liu[1], and Jin Song Dong[1]

[1] School of Computing, National University of Singapore
{tuanluu,liuyang,dongjs}@comp.nus.edu.sg
[2] Singapore University of Technology and Design
sunjun@sutd.edu.sg

Abstract. Security protocols play more and more important role with widely use in many applications nowadays. They are designed to provide security properties for users who wish to exchange messages over unsecured medium. There are many tools were designed to specify and verify security protocols such as Casper/FDR, ProVerif or AVISPA. While most of the existing tools focus on secrecy and authentication properties. few supports properties like anonymity and privacy, which are crucial in many protocols such as in electronic voting systems or anonymous online transactions. Moreover, to the best of our knowledge, there is still not have a fully automatic tool using formal methods to verify these two properties. In this paper, we introduce a framework for specifying security protocols in the Labeled Transition System (LTS) semantics model and give the formal definition for three existing types of anonymity and privacy properties using this model. We also come up with the verification algorithms for verification and implement all the ideas in a module SeVe inside PAT model checker.

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, electronic voting, and so on. These large open networks pose new challenges to security, as trusted and untrusted parties coexist in the system and messages transit through potentially dangerous environment. Properties such as authenticity, confidentiality, proof of identity, proof of delivery, or privacy are difficult to assure in this scenario. Security protocols using cryptographic primitives aim at solving this problem [18]. By a suitable use of shared and public key cryptography, random numbers, hash functions, encrypted and plain messages, a security protocol may assure security requirements for the participants.

Surprisingly, the informal definition of security protocols are usually simple and easy to understand. A protocol typically describes only the actions taken in a complete protocol run between honest principals. Other missing is what happens during unsuccessful runs, for example, with possibly untrusted participants. There has been many research on formally specifying and verifying security protocols such as Casper/FDR [8], ProVerif [4] or AVISPA [1]. Most of them focus on authentication and secrecy properties. Anonymity and privacy properties which take an important roles in many protocols get less attention from researchers. The user may require anonymity and privacy

guarantees to many transactions, such as anonymity of payment, privacy of shopping preference, of email patterns and associations between correspondents, or candidate choice in an election. With all of those requirements, there is a demand on an effectively automated tools to specify and verify security protocols related to those properties.

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

- Simple privacy: the fact that a particular information is not revealed to anyone.
- Receipt freeness: the agent does not gain any receipt which can be used to prove to the intruder that the agent sent a particular information.
- Coercion resistance: the agent cannot cooperate with the intruder to prove to him that the agent sent a particular information.

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

The contribution of our research can be summarized as: we have defined a framework for modeling security protocol in the Labeled Transition System (LTS). Within this framework, we give a formal definition for three types of anonymity and privacy properties: simple privacy, receipt freeness and coercion resistance and show how we can analyze these properties. The verification algorithms for privacy checking are also introduced. Finally, we implement all these ideas inside a module SeVe of PAT model checker to fully automatic specify and verify security protocols. Three test cases in electronic voting protocols in literature are given to show how our verifier works.

Outline: The rest of the paper is organized as follows. Section 2 discusses related works on others formal models and verifications of anonymity and privacy. Operational semantics and model semantics of security protocol are described in Section 3. Section 4 introduces the algorithms used in verification. The completed tool is introduced in Section 5. The experiment and comparison are given in Section 6 and Section 7 finally concludes the paper.

2 Related Works

The definitions for anonymity and privacy properties in natural language are insufficiently precise to allow specification and verification. Benaloh and Tuinstra [3] are the first researchers give the notion of receipt freeness. After that, Benaloh [3] and Okamoto [15] propose some schemes to meet the condition of receipt-freeness but later shown not succesfully because they did not give formal definition of receipt freeness. In [16], Okamoto presents a system resistant to interactive coercers aiming to satisfy coercion resistance, but this property is stated only in natural language. Juels et al. [11] after that proposed a strict definition for coercion resistance.

The idea of formalizing anonymity as some kinds of process equivalence in process algebra was first proposed in the work of Schneider and Sidiropoulos [19]. However, only simple privacy type, i.e. anonymity, was introduced in this paper. The work of

verifying anonymity property in this paper is totally hand-proving and hard to apply in practical systems.

In [7], Fournet and Abadi model the security protocols using applied pi-calculus and observe the observational equivalence in process calculus to prove the anonymity. Similar idea have been used by Mauw et al. [20] as well as Kremer and Ryan [12]. Again, only simple privacy property (or anonymity) is investigated. The recent work of Delaune, Kremer and Ryan [5] give 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 to capture the condition for these two privacy properties, while in our approach, we use knowledge based reasoning to define the condition for them. Moreover, reasoning about bisimulation in this approach is rather informal and mainly by hand which can not prove automatically.

Halpern [9] and Jonker [10] propose the formalizations of anonymity are based on epistemic logics. The authors give a logical characterization of the notion of receipt in electronic voting processes in epistemic logic. However, these formalisms maily focus on reasoning about the property and less suited for modeling the protocol as well as attacker abilities. Their logics aim to expressing properties rather than operational steps of a protocol. Thus, modeling protocols using epistemic-logic requires a high degree of expertise and easily get errors.

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 receive communication 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 [6]. The system is the parallel of agents and intruder activities:

$$System = (||_{X \in \{Agent\}^*}\ Agent_X)\ ||\ Intruder,\ \text{where } || \text{ denotes for parallel.}$$

We start by explaining the number of basic elements of terms sent between agents, such as constant, roles and function. After that we will derive the operation semantics for the rules we use to model the security protocol.

Basic sets. We start with the following sets: $\mathscr{C}$ (denoting constants, such as nonce, session key), $\mathcal{F}$ (denoting function names, such as hash function, bit scheme schema), $\mathcal{A}$ is set of participants, including $\mathcal{A}_{\mathcal{T}}$ denotes trusted agents, $\mathcal{A}_{\mathcal{U}}$ denotes untrusted agent (intruder).

Term. We introduce constructors for pairing and encryption, and we assume that pairing is associative.

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

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: $\_^{-1} : Term \rightarrow Term$.

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

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

At the declaration stage, the knowledge of each participant is defined. During protocol run, that knowledge is enhanced. We donote **V** is the knowledge of participants upon the running of the syste. $\mathbf{V} : \mathcal{A} \rightarrow \mathcal{S}_T$, is the function mapped from set of agents $\mathcal{A}$ to set of terms $\mathcal{S}_T$. The messages are sent and received via an unsecured environment. We use the set **B** of messages as the buffer representing for this environment.

Definition 2 (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, B is the buffer of the messages. At the initial state of the system, 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.*

We have 6 main rules in the operational semantic corresponding to participant activities: Send, Read, Transmit, Deflect, Inject, Eavesdrop and Jam. In addition, we also have other rules for sequence processes and process interleaves. Their semantics will be described in next section.

3.1 Operational semantics

Agent Rules

The 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 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.

$$\frac{p = send(m), Contain(V, m, m.sender)}{(V,\ p \rightarrow Q,\ B) \xrightarrow{p} (V,\ Q,\ B \bigcup \{m'\})} [\ Send\ ]$$

$$\frac{p = read(m), Match(B, m),}{(V,\ p \rightarrow Q,\ B,) \xrightarrow{p} (V',\ Q,\ B \setminus \{m\}),\ V' = AgentUpdate(V, m, m.receiver)} [\ Read\ ]$$

The receiver can compose and decompose pair terms. A term can be encrypted if the sender knows the encryption key, and an encrypted term can be decrypted if the receiver 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).

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

The function $Contain : V \times Message \times \mathcal{A} \rightarrow Boolean$ is defined as:

procedure $Contain(V, message, agent)$
1. $S = V(agent)$;
2. *if* $(S \vdash message.term)$
3. $\quad$ *return True*;
4. *else*
5. $\quad$ *return False*;

Messages from the buffer are accepted by agents if they match a certain pattern, specified in the read event. We define the typed matching predicate $Match : Buffer \times Message \rightarrow Boolean$ to match an incoming message to a message stored in buffer.

procedure $Match(buf, mes)$
1 *foreach* $(b \in Buffer)$
2 $\quad$ *if* $(b == mes)$
3 $\quad\quad$ *return True*;
4 *return False*;

Whenever a message is received, the receiver will decrypt the message (if he can) and get the information. We will represent this decryption as the function $Learn : V \times \mathcal{A} \times Message \rightarrow \mathcal{S}_T$, where $\mathcal{S}_T$ is set of terms .

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

Now, we will give the implementation of the $AgentUpdate : V \times Message \times \mathcal{A}$ as:

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

Intruder rules

If the intruder has eavesdropping capabilities, as stated in the eavesdrop rule, he can learn the message during transmission. The deflect rule states that an intruder with the action capabilities can delete any message from the output buffer. The difference of the jam 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.

$$\frac{m \in B,\ p = \mathit{deflect}(m)}{(V,\ p \rightarrow P,\ B) \xrightarrow{p} (V',\ P,\ B \setminus m),\ V' = \mathit{AgentUpdate}(V, m, \mathit{intruder}).}\ [\ \mathit{deflect}\ ]$$

$$\frac{\mathit{Contain}(V, m, \mathit{intruder}),\ p = \mathit{inject}(m)}{(V,\ p \rightarrow P,\ B) \xrightarrow{p} (V,\ P,\ B \bigcup m)}\ [\ \mathit{inject}\ ]$$

$$\frac{m \in B,\ p = \mathit{eavesdrop}(m)}{(V,\ p \rightarrow P,\ B) \xrightarrow{p} (V',\ P,\ B'),\ V' = \mathit{AgentUpdate}(V, m, \mathit{intruder})}\ [\ \mathit{eavesdrop}\ ]$$

$$\frac{m \in B,\ p = \mathit{jam}(m)}{(V,\ p \rightarrow P,\ B) \xrightarrow{p} (V,\ P,\ B \setminus m)}\ [\ \mathit{jam}\ ]$$

We also have the operational semantics for other rules which we use in SeVe module such as interleaving (denoted as $|||$), external choice (denoted as []) or sequence processes. The operational semantics for them are described in [24].

3.2 Model semantics

In this part, we will investigate on the formalization of anomymity and privacy-type properties properties. The semantics of a model are defined with a labeled transition system (LTS). Let Σ denote the set of all visible events and τ denote the set of all invisible events. Since invisible events are indistinguishable, we sometimes also use τ to represent an arbitrary invisible event. Let Σ^* be the set of finite traces. Let Σ_τ be $\Sigma \cup \tau$.

Definition 3 (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 $\mathit{enabled}(s) = \{e : \Sigma_\tau \mid \exists\, s' \in S, s \xrightarrow{e} s'\}$. We write $s \xrightarrow{e_1, e_2, \cdots, e_n} s'$ iff there exist $s_1, \cdots, s_{n+1} \in S$ such that $s_i \xrightarrow{e_i} s_{i+1}$ for all $1 \leq i \leq n$, $s_1 = s$ and $s_{n+1} = s'$, and $s \xrightarrow{\tau^*} s'$ iff $s = s'$ or $s \xrightarrow{\tau, \cdots, \tau} s'$. The set of states reachable from s by performing zero or more τ transitions is $\tau^*(s) = \{s' : S \mid s \xrightarrow{\tau^*} s'\}$. Let $tr : \Sigma^*$ be a sequence of visible events. $s \xRightarrow{tr} s'$ if and only if there exist $e_1, e_2, \cdots, e_n \in \Sigma_\tau$

such that $s \xrightarrow{e_1,e_2,\cdots,e_n} s'$ and $tr = \langle e_1, e_2, \cdots, e_n \rangle \upharpoonright \tau$ is the trace with invisible events removed. The set of traces of L is $traces(L) = \{tr : \Sigma^* \mid \exists\, s' \in S, init \stackrel{tr}{\Rightarrow} s'\}$.

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

Definition 4 (Refinement and Equivalence). *Let $L_{im} = (S_{im}, init_{im}, T_{im})$ be a LTS for an implementation. Let $L_{sp} = (S_{sp}, init_{sp}, T_{sp})$ be a 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.*

Definition 5 (Unknown knowledge). *Let $L_{im} = (S_{im}, init_{im}, T_{im})$ be a 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 $\forall\, tr = \langle e_1, e_2, .., e_n \rangle \upharpoonright \tau \in traces(L_{im}), \forall\, i \in \{1..n\}\ \mathcal{M}^{tr}_{e_i}(a) \not\vdash t1$, where $M^{tr}_{e_i}(a)$ is the knowledge of agent a of system following trace tr before executing event e_i.*

Given an event e, process P, term x and $x1$, we denote $e[x1/x]$ is a new event which replaces all occurrences of x in e by $x1$, and $P[x1/x]$ is the new process which 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 term e, otherwise $In(x, e) = false$.

Definition 6 (Event renaming function). *Let A be a set of terms, an event renaming function $f_A : \Sigma \rightarrow \Sigma$ 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 $\alpha \notin A$

The process $f_A(P)$ performs the event $f_A(e)$ whenever P would perform e. From this definition, we have the notion of reverse renaming function f_A^{-1}. The process $f_A^{-1}(P)$ can perform any event from the set $f_A^{-1}(e)$ whenever P can perform e. One example of event renaming function will be given in next section.

Simple privacy (anonymity)

For $t \in Term$, we define $simple\_privacy(t)$ is the goal that the protocol will ensure the simple secrecy (or anonymity) on term t. Let $t1$ and $t2$ are two terms representing different values of t. We denote $V0$ as the initial knowledge of the protocol. Consider the process: $System_{im} = System[t1/t]$ [] $System[t2/t]$ ([] denotes for choice) and LTS $L_{im} = (S1, init_{im}, T1)$ where $init_{im} = (V0, System_{im}, \phi)$, $S1 = \{s \mid init_{im} \rightarrow^* s\}$ and $T1 = \{(s_1, e, s_2) : S1 \times \Sigma_\tau \times S1 \mid s_1 \xrightarrow{e} s_2\}$. Denotes $System_{sp} = f^{-1}_{\{t1,t2\}}(f_{\{t1,t2\}}(System_{im}))$ and LTS $L_{au} = (S2, init_{au}, T2)$ where $init_{au} = (V0, System_{au}, \phi)$, $S2 = \{s \mid init_{au} \rightarrow^* s\}$ and $T2 = \{(s_1, e, s_2) : S2 \times \Sigma_\tau \times S2 \mid s_1 \xrightarrow{e} s_2\}$

Definition 7 (Simple privacy (Anonymity)). *The protocol assures simple privacy (or anonymity) on 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 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 means that at anytime t is replaced by $t1$ or $t2$, the intruder cannot observe any difference in traces, so the intruder cannot infer anything about t.

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

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

To see whether the process provides anonymity, we have to consider the traces of:
$f^{-1}_{\{v1,v2\}}(f_{\{v1,v2\}}(Vote_{im}())) =$

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

However, the traces of above process are different traces to $Vote_{im}()$. One of the traces it has is $< Send(V, C, \{v1\}_{PkC}), Read(C, V, r2) >$ which is not possible for $Vote_{im}()$. This indicates that the occurrence of the event $Read(C, V, r2)$ allows a distinction to be made between different events contains $v1, v2$ and so the protocol does not provide anonymity.

Receipt freeness

Similar to simple privacy, receipt freeness can be formalized using event renaming function. In addition, we need to model the fact that the agent is willing to share secret information with the intruder. However, this information should be reliable. We model it by changing the initial knowledge of the system: the knowledge of the intruder will be added the initial knowledge of agents, except some privilege knowledge (such as private key of agent) or unreliable knowledge (such as the key of trap-door commitment scheme as the user can fake and the intruder cannot have any way to detect). We call that initial knowledge is $V1$.

For $t \in Term$, we define $Receipt\_freeness(t)$ is the goal that the protocol will ensure the receipt freeness on term t. Let $t1$ and $t2$ are two terms representing different values of t. Consider the process: $System_{im} = System[t1/t]\ [\,]\ System[t2/t]$

and LTS $L_{im} = (S1, init_{im}, T1)$ where $init_{im} = (V1, System_{im}, \phi)$, $S1 = \{s \mid init_{im} \rightarrow^* s\}$ and $T1 = \{(s_1, e, s_2) : S1 \times \Sigma_\tau \times S1 \mid s_1 \xrightarrow{e} s_2\}$. Denote $System_{sp} = f^{-1}_{\{t1,t2\}}(f_{\{t1,t2\}}(System_{im}))$ and LTS $L_{au} = (S2, init_{au}, T2)$ where $init_{au} = (V1, System_{au}, \phi)$, $S2 = \{s \mid init_{au} \rightarrow^* s\}$ and $T2 = \{(s_1, e, s_2) : S2 \times \Sigma_\tau \times S2 \mid s_1 \xrightarrow{e} s_2\}$. Denote $System_{t1} = System[t1/t]$ and LTS $L_{t1} = (S3, init_{t1}, T3)$ where $init_{t1} = (V1, System_{t1}, \phi)$, $S3 = \{s \mid init_{t1} \rightarrow^* s\}$ and $T3 = \{(s_1, e, s_2) : S3 \times \Sigma_\tau \times S3 \mid s_1 \xrightarrow{e} s_2\}$.

Definition 8 (Receipt freeness). *The protocol assures receipt freeness 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 simple privacy condition. The second condition gives the situation when the agent wants to fake the intruder: while the agent says with the intruder he sent $t2$, he actually sent $t1$. Therefore, the receipt freeness property is hold if the intruder cannot detect the agent is sending $t1$ at every state of traces.

Coercion resistance

Coercion resistance is a strongest property as we give the intruder the ability to communicate interactively with the agent and not only receive information. In this model,the intruder can prepare the message he wants the agent to send.

For $t \in Term$, we define $Coercion\_resistance(t)$ is the goal that the protocol will ensure the coercion resistance on term t. Let $t1$ and $t2$ are two terms representing different values of t. We model the initial knowledge of agent and intruder in this case as: the initial knowledge of intruder is all the information need to generate the message in the session, while the initial knowledge of agent now is only the messages corresponding to the session which t is replaced by $t1$ and $t2$, but without knowing how the intruder constructs them. We call the initial knowledge of this system is $V2$. Consider another case: the intruder knows and wants to vote $t2$, so he supply all the necessary messages for agent to vote in this way; however, he does not know about $t1$. On the contrary, the agent knows $t1$ but he may not know how to contruct the protocol messages to vote $t1$ based on information the intruder supply. We call the initial knowledge of system in this case is $V3$

Consider the process: $System_{im} = System[t1/t]\ [\ ]\ System[t2/t]$ and LTS $L_{im} = (S1, init_{im}, T1)$ where $init_{im} = (V2, System_{im}, \phi)$, $S1 = \{s \mid init_{im} \rightarrow^* s\}$ and $T1 = \{(s_1, e, s_2) : S1 \times \Sigma_\tau \times S1 \mid s_1 \xrightarrow{e} s_2\}$. Denote $System_{sp} = f^{-1}_{\{t1,t2\}}(f_{\{t1,t2\}}(System_{im}))$ and LTS $L_{au} = (S2, init_{au}, T2)$ where $init_{au} = (V2, System_{au}, \phi)$, $S2 = \{s \mid init_{au} \rightarrow^* s\}$ and $T2 = \{(s_1, e, s_2) : S2 \times \Sigma_\tau \times S2 \mid s_1 \xrightarrow{e} s_2\}$. Denote $System_{t1} = System[t1/t]$ and LTS $L_{t1} = (S3, init_{t1}, T3)$ where $init_{t1} = (V3, System_{t1}, \phi)$, $S3 = \{s \mid init_{t1} \rightarrow^* s\}$ and $T3 = \{(s_1, e, s_2) : S3 \times \Sigma_\tau \times S3 \mid s_1 \xrightarrow{e} s_2\}$.

Definition 9 (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. $UnknownKnowledge(L_{t1}, intruder, t1) == true$.

These two conditions are the same conditions of receipt freeness property. The only difference is the knowledge of the participants in the protocols: in case of coercion resistance, the knowledge of the agent comes from the whole messages supplied by the intruder and the intruder but not agent knows how to generate them, while in receipt freeness, the agent knows how to generate those messages and the intruder knowledge is supplied some reliable information carried out by agents.

4 Verification algorithms

In privacy checking, we need to come up 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 as in Fig 1, where $refine$ is denoted for refinement checking function.

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

Fig. 1. Algorithm: $Equivalence(Impl, Spec)$

In this paper, we follow the on-the-fly refinement checking algorithm in [14], which based on the refinement checking algorithms in FDR [17] but applies partial order reduction.

To check the $unknown\ knowledge$ relation, we apply the Depth First Search (DFS) algorithm. The algorithm for $unknown\ knowledge$ checking is presented as in Fig. 2. 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 3.1. In this algorithm, we recorded all the visited states to detect the loop in traces (line 5).

5 Implementation: SeVe module

Model checker *PAT*[3] (Process Analysis Toolkit) is designed to apply state-of-the-art model checking techniques for system analysis. PAT [22][21] supports a wide range of modeling languages. The verification features of *PAT* are abundant in that 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

[3] http://www.patroot.com

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.         endif
11.     endfor
12. endwhile
13. return true;
```

Fig. 2. Algorithm: $unknown\_knowledge(Impl, term, agent)$

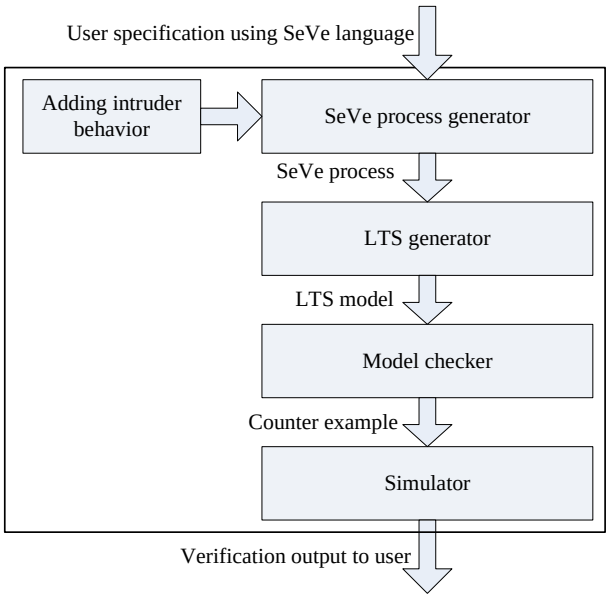

Fig. 3. SeVe architecture

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, temporal behaviors, etc [23]. With all of these advantages, we have implemented the ideas of the above sections into a SeVe module of PAT model checker. The architecture for SeVe module are showed in Fig. 3.

The SeVe language can be considered as the extension of Casper languages [8]; however, we have some amelioration. Firstly, we support many kinds of property checking (integrity, fairness, anonymity, non-repudiation, etc.) for different security purposes. The user also does not need to specify the behavior of the intruder, which is very complicated. By observing the general intruder behavior as in 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 intruder corresponding to different situation. The full grammar of SeVe language is in [24]. In the following, we illustrate an example of SeVe input by considering the a small example of voting protocol involving three agents: voter, admin and counter. The voter and the counter initially share a random number r. The informal description of protocol is:

- $voter$ encryptes the vote vt by r, signs on it and sends to $admin$.
- $admin$ checks the signature of $voter$. If the signature is correct and the $voter$ is legal to vote, the admin signs on the encrypted message and sends back to $voter$.
- $voter$ sends the encrypted message with the sign of $admin$ to $counter$. The $counter$ checks the correctness of admin's signature, decryptes the message to get the vote.

For example, we will examine the receipt freeness property of this protocol. The declaration part and initial knowledge of the protocol in SeVe language are self-explained:

#Variables
Agents: $voter, admin, counter$;
Signature_keys: sv, sa;
Constants: vt, r;

#Initial
$voter$ **knows** $\{sv, vt, r\}$;
$admin$ **knows** $\{sa\}$;
$counter$ **knows** $\{r\}$;

The protocol part and intruder declaration are declared as:

#Protocol_description
$voter \rightarrow admin \ : \{\{vt\}r\}sv$;
$admin \rightarrow voter \ : \{\{vt\}r\}sa$;
$voter \rightarrow counter : \{\{vt\}r\}sa$;

#Intruder
Intruder_name: $Jeeves$;
Intruder_knowledge: r;

The intruder_knowledge is the value supplied by the agent in case of receipt freeness. The verification part is simply declared as:

#Verification
Receipt_freeness: vt;

In the above part, we want the SeVe tool to check the receipt freeness property on vote vt. The user only needs to declare the protocols using SeVe language like above and the tool will automatically generate the LTS model for system, do all verification jobs as well as show the visual result which the user can trace easily. In case of this protocol, the verification result returns false. By looking at counter example, we can find that the second condition of receipt freeness is not satisfied: if the voter tells the intruder he is voting $v2$, but actually he votes $v1$, the intruder can decrypt the first message using value of r supplied by voter to detect $v1$.

6 Experiments and comparison

In this part, we demonstrate our tools with 3 test cases of electronic voting protocols from literature: protocol of Fujioka et al. [2], protocol due to Okamoto et al. [15] and

protocol based on the Lee et al. [13]. These protocols were demonstrated in [5]. However, in that paper, the reasoning task is mainly carried out by hand. 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 types properties by just one click, without any hand-proving step. For briefly, the SeVe specification of the protocols are not introduced here. These specifications of the protocols are fully described in [24]. The SeVe tool can be download from here [4]. The experiment results are summarized in Fig. 4. The experiments were running in Computer with Core 2 Duo CPU E6550 2.33Ghz and 4Gb RAM.

Protocol	**Property**	**#States**	**#Transition**	**Time(s)**	**# Result**
Fujioka et al.	Simple privacy	356	360	0.117s	true
	Receipt freeness	138	141	0.053s	false
	Coercion resistance	97	104	0.040s	false
Okamoto et al.	Simple privacy	484	492	0.133s	true
	Receipt freeness	606	611	0.158s	true
	Coercion resistance	131	136	0.068s	false
Lee et al.	Simple privacy	1744	1810	0.715s	true
	Receipt freeness	2107	2265	0.849s	true
	Coercion resistance	2594	2673	0.930s	true

Fig. 4. Experiment results on three electronic voting protocols

7 Conclusion

In this paper, we have introduced a framework for modeling security protocols using LTS semantics model. Within this framework, we have formally defined three kind of privacy-types properties and apply an some verification algorithms to verify these properties. We create a SeVe module in PAT model checker to implement all these ideas and do some experiments of voting security protocols in literature to show how effectively of our approach.

Different with other previous studies based mainly on hand-proving, our approach is fully automatic: the user only need to specify the protocol in SeVe language, the reasoning will automatically run and return the counter example for the user if exist. This make the proving process is more reliable and avoiding errors. Moreover, by using automatic ways, we can verify larger system, which is crucial in practical use.

In the future work, we will extend the SeVe language and verifier to be able to verify some other cryptography terms and algebra properties such as Deffie-Hellman and

[4] http://www.comp.nus.edu.sg/∼pat/research/

Exclusive-or operators. We also try to apply other optimization techniques to enhance the ability of our tool in verifying large system. Another challenger is to adapt the tool to verify not only for security protocols but also for other security systems such as network layers.

References

1. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. Heam, O. Kouchnarenko, J. Mantovani, S. Modersheim, D. von Oheimb, M. R., J. Santiago, M. Turuani, L. Vigano, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proceedings of CAV'2005*, 2005.
2. T. O. Atsushi Fujioka and K. Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *Advances in Cryptology AUSCRYPT 92*, volume 718, pages 244–251, 1992.
3. J. Benaloh and D. Tuinstra. Receipt-free Secret-ballot Elections. In *Proceedings of 26th Symposium on Theory of Computing (STOC94)*, pages 544–553, 1994.
4. B. Blanchet. Automatic Verification of Correspondences for Security Protocols. volume 19, pages 363–434. Journal of Computer Security, 2009.
5. S. Delaune, S. Kremer, and M. Ryan. Verifying privacy-type properties of electronic voting protocols. In *Journal of Computer Security*, volume 17, pages 435–487, 2009.
6. D. Dolev. and A. Yao. On the Security of Public Key Protocols. In *IEEE Transactions on Information Theory*, volume 29, pages 198–208, 1983.
7. C. Fournet and M. Abadi. Hiding Names: Private Authentication in the Applied pi Calculus. In *Proceedings of International Symposium on Software Security (ISSS02)*, volume 2609, pages 317–338, 2003.
8. L. Gavin. Casper : A Compiler for the Analysis of Security Protocols. In *Journal of Computer Security*, volume 6, pages 53–84, 1998.
9. J. Y. Halpern and K. R. ONeill. Anonymity and Information Hiding in Multiagent Systems. In *Journal of Computer Security*, volume 13, pages 483–512, 2005.
10. H. L. Jonker and E. P. de Vink. Formalising Receipt-Freeness. In *Proceedings of Information Security (ISC06)*, volume 4176, pages 476–488, 2006.
11. A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant Electronic Elections. In *Proceedings of Workshop on Privacy in the Electronic Society (WPES05)*, 2005.
12. S. Kremer and M. D. Ryan. Analysis of an Electronic Voting Protocol in the Applied pi-calculus. In *Proceedings of 14th European Symposium On Programming (ESOP05)*, volume 3444, pages 186–200, 2005.
13. B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Providing Receipt-Freeness in Mixnet-based Voting Protocols. In *Information Security and Cryptology (ICISC03)*, volume 2971, pages 245–258, 2004.
14. Y. Liu, W. Chen, Y. A. Liu, and J. Sun. Model Checking Lineariability via Refinement. In *The sixth International Symposium on Formal Methods (FM 2009)*, pages 321–337, 2009.
15. T. Okamoto. An Electronic Voting Scheme. In *Proceedings of IFIP World Conference on IT Tools*, pages 21–30, 1996.
16. T. Okamoto. Receipt-free Electronic Voting Schemes for Large Scale Elections. In *Proceedings of 5th Int. Security Protocols Workshop*, pages 25–35, 1997.
17. A. W. Roscoe. Model-checking CSP. In *A classical mind: essays in honour of C. A. R. Hoare*, pages 353–378. Prentice Hall International (UK) Ltd, 1994.
18. B. Schneider. Applied Cryptography : Protocols, Algorithms, and Source Code in C. In *John Wiley and Sons.*, 1994.

19. S. Schneider and A. Sidiropoulos. CSP and Anonymity. In *Proceedings of 4th European Symposium On Research In Computer Security (ESORICS96)*, volume 1146, pages 198–218, 1996.
20. SjoukeMauw, J. H. Verschuren, and E. P. de Vink. A Formalization of Anonymity and Onion Routing. In *Proceedings of 9th European Symposium on Research Computer Security (ESORICS04)*, volume 3193, pages 109–124, 2004.
21. J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards Flexible Verification under Fairness. In *21th International Conference on Computer Aided Verification (CAV 2009)*, 2009.
22. J. Sun, Y. Liu, J. S. Dong, and H. Wang. Specifying and Verifying Event-based Fairness Enhanced Systems. In *10th International Conference on Formal Engineering Methods (ICFEM 2008)*, 2008.
23. J. Sun, Y. Liu, J. S. Dong, and X. Zhang. Verifying Stateful Timed CSP Using Implicit Clocks and Zone Abstraction. In *Proceedings of the 11th IEEEInternational Conference on Formal Engineering Methods (ICFEM 2009)*, volume 5885 of *Lecture Notes in Computer Science*, pages 581–600, 2009.
24. L. A. Tuan. Formal Modeling and Verifying Privacy Types Properties of Security Protocols. Technical report, National University of Singapore, 2010. http://www.comp.nus.edu.sg/∼pat/Security.pdf.