Verifying Timed Anonymity of Security Protocols

Yoshinobu Kawabe† and Nobuhiro Ito†
†Department of Information Science, Aichi Institute of Technology, Japan
{ kawabe, n-ito }@kwb.aitech.ac.jp

Abstract - On the Internet anonymity should be provided for many services and protocols. For example, an electronic voting system should guarantee to prevent the disclosure of who voted for which candidate. Trace anonymity is an extension of the formulation of anonymity by Schneider and Sidiropoulos, and we presented an inductive method based on simulation techniques of I/O-automaton theory. In this study we discuss a proof technique for a timed version of trace anonymity. Even though communication patterns are indistinguishable, the sender's identity might be disclosed by detecting the timing of message emission. The sender's identity also might be disclosed by detecting the occurrence of timeout. To deal with such timing features, this study employs timer variables which range over non-negative real numbers. This means that we must deal with an infinite-state system for timed anonymity, but I/O-automaton theory provides various proof techniques for infinite-state systems that incorporate theorem-proving. Our proof method for timed anonymity is based on the conventional I/O-automaton theory, and the existence of an anonymous simulation is shown with two steps. In the first step, we prove the anonymity of an untimed system, and then we extend the anonymity result for the corresponding timed system incrementally.

Keywords: Timed systems, Anonymity, Verification, Formal Method, I/O-automaton

1 INTRODUCTION

We say a security protocol is anonymous if an adversary who can observe all the occurrences of events from the protocol cannot determine who is the "actor" of the events. There are many studies to describe and verify the anonymity of security protocols formally; for example, in [1] a proof technique that incorporates theorem-proving is introduced.

In this paper we discuss the anonymity of timed systems. Recently various real-time systems are used on the Internet. To establish the reliability of timed systems, there have been many studies based on formal methods that modeled and verified the correctness of timed systems [2][3]. To establish anonymity, we should deal with patterns of communication such as the number of messages or the existence/nonexistence of a message. However, even though communication patterns are designed to be indistinguishable, the sender's identity might be disclosed by detecting a timing of message emission. Also, the sender's identity might be disclosed by detecting the occurrence of a timeout. That is, the detection of timing information leads to the disclosure of who is an actor.

We describe a timed system with an I/O-automaton-based formal specification language [4]. This enables us to employ a proof method developed for the anonymity of untimed systems. By introducing a timer variable, we must deal with an infinite-state system. However, I/O-automaton theory [5][6] does not assume finiteness of the number of states or trace length, and it provides a proof technique called a simulation-based method that can handle infinite-state systems directly. In this paper we discuss how to apply a simulation-based method for proving the anonymity of timed systems.

This paper is organized as follows. In section 2, we first describe the basic notion of anonymity, and a proof technique developed in [1]. Then, we present a simple motivating example in Section 3. Also, a timed system is described in IOA language. After showing a basic idea for proving timed anonymity in Section 4, we have discussions in Section 5.

1.1 Related Work

There are studies (e.g. [7]-[9]) that analyze the anonymity of security protocols or communication systems, with dealing with the delay patterns. There are also studies on modeling the anonymity of real-time systems [10][11]. However, it has not been investigated well how to design and verify timed anonymous systems formally.

For untimed systems, a formal modeling of anonymity [12] was introduced based on the applied π-calculus [13][14], and it provides a good framework for verification. However, the notion of timed anonymity is not dealt with. As another approach, an epistemic-logic-based technique is known for modeling the anonymity of multi-agent systems [15]. It might be possible to extend the technique for timed systems by employing temporal logics [16] such as CTL or LTL. However, the anonymity of timed multi-agent systems should be proven by a human prover manually.

Based on Petri net theory [17] or timed I/O-automaton theory [18], there are many studies to verify properties of timed systems. It seems possible to formalize the notion of timed anonymity in such theories directly. However, for that case we must prove both of "the symmetry of communication patterns" and "the symmetry of timing" at the same time; this may be a burden to a protocol designer. In this study, we firstly prove the symmetry of communication patterns only. And then, the result is extended to the timed anonymity incrementally.

A communication system can be described as a state machine. Thus, for finite systems we can employ SAT/SMT-solvers [19]-[21] or model checkers [22][23]; especially, UPPAAL [22] is a well-known model checker for timed systems where we can check properties described in a temporal logic. However, we can see that timed systems are essentially infinite-state. The untimed version of I/O-automaton theory does not assume the finiteness of automata, and it provides techniques to prove the trace inclusion of infinite-state systems, which can be applied with a theorem-proving approach

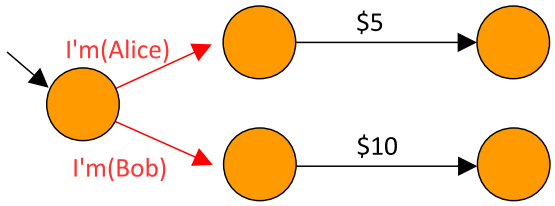

Figure 1: D1

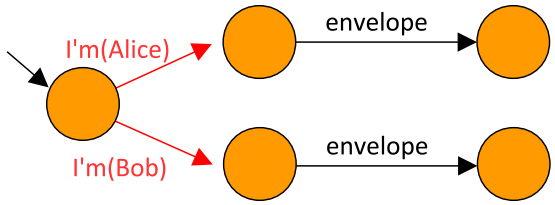

Figure 2: D2

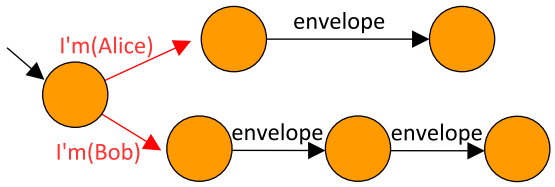

Figure 3: D3

[24]. Hence, in this study we discuss the timed anonymity property with the untimed I/O-automaton theory.

2 PRELIMINARIES

This section explains how to formalize (untimed) anonymity by I/O-automaton. We assume that readers are familiar with the basic notions and notations for I/O-automaton theory and an I/O-automaton-based formal specification language; see also Appendix A.

2.1 Basic Notion of (Untimed) Anonymity

We explain the basic notion of anonymity with the following example.

Example 1 (Donating anonymously) *There are two people, Alice and Bob, and we assume that only one of them has made an anonymous donation. Alice was going to contribute $5, while Bob was going to contribute $10.*

I/O-automaton D1 in Fig. 1 describes the above situation. Actions $5 and $10 of D1 are external actions to represent a donation. After the occurrence of I'm(Alice) or I'm(Bob), either $5 or $10 occurs. Here, I'm(Alice) and I'm(Bob) are actions that specify the donor. For convenience, we call I'm(Alice) and I'm(Bob) *actor actions*. We can see that D1 is anonymous if an adversary who observed all the occurrences of the non-actor actions cannot determine which actor action of D1 will occur.

If an adversary observed that $5 was posted, then the adversary can deduce that Alice made a donation, since action I'm(Alice) can occur in D1 only when action $5 occurs. That is, D1 is not anonymous. One reason for D1 not being anonymous is that an adversary can know how much money was posted. So, we assume that a donation was posted in an envelope. Suppose f is an operation to replace external actions $5 and $10 of D1 with a fresh external action envelope, and we define D2 as f(D1) (see Fig. 2). This operation hides information on how much money was posted. With D2, an adversary who is able to detect the occurrence of envelope cannot deduce which actor action is possible. Hence, D2 is anonymous.

If Bob is going to post $10 in two envelopes each containing $5, then we cannot establish anonymity even though all the messages are encrypted. Figure 3 shows I/O-automaton D3, which describes the above setup. Here, an adversary can determine the identity of a donor by counting the number of times that envelope occurs. Therefore, D3 is not anonymous. This example shows that a system might not be anonymous even though all the messages are encrypted.

Below, we formally discuss the correctness of communication patterns with regard to anonymity. Let X be an I/O-automaton and A be a family with the following conditions: (i) $\bigcup_{A' \in A} A' \subset ext(X)$; (ii) A' and A'' are disjoint for any distinct $A', A'' \in A$. We call A a *family of X's actor actions*, and an element of $\bigcup_{A' \in A} A'$ is called an *actor action* (on A). The occurrences of different actor actions should be indistinguishable to an adversary. That is, if an eavesdropper cannot distinguish the trace set of system X and that of X's "anonymized" version, then we can see that X is anonymous. This is formalized as follows.

Definition 1 *Let X be an I/O-automaton and A be a family of X's actor actions. We define I/O-automaton $anonym_A(X)$ as follows:*

$$
\begin{aligned}
&states(anonym_A(X)) = states(X),\\
&start(anonym_A(X)) = start(X),\\
&ext(anonym_A(X)) = ext(X),\\
&int(anonym_A(X)) = int(X) \text{ and}\\
&trans(anonym_A(X))\\
&\quad = \{(s_1, a, s_2) \mid (s_1, a, s_2) \in trans(X) \wedge a \notin \textstyle\bigcup_{A' \in A} A'\}\\
&\qquad \cup \{(s_1, a, s_2) \mid (s_1, a', s_2) \in trans(X)\\
&\qquad\qquad \wedge A' \in A \wedge a' \in A' \wedge a \in A'\}.
\end{aligned}
$$

If $traces(anonym_A(X)) = traces(X)$ holds, we say X is trace anonymous on A.

2.2 How to Prove Anonymity

This section describes a proof method for anonymity [1].

Definition 2 *Assume X is an I/O-automaton and A is a family of X's actor actions. An anonymous simulation as_A of X on A is a binary relation on $states(X)$ that satisfies the following conditions:*

1. *$as_A(s, s)$ holds for any initial state $s \in start(X)$;*
2. *For any $s_1, s_2, s_1' \in states(X)$ and $a \in sig(X)$, $as_A(s_1, s_1')$ and $s_1 \xrightarrow{a}_X s_2$ implies the following:*

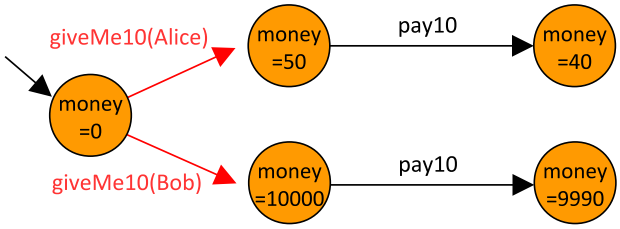

Figure 4: Automaton GMT

If $a \in A'$ holds for some $A' \in A$, for all $a' \in A'$ there is a state s_2' such that $as_A(s_2, s_2')$ and $s_1' \stackrel{a'}{\Longrightarrow}_X s_2'$; otherwise, there is a state s_2' such that $as_A(s_2, s_2')$ and $s_1' \stackrel{a}{\Longrightarrow}_X s_2'$.

I/O-automaton X is trace anonymous on A if X has some anonymous simulation as_A. It is easy to see

$$traces(anonym_A(X)) \supseteq traces(X)$$

since $anonym_A(X)$ has all the transitions of X. The other inclusion

$$traces(anonym_A(X)) \subseteq traces(X)$$

of traces can be shown since as_A is a forward simulation from I/O-automaton $anonym_A(X)$ to I/O-automaton X. A forward simulation from I/O-automaton P to I/O-automaton Q is a binary relation $r \subset states(P) \times states(Q)$ with the conditions shown in [5]; if there is a forward simulation, then we have $traces(P) \subseteq traces(Q)$ ([5], Theorem 3.10).

Proposition 1 *Let X be an I/O-automaton. If there is an anonymous simulation as_A of X on A, then*

$$traces(anonym_A(X)) = traces(X)$$

holds. □

3 ANONYMITY FOR TIMED SYSTEMS

This section discusses a formalization of timed anonymity.

3.1 Example

To explain the notion of timed anonymity, we introduce the following example.

Example 2 *There are two people, Alice and Bob. Alice has $50, while Bob has $10,000. Charlie has requested only one of them to give him $10. We do not know which person makes a payment, but one of them actually sends $10.*

I/O-automaton GMT in Fig. 4 describes the above situation. Action `giveMe10(`mem`)`, where mem is `Alice` or `Bob`, is a special action to represent the actor, and `pay10` is an action for a payment. Automaton GMT has the trace set

$$traces(\mathtt{GMT}) = \left\{ \begin{array}{l} \mathtt{giveMe10(Alice).pay10}, \\ \mathtt{giveMe10(Bob).pay10} \end{array} \right\}.$$

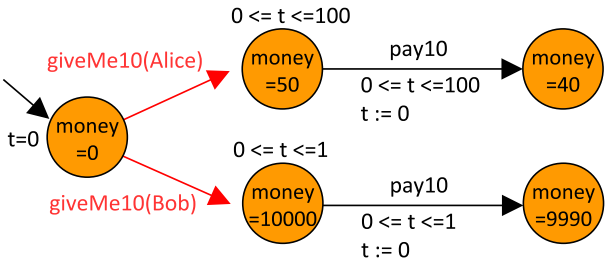

Figure 5: Timed Automaton GMTt

In this case, an adversary who observed the occurrence of action `pay10` cannot determine the preceding action. That is, both of `giveMe10(Alice)` and `giveMe10(Bob)` are possible, so the adversary never knows who made a payment. We can see that this is the same thing of the second setting in Example 1, so GMT is trace anonymous.

In Example 2, Alice possibly pays $10 even though she has only $50. In the following, we would like to consider a modified example.

Example 3 *Bob has much money ($10,000), so he can send $10 immediately. But Alice has only $50. When asked by Charlie, she thinks for a moment before sending $10.*

This is described with a timed automaton in Fig. 5. In this modeling, Alice who has only $50 might take some time up to 100 seconds before sending $10. On the other hand, Bob can make a decision within one second. From this observation, if the payment of $10 occurs after one second, then the payer is Alice. This means that even though communication patterns are indistinguishable, the sender can be identified by detecting the timing of message emission.

3.2 Describing Timed System in IOA

This section describes a timed system with an (infinite-state) untimed I/O-automaton. With examples in the previous section, we explain a basic idea of timed anonymity.

IOA [4] is a formal specification language based on I/O-automaton theory. In IOA, a state is formalized as a tuple of values. Automaton GMT in Fig. 4 is written as follows.

```
automaton GMT
  signaure
                          % AorB = { Alice, Bob }
    output giveMe10(mem: AorB)
    output pay10

  states
     money: Nat  := 0

  transitions
    % This is an actor action.
    output giveMe10(mem)
      pre money = 0
      eff if (mem = Alice) then
            money := 50;
          else
            money := 10000;
```

```
        fi

    output pay10
      pre (money = 50 \/ money = 10000)
      eff money := money - 10
```

Here, two actions `giveMe10(mem)` and `pay10` are defined in a precondition-effect style; `giveMe10(mem)` is an actor action. If we define a candidate binary relation $as_{\mathtt{GMT}}$ as:

$$as_{\mathtt{GMT}}(s, s') \iff s.\mathtt{money} = s'.\mathtt{money} \vee |s.\mathtt{money} - s'.\mathtt{money}| = 9950$$

then the binary relation satisfies the conditions to be an anonymous simulation of automaton `GMT`, where $\alpha.\beta$ represents the value of variable β at state α.

To model a timed system, in this paper we introduce special variables:

- `timer` : a timer variable for elapsing time, and
- `timerFlg` : a flag variable for activating/deactivating the timer.

Thus, we can define the following automaton `GMT2`:

```
automaton GMT2
  signaure
    output giveMe10(mem: AorB)
    output pay10
    internal timerDeactivate

   states
     money: Nat  := 0,
     timer: Real  := 0.0,
     timerFlg: Bool  := true

   transitions
     output giveMe10(mem)
       pre    ~timerFlg
           /\ money = 0
       eff if (mem = Alice) then
             money := 50;
           else
             money := 10000;
           fi
           timerFlg := true

     output pay10
       pre    ~timerFlg
           /\ (money = 50 \/ money = 10000)
       eff money := money - 10;
           timerFlg := true

     % This action is internal and it does
     % not appear in traces.
     internal timerDeactivate
       pre timerFlg
       eff timerFlg := false
```

where we can easily see that $traces(\mathtt{GMT2}) = traces(\mathtt{GMT})$ holds. The value of `timerFlg` should be `false` if either `giveMe10(mem)` or `pay10` is enabled, and `timerFlg` becomes `true` if the action is actually fired. Also, action `timerDeactivate`, which is called a time action, is enabled only if `timerFlg` is `true` and the value of `timerFlg` becomes `false`. This means that a normal action and a time action occur alternately. Note that action `timerDeactivate` does not change the value of `timer` and the time action does not appear in traces since it is internal.

By modifying `GMT2`, we can develop Fig. 5's automaton `GMTt`. Specifically, we remove `timerDeactivate` from `GMT2` and we add the following three actions.

```
output giveMe10Time
  pre    timerFlg
      /\ money = 0
  eff timerFlg := false

output pay10Time(t)
  pre    timerFlg /\ t = timer
      /\ (money = 50 \/ money = 10000)
      /\ ((money = 50)
          => (   0.0 <= timer
              /\ timer <= 100.0))
      /\ ((money = 10000)
          => (   0.0 <= timer
              /\ timer <= 1.0))
  eff timer := 0.0;
      timerFlg := false

output elapse(delta)
  pre    timerFlg /\ delta > 0.0
      /\ (money = 50 \/ money = 10000)
      /\ ((money = 50)
           => (   (   0.0 <= timer
                   /\ timer <= 100.0)
               /\ (   0.0 <= timer + delta
                   /\ timer + delta
                        <= 100.0)))
      /\ ((money = 10000)
           => (   (   0.0 <= timer
                   /\ timer <= 1.0)
               /\ (   0.0 <= timer + delta
                   /\ timer + delta
                        <= 1.0)))
  eff timer := timer + delta
```

Below we classify `GMTt`'s actions as follows:

- Normal actions (`giveMe10(mem)` and `pay10`): appear in the original automaton `GMT`; and
- Time actions (`giveMe10Time`, `pay10Time(t)` and `elapse(delta)`) : are employed for expressing timing constraints and for elapsing time.

In IOA specification `GMTt`, action `giveMe10Time` and its corresponding normal action `giveMe10(mem)` have a common condition "`money = 0`" in their precondition part. Also, actions `pay10Time(t)` and `pay10` have a common condition "`(money = 50 \/ money = 10000)`". Moreover, actions `giveMe10Time` and `pay10Time(t)` do not rewrite variable `money`. Hence, after firing `giveMe10Time` or `pay10Time(t)`, its corresponding normal action is enabled. From this observation, we can see that a one-step transition by action `pay10` in Fig. 5 is formalized with a two-

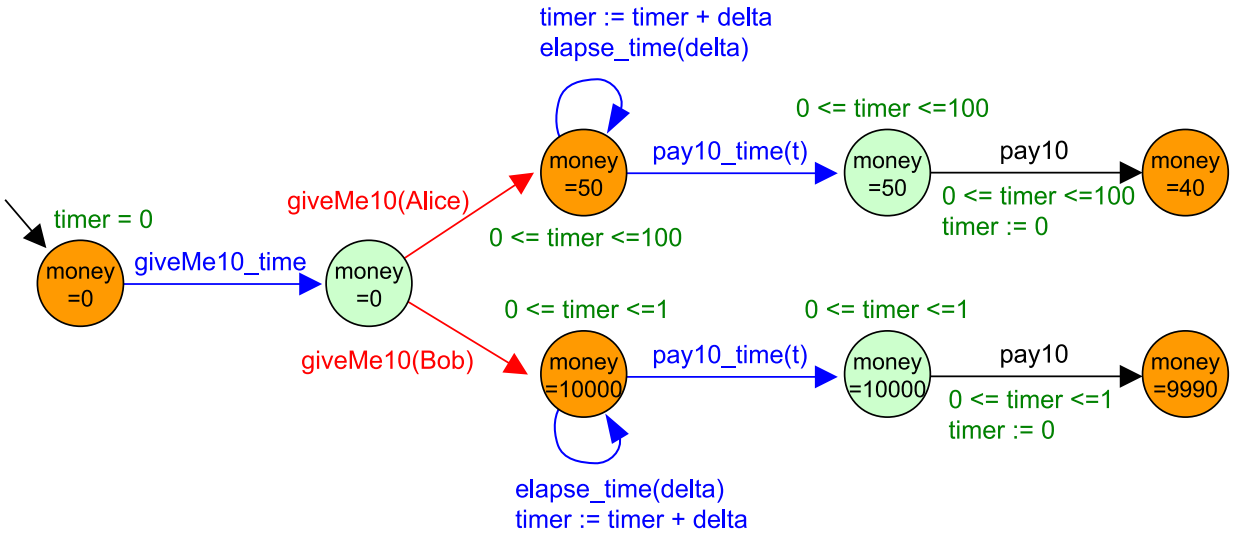

Figure 6: GMTt's Transitions (Expanded in Conventional I/O-Automaton)

step transition sequence with `pay10Time(t)` and `pay10` in IOA language; this is shown in Fig. 6[1].

`GMTt` has another time action, `elapse(delta)`, and the output action is employed for elapsing time. The precondition of `elapse(delta)` defines the timing constraint at time `timer` and at time `time` + `delta`.

4 ANALYZING ANONYMITY FOR TIMED SYSTEMS

This section analyzes the anonymity of timed systems.

4.1 Counterexample for `GMTt`'s Anonymity

Automaton `GMTt` does not have a trace

```
giveMe10Time.giveMe10(Bob).
    elapse(30).pay10Time(30).pay10
```

that represents "Bob is asked and he pays $10 after 30 seconds"; note that Bob must make a payment in one second. However, automaton `GMTt`'s corresponding anonymous system $anonym_{\{\{\texttt{Alice,Bob}\}\}}(\texttt{GMTt})$ has the above trace; that is, we cannot say:

$$traces(\texttt{GMTt}) = traces(anonym_{\{\{\texttt{Alice,Bob}\}\}}(\texttt{GMTt})).$$

This means that $anonym_{\{\{\texttt{Alice,Bob}\}\}}(\texttt{GMTt})$'s anonymity does not lead to `GMTt`'s anonymity. Therefore, `GMTt` is not anonymous.

4.2 Anonymizing `GMTt`

In this section we modify `GMTt`. Specifically, we define:

[1]In Fig. 6, there are green states and orange ones. Time actions can be enabled in orange states only, while in green states non-time actions are enabled. Also, for simplicity of depicting the automaton, a state in this figure actually represents *a collection of states*; for example, a state of "`money = 50`" represents infinitely many states where the value of `timer` ranges over $\{\, t \mid 0 \leq t \leq 100 \,\}$.

```
output pay10Time(t)
  pre    timerFlg /\ t = timer
      /\ (money = 50 \/ money = 10000)
      /\ ((money = 50)
           => (   0.0 <= timer
               /\ timer <= 1.0))
      /\ ((money = 10000)
           => (   0.0 <= timer
               /\ timer <= 1.0))
  eff timer := 0.0;
      timerFlg := false

output elapse(delta)
  pre    timerFlg /\ delta > 0.0
      /\ (money = 50 \/ money = 10000)
      /\ ((money = 50)
           => (   (   0.0 <= timer
                   /\ timer <= 1.0)
               /\ (   0.0 <= timer + delta
                   /\ timer + delta
                        <= 1.0)))
      /\ ((money = 10000)
           => (   (   0.0 <= timer
                   /\ timer <= 1.0)
               /\ (   0.0 <= timer + delta
                   /\ timer + delta
                        <= 1.0)))
  eff timer := timer + delta
```

for `pay10Time(t)` and `elapse(delta)`. That is, we replace "`timer<=100.0`" and "`timer+delta<=100.0`" in actions `pay10Time(t)` and `elapse(delta)` with conditions "`timer<=1.0`" and "`timer+delta<=1.0`", respectively. We call the resulting automaton `GMTt2`. This is to assume Alice responds in one second.

The modified automaton has an anonymous simulation:

$$\begin{aligned} as_{\texttt{GMTt2}}(s, s') \iff\ & as_{\texttt{GMT}}(s, s') \\ & \wedge s.\texttt{timer} = s'.\texttt{timer} \\ & \wedge (s.\texttt{timerFlg} \iff s'.\texttt{timerFlg}). \end{aligned}$$

This formula contains GMT's anonymous simulation relation $as_{\texttt{GMT}}$. With this binary relation, we can prove the anonymity of GMTt2 with the following steps:

1. Find an anonymous simulation for GMT;
2. Then, extend the anonymity result for GMTt2.

In the remainder of this section, we describe why the above proof is possible.

4.2.1 GMTt2's Initial State's Condition

Let $(s, t, p) \in start(\texttt{GMTt2})$ be an initial state of GMTt2, where s is a tuple that represents a state of automaton GMT, t is a value of variable timer, and p is a value of variable timerFlg. From the definition of GMTt2, we have $t = 0.0$ and $u = \texttt{true}$. Clearly, we have $as_{\texttt{GMT}}(s, s)$ implies $as_{\texttt{GMTt2}}((s, 0.0, \texttt{true}), (s, 0.0, \texttt{true}))$.

4.2.2 Step's Correspondence for Normal Actions

A normal action of GMTt2 can be enabled only if the value of variable timerFlg is false. If the action is fired, then variable timerFlg is changed to be true, but timer is not changed. Hence, we can see that for any normal action a we have:

- $(s_1, t, p) \xrightarrow{a}_{\texttt{GMTt2}} (s_1', t, p')$ and
- $as_{\texttt{GMTt2}}((s_1, t, p), (s_2, u, q))$

implies

- We have $t = u$ from the definition of $as_{\texttt{GMTt2}}$;
- We have $p = q = \texttt{false}$ and $p' = \texttt{true}$ since a is a normal action; and
- We have $as_{\texttt{GMT}}(s_1, s_2)$ and $s_1 \xrightarrow{a}_{\texttt{GMT}} s_1'$.

Thus, there exists a state s_2' of GMT such that:

- We have $s_2 \xRightarrow{a'}_{\texttt{GMT}} s_2'$ and $as_{\texttt{GMT}}(s_1', s_2')$;
- $a \in \{\texttt{giveMe10(Alice)}, \texttt{giveMe10(Bob)}\}$ implies $a' \in \{\texttt{giveMe10(Alice)}, \texttt{giveMe10(Bob)}\}$; and
- $a = \texttt{pay10}$ implies $a' = a = \texttt{pay10}$.

Therefore, for the state $(s_2', t, \texttt{true})$, we have:

- $(s_2, u, q) \equiv (s_2, t, \texttt{false}) \xRightarrow{a'}_{\texttt{GMTt2}} (s_2', t, \texttt{true})$, and
- $as_{\texttt{GMTt2}}((s_1', t, p'), (s_2', t, \texttt{true}))$.

Consequently, if binary relation $as_{\texttt{GMT}}$ is an anonymous simulation, then binary relation $as_{\texttt{GMTt2}}$ satisfies a step correspondence condition for any normal action.

4.2.3 Step's Correspondence for Time Actions

If a time action is enabled, the value of timerFlg is true. Also, variables timer and timerFlg can be changed by the time action. Hence, for any time action b, we have:

- $(s_1, t, p) \xrightarrow{b}_{\texttt{GMTt2}} (s_1', t', p')$, and
- $as_{\texttt{GMTt2}}((s_1, t, p), (s_2, u, q))$

implies

- $s_1' = s_1$, $t = u$, and $p = q = \texttt{true}$ holds;
- If b is elapse(delta) then $p' = \texttt{true}$; otherwise, $p' = \texttt{false}$; and
- $as_{\texttt{GMT}}(s_1, s_2)$ holds.

If we can prove

$$(s_2, u, q) \equiv (s_2, t, \texttt{true}) \xRightarrow{b}_{\texttt{GMTt2}} (s_2, t', p')$$

for state (s_2, t', p'), then $as_{\texttt{GMTt2}}((s_1, t', p'), (s_2, t', p'))$ holds. Hence, $as_{\texttt{GMTt2}}$ satisfies the conditions to be an anonymous simulation of GMTt2 for action b.

4.3 Further Analysis for GMTt2

We consider the transition

$$(s_1, t, p) \equiv (s_1, t, \texttt{true}) \xrightarrow{b}_{\texttt{GMTt2}} (s_1, t', p') \equiv (s_1', t', p')$$

shown in the previous section and a transition

$$(s_2, t, \texttt{true}) \xrightarrow{b}_{\texttt{GMTt2}} (s_2, u', q')$$

by time action b. From the definition of each time action, we have $u' = t'$ and $q' = p'$. Moreover, the condition sequence

$$(s_2, u, q) \equiv (s_2, t, \texttt{true}) \xRightarrow{b}_{\texttt{GMTt2}} (s_2, t', p')$$

is actually a one-step transition

$$(s_2, u, q) \equiv (s_2, t, \texttt{true}) \xrightarrow{b}_{\texttt{GMTt2}} (s_2, t', p')$$

since GMTt2 does not have any internal actions. Hence, for GMTt2, we can prove the anonymity by proving:

> For any GMTt2's time action b and any states s_1, s_2 with $as_{\texttt{GMT}}(s_1, s_2)$, if action b is enabled at state (s_1, t, p) then b is also enabled at (s_2, u, q).

Specifically, it suffices to show the following three formulae with a theorem proving tool [24], where enabled(s, a) is true if action a is enabled at state s:

```
(as(s1, s2) /\ enabled(s1, giveMe10Time))
 => enabled(s2, giveMe10Time),
(as(s1, s2) /\ enabled(s1, pay10Time(t)))
 => enabled(s2, pay10Time(t))
```

and

```
(as(s1, s2) /\ enabled(s1, elapse(delta)))
 => enabled(s2, elapse(delta)).
```

5 DISCUSSION

This section discusses the formal proof approaches for timed anonymity.

5.1 Untimed Automaton vs. Timed Automaton

In this study, we described a timed system as an infinite-state system with a conventional I/O-automaton, and we applied the proof method for anonymity [1] directly. As another approach, it seems possible to redefine the anonymity proof technique of [1][25] in timed automaton [26] or in timed I/O-automaton [18]. In this section we compare the approaches.

Timed automaton models are designed for dealing with timing features of computation; so, several constraints are introduced to verify timing properties properly. For example, an execution sequence where only time actions occur infinitely often and normal actions do not occur is regarded as unfair, and unfair execution sequences are prohibited. However, for anonymity verification we might not need such a condition; even though there is an unfair execution sequence by actor Alice in a security protocol, we can discuss the anonymity if the security protocol has its corresponding (unfair) execution sequence by actor Bob.

The untimed I/O-automaton model does not support such conditions, but it has various verification tools and proof methods, and we can use them to prove anonymity. This as an advantage of using untimed I/O-automaton theory. However, in our approach we should introduce a parameter for real-valued times; that is, we must handle infinite-state systems. We can overcome this problem since I/O-automaton theory [5][6] does not assume finiteness of the number of states or trace length, and simulation-based proof techniques are applicable to prove the trace inclusion of infinite-state systems.

We compared the both approaches, and in this study we employed a formal specification language based on conventional I/O-automaton theory. The main reason is that various verification tools are available.

5.2 On Anonymity Proof Method for Security Protocols with Stronger Adversaries

We have introduced an automaton that has variables `timer` and `timerFlg` in this study. A similar approach is employed in [27] to deal with stronger adversaries.

The technique in [1] can only deal with eavesdroppers, and in [27] an adversary model has been introduced to handle stronger adversaries, which may change the protocol's state in various ways, e.g. by sending dummy messages and by rewriting disk image of a PC. This is formalized as follows.

Definition 3 (Attacker part) *Attacker Atk of system X is quadruplet $(states(X), S_{Atk}, A_{Atk}, T_{Atk})$, where a set of attacker's states S_{Atk}, a set of attacker's actions A_{Atk} and a set of attacker's transitions T_{Atk} should satisfy:*

$$\begin{cases} sig(X) \cap A_{Atk} = \emptyset \text{ and} \\ T_{Atk} \subseteq \{((s_1, v_1), a, (s_2, v_2)) \mid s_1, s_2 \in states(X), \\ \qquad\qquad\qquad\qquad v_1, v_2 \in S_{Atk}, \\ \qquad\qquad\qquad\qquad a \in A_{Atk}\}. \end{cases}$$

With the attacker part Atk, automaton (X, Atk) is defined with:

$$\begin{aligned} &states((X, Atk)) = \{(s, v) \mid s \in states(X), v \in S_{Atk}\}, \\ &start((X, Atk)) = \{(s, v) \mid s \in start(X), v \in S_{Atk}\}, \\ &ext((X, Atk)) = ext(X) \cup A_{Atk}, \\ &int((X, Atk)) = int(X), \\ &act((X, Atk)) = act(X), \text{ and} \\ &trans((X, Atk)) \\ &\quad = \{((s_1, v), a, (s_2, v)) \mid (s_1, a, s_2) \in trans(X), v \in S_{Atk}\} \\ &\qquad \cup T_{Atk}. \end{aligned}$$

For automaton (X, Atk), a state $(s, v) \in states((X, Atk))$ has two parts. The second part v is a state of the attacker, and the protocol part X does not change the second part. We can see that, in analogy with the above adversary model, timer-related variables correspond to the attacker's state v, and time actions correspond to attacker's actions T_{Atk}.

This paper has shown a basic idea to prove the anonymity of timed systems, but we have not introduced a formal definition for timed anonymity. We believe that it is possible to introduce such a formal definition as in a similar way of [27].

6 CONCLUSION

This paper discussed a method to verify the anonymity of timed systems. By describing a timed system with an I/O-automaton-based formal specification language, a proof technique for anonymity of untimed systems can be applied to a timed system.

This paper has shown a basic idea to prove the anonymity of timed systems with a small example. As described in Section 5.2 , the formalization of timed anonymity is not complete, and it is an important future work. Also, it is another interesting future work to deal with a larger example such as Mixnet [28]. This is a well-known protocol that relaizes anonymous communication, and we can see that this is a larger and real application. In this study, we have conducted a stepwise verification of anonymity with the following three steps:

1. After describing an untimed version's specification of a communication protocol, we prove the existence of an anonymous simulation;
2. We obtain a corresponding timed version's specification by introducing some time actions;
3. We can easily extend the anonymous simulation relation for the timed system, and we prove that the extended binary relation satisfies the step's correspondence condition for all of the time actions.

We believe that the above approach is also applicable to various real applications, including Mixnet, and it is an interesting future work.

ACKNOWLEDGEMENT

This study is supported by the Grant-in-Aid for Scientific Research (C), No.26330166, of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

REFERENCES

[1] Y. Kawabe, K. Mano, H. Sakurada and Y. Tsukada, "Theorem-proving anonymity of infinite-state systems," *Inf. Proc. Lett.*, Vol. 101, No. 1, pp. 46-51 (2007).

[2] K. van Hee and N. Sidorova, "The Right Timing: Reflections on the Modeling and Analysis of Time," *PETRI NETS 2013*, LNCS 7927, Springer, pp.1-20 (2013).

[3] M. Wehrle and S. Kupferschmid, "Mcta: Heuristics and Search for Timed Systems," *FORMAT 2012*, LNCS 7595, Springer, pp.252-266 (2012).

[4] A. Bogdanov, "Formal verification of simulations between I/O-automata," Master's thesis, MIT (2000).

[5] N. A. Lynch and F. Vaandrager, "Forward and backward simulations — part I: Untimed systems," *Inform. and Comput.*, Vol. 121, No. 2, pp. 214-233 (1995).

[6] N. A. Lynch, *Distributed algorithms*, Morgan Kaufmann Publishers, (1996).

[7] R. E. Newman, V. R. Nalla and I. S. Moskowitz, "Anonymity and Covert Channels in Simple Timed Mix-firewalls," In *PET 2004*, LNCS 3424, Springer-Verlag, pp. 1-16 (2004).

[8] V. C. Perta, M. V. Barbera and A. Mei, "Exploiting Delay Patterns for User IPs Identification in Cellular Networks," In *PETS 2014*, LNCS 8555, Springer-Verlag, pp. 224-243 (2014).

[9] P. Palmieri and J. Pouwelse, "Paying the Guard: an Entry-Guard-based Payment System for Tor," In *FC 2015*, LNCS 8975, Springer-Verlag, pp. 437-444 (2015).

[10] A. Serjantov and R. E. Newman, "On the Anonymity of Timed Pool Mixes," In *Proceedings of the Workshop on Privacy and Anonymity Issues in Networked and Distributed Systems*, Kluwer, Athens, Greece, pp. 427-434 (2003).

[11] G. Tóth and Z. Hornák, "Measuring Anonymity in a Non-adaptive, Real-time System," In *PET 2004*, LNCS 3424, Springer-Verlag, pp. 226-241 (2004).

[12] S. Kremer and M. Ryan. "Analysis of an electronic voting protocol in the applied Pi calculus," In *ESOP '05*, LNCS 3444, Springer-Verlag, pp. 186-200 (2005).

[13] M. Abadi and C. Fournet. "Mobile values, new names, and secure communication," In *POPL '01*, ACM Press, pp. 104-115 (2001).

[14] M. Abadi and C. Fournet. "Private authentication," *TCS*, Vol. 322, pp. 427-476 (2004).

[15] J. Y. Halpern and K. R. O'Neill. "Anonymity and information hiding in multiagent systems," *J. of Computer Security*, Vol. 13, No. 3, pp. 483-514 (2005).

[16] E. M. Clarke Jr., O. Grumberg and D. Peled, *Model Checking*, MIT Press (1999).

[17] W. Reisig, *Understanding Petri Nets*, Springer (2013).

[18] D. Kaynar, N. Lynch, R. Segala and F. Vaandrager, "The Theory of Timed I/O Automata," *Synthesis Lectures on Computer Science*, Morgan Claypool Publishers (2010).

[19] R. M. Podorozhny, S. Khurshid, D. E. Perry and X. Zhang, "Verification of multi-agent negotiations using the alloy analyzer," In *IFM '07*, LNCS 693, Springer-Verlag, pp.501-517 (2007).

[20] Alloy, `http://alloy.mit.edu/alloy/`, Retrieved (2016).

[21] The Yices SMT Solver, `http://yices.csl.sri.com/`, Retrieved (2016).

[22] UPPAAL, `http://www.uppaal.org/`, Retrieved (2016).

[23] NuSMV home page, `http://nusmv.fbk.eu/`, Retrieved (2016).

[24] J. F. Soegaard-Andersen, S. J. Garland, J. V. Guttag, N. A. Lynch and A. Pogosyants. "Computer-assisted simulation proofs," In *CAV '93*, LNCS 697, Springer-Verlag, pp. 305-319 (1993).

[25] I. Hasuo, Y. Kawabe and H. Sakurada, "Probabilistic anonymity via coalgebraic simulations," *TCS*, Vol. 411, No. 22-24, pp. 2239-2259 (2010).

[26] R. Alur and D. Dill, "A theory of timed automata," *TCS*, Vol. 126, pp. 183-235 (1994).

[27] Y. Kawabe and H. Sakurada, "An adversary model for simulation-based anonymity proof," *IEICE Trans.*, Vol. E91-A, No. 4, pp. 1112-1120 (2008).

[28] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *CACM*, Vol. 24, No. 2, pp. 84-90 (1981).

A I/O-AUTOMATON AND IOA LANGUAGE

I/O-automaton theory is a formal system to describe and analyze distributed algorithms. This section provides a brief overview of I/O-automaton theory and IOA formal specification language. A formal definition of an I/O-automaton is given in Section A.1.

A.1 I/O-Automaton Theory

An I/O-automaton X is a tuple

$$(sig(X), states(X), start(X), trans(X))$$

where $sig(X)$ is a set of actions, $states(X)$ is a set of states, $start(X) \subset states(X)$ is a set of initial states, and

$$trans(X) \subset states(X) \times sig(X) \times states(X)$$

is a set of transitions. There are three sorts of actions — input, output and internal. We use $in(X)$, $out(X)$ and $int(X)$ for the sets of input, output and internal actions, respectively. We assume that $in(X)$, $out(X)$ and $int(X)$ are disjoint. We define $ext(X)$ as the union of $out(X)$ and $in(X)$, and an element of $ext(X)$ is called an external action. For simplicity, this paper only deals with I/O-automaton X satisfying $in(X) = \emptyset$; that is, we assume that $ext(X) = out(X)$.

Transition $(s, a, s') \in trans(X)$ is written as $s \xrightarrow{a}_X s'$; we also write $s \rightarrow_X s'$ if a is internal. We define the relation $\twoheadrightarrow_X$ as the reflexive transitive closure of $\rightarrow_X$. For any $a \in sig(X)$ and $s, s' \in states(X)$, we write $s \stackrel{a}{\Longrightarrow}_X s'$ for $s \twoheadrightarrow_X s_1 \xrightarrow{a}_X s_2 \twoheadrightarrow_X s'$ with some $s_1, s_2 \in states(X)$ if a is external, or for $s \twoheadrightarrow_X s'$ if a is internal.

```
automaton channel(i, j: ID)
  signature
    input  send(const i, const j, r:Req)
    output recv(const i, const j, r:Req)

  states
    queue: Seq[MES] so that queue = empty

  transitions
    input send(i, j, r)
      eff queue := packet(i, j, r) -| queue
    output recv(i, j, r)
      pre queue ~= empty
          /\ last(queue) = packet(i, j, r)
      eff queue := init(queue)
```

Note: We employ several pre-defined datatypes or operators, such as datatype `Seq` for a sequence, operators `:=` (substitution), `-|` (appending an element to the head of a sequence), `|-` (appending an element to the tail), `last` (a sequence's tail element) and `init` (deleting the tail element).

Figure 7: Formalizing a communication channel

For any initial state $s_0 \in start(X)$ and transition sequence $\alpha \equiv s_0 \xrightarrow{a_1}_X s_1 \xrightarrow{a_2}_X \cdots \xrightarrow{a_n}_X s_n$, the sub-sequence of $a_1 a_2 \cdots a_n$ that consists of all the external actions is called the *trace* of α. We write $traces(X)$ for the entire set of X's traces. In I/O-automaton theory, various properties of a distributed system can be defined as conditions of a trace set (see Sec. 8.5.3 of [6]), and a proof technique with *forward simulations* is available to show the inclusion of trace sets of two I/O-automata.

Definition 4 *A forward simulation $f \subset states(X) \times states(Y)$ from automaton X to automaton Y is a binary relation with the following:*

1. *For any initial state a of X, there is some initial state b of Y and $f(a, b)$ holds;*
2. *For any reachable states a_1, a_2 of X, any reachable state b_1 of Y and any action π of X, if $f(a_1, b_1)$ and $a_1 \xrightarrow{\pi}_X a_2$ hold then there is a state b_2 that satisfies $f(a_2, b_2)$ and $b_1 \overset{\beta}{\Longrightarrow}_Y b_2$ with $\beta = trace(a_1 \xrightarrow{\pi}_X a_2)$.*

Proposition 2 (Th. 3.10 of [5]) *$traces(X) \subseteq traces(Y)$ holds if there is a forward simulation from X to Y.* □

A.2 IOA Specification Language

In this paper we specify systems in IOA language [4], which is a formal specification language based on I/O-automaton theory. Figure 7 is an IOA example that models a communication channel from agent `i` to agent `j`. IOA specification `channel(i, j)` consists of the following portions:

1. `signature`: declares actions and their sorts;
2. `states`: declares variables. In this example, a variable `queue` for a message sequence is declared with the initial value `empty` and a sort `Seq[MES]`;
3. `transitions`: defines a body for each action, and the body is described in a precondition-effect style. There are two actions in the above example:

   Input action `send(i, j, r)`: attaches message `packet(i, j, r)` to the head of `queue`. Input actions do not have preconditions;

   Output action `recv(i, j, r)`: is executable when `queue` is not `empty` and the last element of `queue` is `packet(i, j, r)`. The effect is to remove `packet(i, j, r)` from `queue`.

A state is formalized as a tuple of values for which a sort is declared in the `states`-part of IOA specification. For example, state set $states$(`channel(i, j)`) of I/O-automaton `channel(i, j)` is

$$\{\, (queue) \mid queue \text{ is a value of sort } \texttt{Seq[MES]} \,\}.$$

(Received October 17, 2016)

Yoshinobu Kawabe received his B.E., M.E. and D.E. degrees in information engineering from Nagoya Institute of Technology in 1995, 1997 and 2003, respectively. He joined NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation in 1997. In 2002, he was a visiting research scientist at MIT Laboratory for Computer Science. Since 2008, he has been with Aichi Institute of Technology, where he is a professor in the Department of Information Science. His research interests include term rewriting systems, process algebras, network programming languages, formal methods and security verification. He is a member of ACM, JSSST, IPSJ and IEICE.

Nobuhiro Ito was born in Aichi, Japan, in 1970. He received the B.E., M.E. and D.E. degrees from Nagoya Institute of Technology, in 1994, 1996 and 1999, respectively. He has been a professor in the department of Information Science, Aichi Institute of Technology, Japan, since 2015. His current research interests are protocols and algorithms for teamwork and contributions for real world by disaster simulations.