Formalizing and Verifying Anonymity of Crowds-Based Communication Protocols with IOA

Yoshinobu KAWABE
Department of Information Science, Aichi Institute of Technology
1247 Yachigusa Yakusa-cho, Toyota, Aichi, Japan
kawabe@aitech.ac.jp

***Abstract*—Crowds is a communication protocol that guarantees sender's anonymity. As a case study, this paper provides a computer-assisted anonymity proof for Crowds. To prove anonymity, we first describe a simple specification of Crowds with an I/O-automaton-based formal specification language. Then, the specification is translated into first-order logic formulae with a formal verification tool. Finally, by showing the existence of an anonymous simulation, the anonymity of Crowds is proved. In this proof, a theorem proving tool is employed. Also, in this study, we formalize an extension of Crowds that guarantees the anonymity with regard to a recipient.**

***Keywords*-anonymity; formal verification; Crowds; theorem-proving**

I. Introduction

On the Internet, there are many services and protocols where anonymity should be provided. For example, an electronic voting system should guarantee anonymity to prevent the disclosure of who voted for which candidate. When such services and protocols are developed, an anonymous communication system, such as Crowds [8], is often employed as a sub-protocol.

It is important to prove the correctness of anonymous communication systems. In the field of software engineering, there are formal method studies that have analyzed distributed systems. There are also formal method studies for anonymity, e.g. [4][9]; the method in [9] is a model-checking approach, and the method in [4] incorporates theorem-proving. In this study, based on the proof method in [4] we verify that a Crowds-based communication protocol is anonymous. To prove the anonymity, this study describes a simple specification of the protocol with a formal specification language. The specification is translated into first-order predicate logic's formulae with a verification tool, and the anonymity of Crowds is proved with a theorem prover. This paper also specifies an extension [5][6] of Crowds that guarantees recipient's anonymity as well as sender's anonymity.

There are already studies [5][6][10] that analyzed Crowds-based communication protocols. To analyze a Crowds-based protocol, in this study we employ I/O-automaton and a theorem proving tool; especially, the author believes that this is the first attempt to describe [5]'s protocol with I/O-automaton.

This paper is organized as follows. Section II illustrates the notion of anonymity and its formalization. In Section III, a formal specification of Crowds is described. In Section IV, the specification is translated into first-order predicate logic's formulae, and the anonymity is verified with a theorem proving tool. Section V formalizes an extension of Crowds that guarantees the anonymity of a recipient. We have discussions in Section VI.

II. Preliminaries

This section first describes notations in I/O-automaton theory [7]. Then, we explain the notion of anonymity and its I/O-automaton-based formalization.

A. I/O-automaton

I/O-Automaton X has a set of actions $sig(X)$, a set of states $states(X)$, a set of initial states $start(X) \subset states(X)$ and a set of transitions $trans(X) \subset states(X) \times sig(X) \times states(X)$. We use $in(X)$, $out(X)$ and $int(X)$ as sets of input, output and internal actions, respectively; that is, $sig(X) = in(X) \cup out(X) \cup int(X)$. We assume that $in(X)$, $out(X)$ and $int(X)$ are disjoint. We define $ext(X) = out(X) \cup in(X)$ whose element 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 $ext(X) = out(X)$.

To formalize anonymity, this paper employs a family of actor action sets $act(X)$ with the following conditions:

- $\bigcup_{A \in act(X)} A \subset ext(X)$
- A and A' are disjoint for any distinct $A, A' \in act(X)$.

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 a 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. For any $s_0 \in start(X)$ and transition sequence $\alpha = s_0 \xrightarrow{a_1}_X s_1 \xrightarrow{a_2}_X \cdots \xrightarrow{a_n}_X s_n$, the trace of α is the sub-sequence of $a_1 a_2 \cdots a_n$ consisting of all the external actions. In addition, we write $traces(X)$ for the entire set of X's traces.

 ISBN: 978-1-61208-204-2

B. Basic notion of 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. I/O-automaton `D1` has an initial state, and only one of `I'm(Alice)` or `I'm(Bob)` is possible at the initial state. Here, `I'm(Alice)` and `I'm(Bob)` are special 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` occurred.

Q1: Suppose an adversary observed that \$5 was posted. Can the adversary deduce who is the donor?

In `D1`, action `$5` can occur only when actor action `I'm(Alice)` occurs. Thus, the adversary can deduce that Alice made a donation. That is, `D1` is not anonymous.

One reason for `D1` not being anonymous is that an adversary can know how much money was posted. To discuss this aspect, the next question is considered.

Q2: A donation was posted in an envelope. Is this donation anonymous?

We consider an operation to replace external actions `$5` and `$10` of `D1` with a fresh external action `env`, and the resulting automaton is called `D2` (see Fig. 1). This operation hides information on how much money was posted, so we can see that this operation formalizes the encryption of messages. With `D2`, an adversary who can detect the occurrence of `env` cannot deduce which actor action is possible. Hence, `D2` is anonymous.

There are cases where we can establish the anonymity by encrypting messages. But, there are cases where we cannot establish the anonymity even though all the messages are encrypted. To explain this, our final question is introduced.

Q3: Bob was going to post \$10 in two envelopes each containing \$5. Is this donation anonymous?

Figure 1 also shows I/O-automaton `D3`, which describes the above setup. In this case, an adversary can determine the identity of a donor by counting the number of time that `env` occurs. Therefore, `D3` is not anonymous. This example shows that a system might not be anonymous even though all the messages are encrypted. Hence, to establish anonymity, we should deal with patterns of communication such as the number of messages or the existence/nonexistence of a message.

C. Formalization of anonymity

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. The anonymized system is formalized as follows.

Definition 1: Let X be an I/O-automaton. We define I/O-automaton $anonym(X)$ as follows:

- $states(anonym(X)) = states(X)$,
- $start(anonym(X)) = start(X)$,
- $ext(anonym(X)) = ext(X)$,
- $int(anonym(X)) = int(X)$,
- $act(anonym(X)) = act(X)$,
- $trans(anonym(X)) = trans(X)$
  $\cup \{(s_1, a, s_2) \mid (s_1, a', s_2) \in trans(X)$
  $\wedge A \in act(X) \wedge a' \in A \wedge a \in A\}$.

Definition 2: I/O-automaton X is trace anonymous if $traces(anonym(X)) = traces(X)$ holds.

For I/O-automata `D1`, `D2` and `D3` in Fig. 1, we can see that

$$\begin{cases} traces(anonym(\mathtt{D1})) \neq traces(\mathtt{D1}) \\ traces(anonym(\mathtt{D2})) = traces(\mathtt{D2}) \\ traces(anonym(\mathtt{D3})) \neq traces(\mathtt{D3}) \end{cases}$$

if we define $act(\mathtt{D1})$, $act(\mathtt{D2})$ and $act(\mathtt{D3})$ as $act(\mathtt{D1}) = act(\mathtt{D2}) = act(\mathtt{D3}) = \{\{\mathtt{I'm(Alice)}, \mathtt{I'm(Bob)}\}\}$. This follows Section II-B's result.

A simulation-based proof method for trace anonymity was introduced in [4].

Definition 3 ([4]): Assume X is an I/O-automaton. An anonymous simulation as of X is a binary relation on $states(X)$ that satisfies the following conditions:

1) $as(s, s)$ holds for any initial state $s \in start(X)$;
2) For any states $s_1, s_2, s_1' \in states(X)$ and action $a \in sig(X)$, $as(s_1, s_1')$ and $s_1 \xrightarrow{a}_X s_2$ implies the following:
   a) If $a \in A$ for some $A \in act(X)$ holds, for all $a' \in A$ there is a state s_2' such that $as(s_2, s_2')$ and $s_1' \overset{a'}{\Longrightarrow}_X s_2'$;
   b) If $a \notin \bigcup_{A \in act(X)} A$, there is a state s_2' such that $as(s_2, s_2')$ and $s_1' \overset{a}{\Longrightarrow}_X s_2'$.

Intuitively, for any states $s_1, s_2 \in states(X)$ and anonymous simulation as, $as(s_1, s_2)$ iff s_1 and s_2 are indistinguishable to an observer. The trace anonymity of an automaton can be proved by finding an anonymous simulation.

Theorem 1 ([4]): If automaton X has an anonymous simulation, X is trace anonymous. □

III. CROWDS AND ITS FORMALIZATION

In this section, an overview of Crowds is described, and we formalize Crowds with an I/O-automaton.

A. Overview of Crowds

Crowds consists of a collection of agents that can communicate with each other (see Fig. 2). To set up a communication path to a website, agents employ the following protocol:

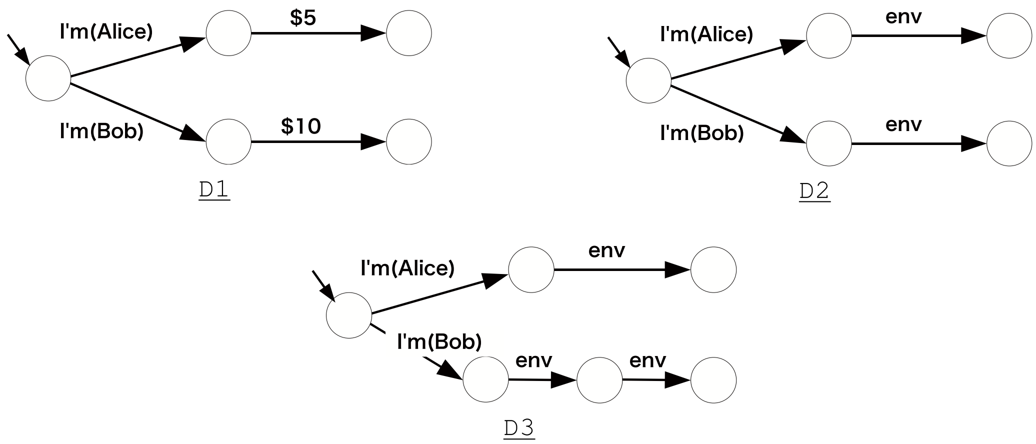

Figure 1. Formalizing Anonymous Donation

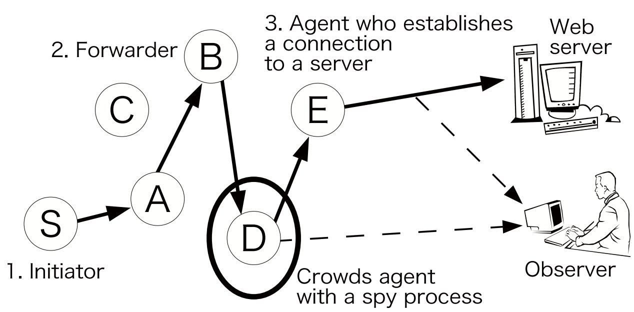

Figure 2. Crowds

The protocol of Crowds

phase 1	An initiator agent first generates a new request;
phase 2	If an agent i has a request, then the agent chooses another agent j randomly and forwards the request to j. By forwarding a request, agents i and j establish a link with regard to the request;
phase 3	After the request has been forwarded several times, some agent establishes a connection to a website.

After making a communication path with the above protocol, the initiator agent connects to a website. We assume that an observer (i.e. an eavesdropper) can observe a connection from a final agent to a website but the observer cannot observe a connection among Crowds agents.

This paper introduces a spy process, which is a computer virus that can read a message from the memory of a Crowds agent and broadcast the message to the public. We say a Crowds agent is ‘corrupt’ if the agent has a spy process. That is, a corrupt Crowds agent can:

- forward a request to another Crowds agent;
- establish a connection to a website; and
- reveal from which agent a request comes.

We say the Crowds system is anonymous if an observer cannot know which agent is the initiator agent. If there is no spy process then the Crowds system is clearly anonymous. However, it is not trivial in case of allowing spy processes.

B. Formalizing Crowds with IOA

Automaton `crowds` in Fig. 3 is a formal specification for Crowds. This is written in IOA, which is an I/O-automaton-based formal specification language. An IOA specification has three portions:

- `signature` declares sorts and actions;
- `states` declares variables and initial values;
- `transitions` defines the body of actions, where each action consists of a precondition (`pre`-part) and an effect (`eff`-part).

A state of automaton `crowds` is a tuple of values `pc`, `mesIsAt`, `mesIsFrom` and `corrp`. These values are as follows:

- `pc` is a program counter of the Crowds system. The value of `pc` ranges over:
  - `init`: Crowds agents waiting for a new request created,
  - `shuffle`: Crowds agents making a communication path, and
  - `terminate`: a communication path to a website established;
- `mesIsAt` is an ID of an agent that has a request;
- `mesIsFrom` is an ID of an agent that had a request in the previous step;
- `corrp` is an array of Boolean values. If `corrp[i]` is true, then agent `i` is corrupt.

Automaton `crowds` has four actions `start(i)`, `pass(i, j)`, `out(i)` and `reveal(i, j)`. Specifically, these actions are as follows:

```
automaton crowds
  signature
    output   start(i:ID)
    internal pass(i:ID, j:ID)
    output   out(i:ID)
    output   reveal(i:ID, j:ID)

  states
    pc:         PC := init,
    mesIsAt:    ID,
    mesIsFrom:  ID,
    corrp:      Array[ID, Bool]

  transitions
    output start(i)  % actor action
      pre pc = init
      eff pc := shuffle;
          mesIsAt := i;
          mesIsFrom := i

    internal pass(i, j)
      pre pc = shuffle /\ i = mesIsAt
      eff mesIsFrom := i;
          mesIsAt := j

    output out(i)
      pre pc = shuffle /\ i = mesIsAt
      eff pc := terminate

    output reveal(i, j)
      pre    pc = shuffle
          /\ i = mesIsFrom
          /\ j = mesIsAt
          /\ corrp[j]
      eff do nothing
```

Figure 3. IOA specification for Crowds

- `start(i)` is an actor action, which represents that agent `i` creates a new request;
- `pass(i, j)` represents that a request is forwarded from agent `i` to agent `j`. This action is introduced as internal, since we assume that the observer cannot observe a connection between Crowds agents;
- `out(i)` represents that a final agent `i` establishes a connection to the website. Since a connection to a website is observable, `out(i)` is defined as an external action.
- `reveal(i, j)` is an action for a spy process.

We can see that actions `start(i)`, `pass(i, j)` and `out(i)` formalize phases 1, 2 and 3 of the Crowds protocol, respectively.

IV. Theorem-proving anonymity of Crowds

This section shows that `crowds` is trace anonymous. In this proof, a theorem proving tool is employed.

A. Translating IOA into first-order logic

Larch [2] is a theorem prover based on first-order predicate logic. I/O-automaton `crowds` is translated into Larch's language by IOA-Toolkit [1]. For example, the following is the result of translation with regard to action `start(i)`:

```
enabled(s, start(i)) <=> (s.pc = init)
effect(s, start(i)).pc = shuffle
effect(s, start(i)).mesIsAt = i
effect(s, start(i)).mesIsFrom = i
effect(s, start(i)).corrp = s.corrp
```

where

- `s.`α is the value of α at state `s`;
- `enabled(s, a)` is true iff action `a` is executable at state `s`; and
- `effect(s, a)` is the successor state of `s` for action `a`.

The first formula is for a precondition of action `start(i)`, and four equations are for a state change by `start(i)`.

B. Computer-assisted anonymity proof for Crowds

Below is a binary relation over $states($`crowds`$)$:

```
as(s, s')
  <=> (s.pc = s'.pc
       /\ (s.corrp[s.mesIsAt]
            <=> s'.corrp[s'.mesIsAt])).
```

This means that states `s` and `s'` are indistinguishable to an observer iff:

- `s.pc` and `s'.pc` are the same; and
- A corrupt agent has a request at state `s` iff a corrupt agent has a request at state `s'`.

We prove that `as` is an anonymous simulation. At first, the condition 1 of Definition 3 is proved. Specifically, we prove the following:

```
% --- Initial state condition
start(s:States[crowds]) => as(s, s)
```

where `start(s)` is true iff state `s` is an initial state. We can easily prove this with the Larch prover.

Then, the step correspondence for actions `pass(i, j)`, `out(i)` and `reveal(i, j)` is proved; that is, we prove condition 2-b in Definition 3. It suffices to show

```
% --- step correspondence condition
% --- for internal action pass(i, j)
(reachable(s1)
 /\ reachable(s1')
 /\ as(s1, s1')
 /\ enabled(s1, a)
 /\ effect(s1, a) = s2
 /\ ~anonymp(a)
 /\ internal(a))
 => (\E s2':States[crowds] (\E a':Actions[crowds]
        (   as(s2, s2')
         /\ enabled(s1', a')
         /\ effect(s1', a') = s2'
         /\ internal(a'))))
```

and

```
% --- step correspondence condition
% --- for output actions (except start(i))
(reachable(s1)
```

```
 /\ reachable(s1')
 /\ as(s1, s1')
 /\ enabled(s1, a)
 /\ effect(s1, a) = s2
 /\ ~anonymp(a)
 /\ output(a))
 => (\E s2':States[crowds]
        (   enabled(s1',
                    pass(s1'.mesIsAt,
                         s1.mesIsFrom))
         /\ enabled(effect(s1',
                           pass(s1'.mesIsAt,
                                s1.mesIsFrom)),
                    pass(s1.mesIsFrom, s1.mesIsAt))
         /\ effect(effect(
                   effect(s1', pass(s1'.mesIsAt,
                                    s1.mesIsFrom)),
                              pass(s1.mesIsFrom,
                                   s1.mesIsAt)), a)
             = s2'
         /\ as(s2, s2')))
```

where

- `reachable(s)` is true iff state `s` is reachable from an initial state;
- `anonym(a)` is true iff `a` is an actor action;
- `internal(a)` is true iff `a` is an internal action; and
- `output(a)` is true iff `a` is an output action.

Finally, we prove the step correspondence for actor action `start(i)`. It suffices to show

```
% --- step correspondence condition
% --- for actor action start(i)
(reachable(s1)
 /\ reachable(s1')
 /\ as(s1, s1')
 /\ enabled(s1, a)
 /\ effect(s1, a) = s2
 /\ a = start(i))
 => (\A i':ID (\E s':States[crowds]
    (\E i'':ID (\E s2':States[crowds]
        (   enabled(s1', start(i'))
         /\ effect(s1', start(i')) = s'
         /\ enabled(s', pass(i', i''))
         /\ effect(s', pass(i', i'')) = s2'
         /\ as(s2, s2'))))))
```

and this is to prove condition 2-a in Definition 3.

All the conditions in this section can be proved with the Larch theorem prover. Consequently, from Theorem 1, we obtain the following result.

Theorem 2: `crowds` is trace anonymous. □

V. Formalizing 3-mode Crowds

Kono et al. introduced an extension [5][6] of Crowds that guarantees recipient's anonymity as well as sender's anonymity. In Crowds, an agent can either:

1) forward a request to another agent; or
2) establish a connection to a website.

We call the former action *mode 1*, and the latter is called *mode 2*. In the extended version, a Crowds agent has another mode, called *mode 3*, where an agent (say, `i`) can change the destination of a request temporarily; the new destination is `i`. By this change, the proper destination is hidden.

```
automaton crowds3mode
  signature
    output   start(i:ID, j:ID)
    internal pass(i:ID, j:ID)
    internal loop(i:ID, j:ID)
    internal out(i:ID)
    output   reveal(i:ID, j:ID)

  states
    pc:         PC := init,
    mesIsAt:    ID,
    mesIsFrom:  ID,
    mesIsTo:    ID,
    corrp:      Array[ID, Bool],
    lst:        Array[ID, List[ID]]
                   := constant(empty)

  transitions
    output start(i, j)
      pre pc = init
      eff pc := shuffle;
          mesIsAt := i;
          mesIsFrom := i;
          mesIsTo := j

    internal pass(i, j)
      pre pc = shuffle /\ i = mesIsAt
      eff mesIsFrom := i;
          mesIsAt := j

    internal loop(i, j)
      pre    pc = shuffle
          /\ i = mesIsAt
          /\ lst[i] = empty
      eff lst[i] := mesIsTo -| empty;
          mesIsTo := i;
          mesIsFrom := i;
          mesIsAt := j

    output reveal(i, j)
      pre    pc = shuffle
          /\ i = mesIsFrom
          /\ j = mesIsAt
          /\ corrp[j]
      eff do nothing

    internal out(i)
      pre    pc = shuffle
          /\ i = mesIsAt
          /\ i = mesIsTo
      eff if lst[i] = empty then
            pc := terminate
          else
            mesIsTo := head(lst[i]);
            lst[i] := empty
          fi
```

Figure 4. Formalization of Crowds with 3 modes

I/O-automaton `crowds3mode` in Fig. 4, which is a modified version of `crowds`, formalizes this extension. For `crowds3mode`, we introduced new variables:

- `mesIsTo`: ID of the destination of a request, and
- `lst`: list of an ID.

Variable `lst` is to store a destination of request and it is used when a Crowds agent changes the destination. For mode 3, automaton `crowds3mode` has action `loop(i, j)`, and actions `start` and `out` are modified.

Verifying the anonymity of `crowds3mode` with a theorem prover is an interesting future work.

VI. Discussions

This section discusses the strength of an adversary and a probabilistic aspect of anonymity.

A. Introducing too strong adversaries cannot establish anonymity

In `crowds`, transition $s_1 \xrightarrow{\texttt{start(i)}} s_2$ by initiator `i` can be simulated by an initiator `j`'s transition sequence

$$s_1 \xrightarrow{\texttt{start(j)}} p \xrightarrow{\texttt{pass(j, i)}} q \xrightarrow{\texttt{pass(i, i)}} s_2.$$

This is essential for the anonymity of `crowds`. In order to construct the transition sequence by `j`, we need the following two conditions:

1) Action `pass` is internal;
2) We can construct a transition sequence that does not contain action `reveal`.

The first condition guarantees that a communication packet is invisible to an observer. We can easily see that system `crowds` may not be anonymous if this requirement is not satisfied; that is, if the occurrences of packets are visible to an observer, then the Crowds system is not anonymous. The second condition is with regard to the timing of attacker's execution. In this paper's example, an agent and its spy process run concurrently, and the spy process may miss to read the agent's memory. If we employ a stronger attacker such that the attacker can execute `reveal(j, j)` immediately after the occurrence of `start(j)`, then an observer knows the identity of the initiator agent.

If an attacker is too strong, we cannot establish the anonymity of a security protocol. This study employed an attacker that was modeled with action `reveal`, and the anonymity of `crowds` was confirmed with a theorem-proving tool.

B. Probabilistic anonymity

This study analyzed Crowds in a nondeterministic setting, since we employed a nondeterministic version of I/O-automaton and a theorem proving tool. However, it is important to deal with probabilities, and Crowds-based protocols are actually analyzed in a probabilistic setting [5][6][8][10]; the original version of Crowds in [8] has a probabilistic anonymity called "probable innocence".

A probabilistic version of anonymous simulation technique is introduced in [3], and probable innocence is proved with this technique for the original version of Crowds. This proof is by induction on the length of execution sequences, and the proof is done by hand; that is, it is not a computer-assisted proof. It is an interesting future work to provide a computer-assisted proof for probable innocence in a probabilistic setting.

VII. Conclusion

This paper presented a computer-assisted anonymity proof of Crowds. Specifically, to enable us to prove the anonymity, we described Crowds with an I/O-automaton, and verified the existence of an anonymous simulation. In this verification, a theorem proving tool based on first-order predicate logic was employed.

This paper also formalized an extended version of Crowds with an I/O-automaton. The extended version `crowds3mode` guarantees the anonymity with regard to the proper recipient. As future work, we are planning to verify the anonymity of `crowds3mode` with a theorem proving tool.

Acknowledgment

This study is supported by the Grant-in-Aid for Young Scientists (B), No.23700024, of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

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

[2] S. J. Garland, J. V. Guttag, J. J. Horning: An overview of Larch, LNCS 693, pp. 329–348. Springer (1993).

[3] I. Hasuo, Y. Kawabe, H. Sakurada, Probabilistic anomymity via coalgebraic simulations, Theoretical Computer Science, Vol. 411, No. 22-24, pp. 2239-2259 (2010).

[4] Y. Kawabe, K. Mano, H. Sakurada, Y. Tsukada: Theorem-proving anonymity of infinite-state systems, Information Processing Letters, vol. 101, no. 1, pp. 46–51 (2007).

[5] K. Kono, Y. Ito, N. Babaguchi: Anonymous communication system using probabilistic choice of actions and multiple loopbacks, Proc. Information Assurance and Security (IAS), pp. 210-215 (2010).

[6] K. Kono, Y. Ito, N. Babaguchi: Anonymous communication system based on multiple loopbacks, Journal of Information Assurance and Security, vol. 6, no. 2, pp. 124-131 (2011).

[7] N. A. Lynch: Distributed algorithms, Morgan Kaufmann Publishers (1996).

[8] M. K. Reiter, A. D. Rubin: Crowds: anonymity for Web transactions, ACM Trans. on Information and System Security, vol. 1, no.1, pp. 66-92 (1998).

[9] S. Schneider, A. Sidiropoulos: CSP and anonymity, Proc. ESORICS '96, LNCS 1146, pp. 198–218, Springer (1996).

[10] V. Shmatikov: Probabilistic model checking of an anonymity system, Journal of Computer Security, vol. 12, no. 3/4, pp. 355-377 (2004).