Authentication Revisited: Flaw or Not, the Recursive Authentication Protocol

Guoqiang Li[1] and Mizuhito Ogawa[2]

[1] NCES, Graduate School of Information Science, Nagoya University
li.g@nces.is.nagoya-u.ac.jp
[2] Japan Advanced Institute of Science and Technology
mizuhito@jaist.ac.jp

Abstract. Authentication and secrecy have been widely investigated in security protocols. They are closely related to each other and variants of definitions have been proposed, which focus on the concepts of corresponding assertion and key distribution. This paper proposes an *on-the-fly model checking* method based on the pushdown system to verify the authentication of recursive protocols with an unbounded number of principals. By experiments of the Maude implementation, we find the recursive authentication protocol, which was verified in the sense of (weak) key distribution, has a flaw in the sense of correspondence assertion.

1 Introduction

Security protocols, although each of them only contains several flows, easily cause attacks even without breaking cryptography algorithms. Design and analysis of security protocols have been a challenging problem over 30 years.

Woo and Lam proposed two goals for security protocols, *authentication* and *key distribution* [1]. By authentication, we mean that after termination of the protocol execution, a principal should be assured that it is "talking" to the intended principal. Key distribution means that if a principal receives a session key, then only the principal who sent the key (and the server) knew the key. They also gave the formal definitions: authentication is defined as *correspondence assertion*, and key distribution is defined as *secrecy*. Note that this secrecy is stronger than the one widely used later [2,3]. Correspondence assertion is later widely used to define the authentication [2,3,4]. The intuitive meaning is, when B claims the message it accepted from A, then A exactly sent the same message.

These properties has various different points of view. For instance, Bellare et. al. stated that key distribution is "very different from" authentication [5]. Bella pointed out that two goals "are strictly related" and "might be equivalent" [4].

Paulson et al. formally defined the key distribution[1], which intuitively means, if a principal receives a session key, then only the principal who sent the key (and the server) *can know* the key [4,6]. Its difference from the key distribution Woo and Lam defined is quite subtle, since "can know" implies "may not know". In

[1] This "key distribution" is weaker than what Woo and Lam has defined in [1].

Cha et al. (Eds.): ATVA 2008, LNCS 5311, pp. 374–385, 2008.

their sense of key distribution, Paulson proved the correctness of the *recursive authentication protocol* (referred to as the RA protocol) [6].

This paper proposes an *on-the-fly model checking* method [7,8,9] based on the pushdown system to verify the authentication property of recursive protocols with an unbounded number of principals. By experiments with the Maude implementation, we find out that the RA protocol has a flaw in the sense of correspondence assertion.

The model checking method tackles various sources of infinity in the verification of the RA protocol. Our main ideas are summarized as:

- Lazy instantiation on messages, i.e., message contents that do not affect protocol actions will be left unsubstantiated.
- Lazy instantiation on names, i.e., names, such as encryption keys, are extended from constants to terms, and left uninstantiated until actual principals are assigned during communications.
- Identification of fresh messages by contexts, i.e., since the RA protocol does not repeat the same context (i.e., once pop starts, never push again), each nonce in a session is identified by the stack content.

The first idea is realized by a *parametric semantics* and a *refinement step*. The second and the third ideas are realized by *binders* [7]. These ideas supply sound and compete model checking for verifying authentication of the RA protocol.

Note that this methodology covers only a restricted class of recursive protocols, which are described by *sequential recursive processes*. To the best of our knowledge, this is the first model checking applied to recursive protocols.

This paper is organized as follows. Section 2 presents an environment based process calculus for security protocol descriptions, and a trace equivalence to specify the authentication property. Section 3 shows how to describe and analyze the RA protocol in our setting. The encoding of the pushdown system and experimental results by Maude are reported in Section 4. Section 5 presents related work, and Section 6 concludes the paper.

Due to the lack of space, we omit detailed explanations, examples and theorems; these can be found in the extended version [10].

2 A Process Calculus for Security Protocol Descriptions

2.1 The Syntax of the Calculus

Assume three disjoint sets: $\mathcal{L}$ for *labels*, $\mathcal{B}$ for *binder names* and $\mathcal{V}$ for *variables*. Let $a, b, c, \ldots$ denote labels, let $\mathtt{m}, \mathtt{n}, \mathtt{k}, \ldots$ for binder names, and let $x, y, z, \ldots$ for variables.

Definition 1 (Messages). *Messages $M, N, L \ldots$ in a set $\mathcal{M}$ are defined iteratively as follows:*

$$pr ::= x \mid \mathtt{m}[pr, \ldots, pr]$$
$$M, N, L ::= pr \mid (M, N) \mid \{M\}_L \mid \mathcal{H}(M)$$

A message is ground, if it does not contain any variables.

- pr ranges over a set of undecomposable *primary messages*.
- A binder, $\mathtt{m}[pr_1, \ldots, pr_n]$ is an atomic message indexed by its parameters, $pr_1, \ldots, pr_n$. A binder with 0 arity is named a *name*, which ranges over a set $\mathcal{N}$ ($\mathcal{N} \subseteq \mathcal{B}$).
- (M, N) represents a *pair* of messages.
- $\{M\}_L$ is an *encrypted message* where M is its *plain message* and L is its *encryption key*.
- $\mathcal{H}(M)$ represents a one-way *hash function message*.

Definition 2 (Processes). *Let $\mathcal{P}$ be a countable set of processes which is indicated by $P, Q, R, \ldots$. The syntax of processes is defined as follows:*

$$\begin{aligned} P, Q, R ::= {} & \mathbf{0} \mid \overline{a}M.P \mid a(x).P \mid [M = N]\, P \mid (\mathtt{new}\ x : \mathcal{A})P \mid (\nu\, n)P \mid \\ & let\ (x, y) = M\ in\ P \mid case\ M\ of\ \{x\}_L\ in\ P \mid \\ & P \| Q \mid P + Q \mid P; Q \mid \mathbb{A}(\tilde{pr}) \end{aligned}$$

Variables x and y are bound in $a(x).P$, $(\mathtt{new}\, x : \mathcal{A})P$, $let\ (x, y) = M\ in\ P$, and $case\ M\ of\ \{x\}_L\ in\ P$. The sets of free variables and bound variables in P are denoted by $f_v(P)$ and $b_v(P)$, respectively. A process P is closed if $f_v(P) = \emptyset$. A name is free in a process if it is not restricted by a restriction operator ν. The sets of free names and local names of P are denoted by $f_n(P)$ and $l_n(P)$, respectively.

Their intuition is,

- $\mathbf{0}$ is the *Nil* process that does nothing.
- $\overline{a}M.P$ and $a(x).P$ are *communication processes*. They are used to describe sending message M, and awaiting an input message via x, respectively.
- $(\mathtt{new}\ x : \mathcal{A})P$ and $(\nu\, n)P$ are *binding processes*. The former denotes that x ranges over $\mathcal{A}$ ($\subseteq \mathcal{N}$) in P; The latter denotes that the name n is local in P.
- $[M = N]\, P$, $let\ (x, y) = M\ in\ P$ and $case\ M\ of\ \{x\}_L\ in\ P$ are *validation processes*. They validate whether the message M is equal to N, whether it is a pair, and whether it is an encrypted message, respectively.
- $P \| Q$, $P + Q$, and $P; Q$ are *structure processes*. $P \| Q$ means that two processes run concurrently; $P + Q$ means nondeterministic choices of a process; $P; Q$ means when P terminates, then Q runs.
- For each *identifier* $\mathbb{A}(pr_1, \ldots, pr_n)$, there is a unique definition, $\mathbb{A}(pr_1, \ldots, pr_n) \triangleq P$, where the $pr_1, \ldots, pr_n$ are free names and variables in P.

We assume a set of *identifier variables*, X will range over identifier variables. A *process expression* is like a process, but may contain identifier variables in the same way as identifers. E, F will range over process expressions.

Definition 3 (Recursive process). *A recursive process is defined as an identifier, with the format, $\mathbb{A}_i \triangleq E(\mathbb{A}_1, \ldots, \mathbb{A}_i, \ldots, \mathbb{A}_n)$.*

If a process is not a recursive process, we name it a *flat process*.

Definition 4 (Sequential). *Let E be any expression. We say that an identifier variable X is* sequential *in E, if X does not occur in any arguments of parallel compositions. An expression E is sequential if all variables in E are sequential. A sequential process is an identifier defined by an sequential expression.*

2.2 Characterizations and Restrictions on the Process Calculus

We use an environment-based process calculus [3], while traditional process calculi, such as π-calculus [11], use channel-based communications. There are several notable differences between two types of calculi.

- Communications.
  - In channel-based calculi, two processes communicate through a specific channel. For example, a communication in π-calculus [11] is,

    $$((\nu\, z)\overline{x}\, z.P) \mid x(y).Q \mid R \longrightarrow^{+} ((\nu\, z)P \mid Q\{z/y\}) \mid R$$

    The first process sends a local name z through the channel x, while the second process awaits a name via y on the same channel x. Thus the name z will be communicated between two processes.
  - In the environment-based process calculus, all processes communicate through a public environment, which records all communicated messages. The calculus is thus natural to describe a hostile network.
- Freshness of names.
  - Channel-based calculi adopt scopes of local names for fresh names. In the example above, the scope of z enlarges after the transition. Although R is included in the system, it cannot "touch" the z during the transition. Due to α-conversation, z can be substituted to any fresh name.
  - All local names in the environment-based process calculus will be substituted to fresh public names during transitions. Since when two principals exchange a message through a hostile network, we assume that all other principals will know the message. Several techniques will be performed to guarantee that each public name is fresh to the whole system.
- Infinitely many messages that intruders and dishonest principals generate.
  - Channel-based calculi adopt recursive processes to generate these messages. Thus even describing a simple protocol, the system is complex [12].
  - The environment based process calculus adopt deductive systems to generate the messages generated by intruders and dishonest principals [3,8]. Security protocols can be described in a straightforward way.

For both types of calculi, there are two representations for infinite processes, *identifiers* and *replications*. Identifiers can represent recursive processes. Replications take the form $!P$, which intuitively means an unbounded number of concurrent copies of P. For fitness to model as a pushdown system, we choose identifiers with the sequential restriction.

2.3 Trace Semantics and Equivalence

An *environmental deductive system* (represented as $\vdash$, see Appendix **??**) generates messages that intruders can produce, starting from the the logged messages. It produces, encrypts/decrypts, composes/splits, and hashes messages.

An *action* is a term of form $\overline{a}M$ or $a(M)$. It is ground if its attached message is ground. A string of ground actions represents a possible run of the protocol,

if each input message is deduced by messages in its prefix string. We named such a kind of string (concrete) *trace*, denoted by $s, s', s'', \ldots$. The messages in a concrete trace s, denoted by $msg(s)$, are those messages in output actions of the concrete trace s. We use $s \vdash M$ to abbreviate $msg(s) \vdash M$.

Definition 5 (Concrete trace and configuration). *A concrete trace s is a ground action string, satisfying each decomposition $s = s'.a(M).s''$ implies $s' \vdash M$. A concrete configuration is a pair $\langle s, P \rangle$, in which s is a concrete trace and P is a closed process.*

The extended version [10] presents the trace semantics, the parametric semantics and a refinement step as the lazy instantiation. We proved the sound and complete correspondence between two semantics [7,9].

Abadi and Gordon adopted *testing equivalence* to define security properties [2], in which the *implementation* and the *specification* of a security protocol are described by two processes. If they satisfy the equivalence for a security property, the protocol guarantees the property.

Testing equivalence is defined by quantifying the environment with which the processes interact. Intuitively, the two processes should exhibit the same traces under arbitrary *observers* (as intruders). In our calculus, capabilities of intruders are captured by the environmental deductive system. Thus, a *trace equivalence* is directly applied for the authentication property without quantifying observers.

For simplicity, we say a concrete configuration $\langle s, P \rangle$ *generates* a concrete trace s', if $\langle s, P \rangle \longrightarrow^* \langle s', P' \rangle$ for some P'.

Definition 6 (Trace equivalence). *P and Q are trace equivalent, written $P \sim_t Q$, if for all trace s, P generates s if and only if Q generates s.*

3 Analysis of the Recursive Authentication Protocol

3.1 The Recursive Authentication Protocol

The recursive authentication protocol is proposed in [13]. It operates over an arbitrarily long chain of principals, terminating with a key-generated server.

Assume an unbounded number of principals intending to generate session keys between each two adjacent principals by contacting a key-generated server once. Each principal either contacts the server, or forwards messages and its own information to the next principal. The protocol has three stages (see Fig. 1): *Communication stage.* Each principal sends a request to its next principal, composing its message and the message accepted from the previous one. *Submission stage.* One principal submits the whole request to the server. *Distribution stage.* The server generates a group of session keys, and sends back to the last principal. Each principal distributes the session keys to its previous principal.

The RA protocol is given informally as follows. For simplicity, we use a convenient abbreviation of the hash message,

$$\mathcal{H}_K(X) = (\mathcal{H}(K, X), X)$$

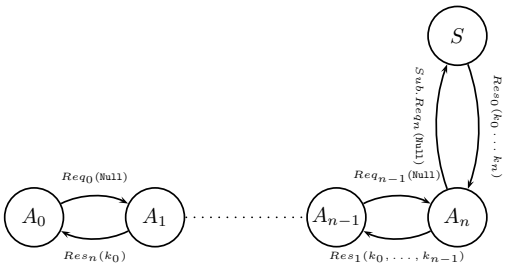

Fig. 1. The Recursive Authentication Protocol

Communication Stage

$$A_0 \longrightarrow A_1 : \quad \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \mathtt{Null})$$
$$A_i \longrightarrow A_{i+1} : \quad \mathcal{H}_{K_{A_iS}}(A_i, A_{i+1}, N_{A_i}, X_i)$$

Submission Stage

$$A_n \longrightarrow S : \quad \mathcal{H}_{K_{A_nS}}(A_n, S, N_{A_n}, X_n)$$

Distribution Stage

$$\begin{aligned} S \longrightarrow A_n : \quad & \{K_n, S, N_{A_n}\}_{K_{A_nS}}, \{K_{n-1}, A_{n-1}, N_{A_n}\}_{K_{A_nS}}, \\ & \{K_{n-1}, A_n, N_{A_{n-1}}\}_{K_{A_{n-1}S}}, \{K_{n-2}, A_{n-2}, N_{A_{n-1}}\}_{K_{A_{n-1}S}}, \\ & \ldots \\ & \{K_1, A_2, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_0, N_{A_1}\}_{K_{A_1S}}, \\ & \{K_0, A_1, N_{A_0}\}_{K_{A_0S}} \\ A_i \longrightarrow A_{i-1} : \quad & \{K_{i-1}, A_i, N_{A_{i-1}}\}_{K_{A_{i-1}S}}, \{K_{i-2}, A_{i-2}, N_{A_{i-1}}\}_{K_{A_{i-1}S}}, \ldots \\ A_1 \longrightarrow A_0 : \quad & \{K_0, A_1, N_{A_0}\}_{K_{A_0S}} \end{aligned}$$

where $\mathtt{Null}$ is a special name, and X_i is the message from A_{i-1} to A_i.

3.2 Authentication of the RA Protocol

To represent authentication, *declaration processes* will be inserted into a protocol description [2,9]. For instance, the implementation, SYS^{RA}_{imp}, of the RA protocol below contains a declaration process $\overline{acc}\,x.\mathbf{0}$ for authentication.

$$\begin{aligned} \mathbb{O}_a(x_1, x_2) \triangleq & \overline{a1}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1, x_2, \mathtt{N}[\mathtt{Null}], \mathtt{Null}).a2(x).case\ x\ of \{y_1, y_2, y_3\}_{\mathtt{lk}[x_1,S]}. \\ & [y_3 = \mathtt{N}[\mathtt{Null}]]\,\overline{\mathbf{acc}}\,\mathbf{x}.\mathbf{0} \\ \mathbb{R}_a(x_1, x_2) \triangleq & (b1(x).let\ (y_1, y_2, y_3, y_4, y_5) = x\ in\ [y_2 = x_1] \\ & \overline{b2}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1, \mathtt{A}[x_1], \mathtt{N}[y_3], x).(\mathbb{R}(\mathtt{A}[x_1], x_1) \\ & + \overline{b3}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1, S, \mathtt{N}[y_3], x).\mathbf{0})); (b4(x).let\ (z_1, z_2, z_3) = x\ in \\ & case\ z_1\ of\ \{z_4, z_5, z_6\}_{\mathtt{lk}[x_1,S]}\ in\ [z_5 = \mathtt{A}[x_1]]\,[z_6 = \mathtt{N}[y_3]] \\ & case\ z_2\ of\ \{z_7, z_8, z_9\}_{\mathtt{lk}[x_1,S]}\ in\ [z_8 = x_2]\,[z_9 = \mathtt{N}[y_3]]\,\overline{b5}z_3.\mathbf{0}) \\ S \triangleq & s1(x).\overline{s2}\,(\mathscr{F}(x)).\mathbf{0} \\ SYS^{RA}_{imp} \triangleq & \mathbb{O}_a(\mathtt{A}[\mathtt{Null}], \mathtt{A}[\mathtt{A}[\mathtt{Null}]]) \| \mathbb{R}_a(\mathtt{A}[\mathtt{A}[\mathtt{Null}]], \mathtt{A}[\mathtt{Null}]) \| S \end{aligned}$$

In the description, we use a group of nested binders to describe unbounded number of fresh names. For instances, by $\mathtt{N[Null]}, \mathtt{N[N[Null]]}, \ldots$ we describe fresh nonces $N_{A_0}, N_{A_1}, \ldots$.

$\mathscr{F} : \mathcal{M} \rightarrow \mathcal{M}$ is an iterative procedure that generates an arbitrarily long message. We name this kind of messages *recursive messages*.

$\mathscr{F}$ is defined as follows:

$$
\begin{aligned}
\mathscr{F}(x) = {} & \mathtt{let}\ (y_1, y_2, y_3, y_4, y_5) = x; \\
& \mathtt{let}\ t = \epsilon; \\
& \mathtt{while}\ (y_1 = \mathcal{H}(y_2, y_3, y_4, y_5, \mathtt{lk}[y_2, S])\ \ \&\&\ \ y_5 ! = \mathtt{Null}) \\
& \qquad \mathtt{let}\ (z_1, z_2, z_3, z_4, z_5) = y_5; \\
& \qquad \mathtt{if}\ (z_1 = \mathcal{H}(z_2, z_3, z_4, z_5, \mathtt{lk}[z_2, S]) \&\& z_3 == y_2) \\
& \qquad \mathtt{then}\ t = (t, \{\mathtt{k}[y_4], y_3, y_4\}, \{\mathtt{k}[y_3], z_2, z_4\}); \\
& \qquad \mathtt{else\ raise\ error} \\
& \qquad \mathtt{endif} \\
& \qquad (y_1, y_2, y_3, y_4, y_5) := (z_1, z_2, z_3, z_4, z_5); \\
& \mathtt{endwhile} \\
& t := (t, \{\mathtt{k}[y_4], y_3, y_4\}); \\
& \mathtt{return}\ t;
\end{aligned}
$$

The specification for the authentication, SYS^{RA}_{spe}, is a process that replaces x in $\overline{acc}\, x.\mathbf{0}$ with $\{\mathtt{k[Null]}, \mathtt{A[A[Null]]}, \mathtt{N[Null]}\}_{\mathtt{lk[A[Null]},S]}$.

Authentication between the originator and its recipient is defined by

$$SYS^{RA}_{imp} \sim_t SYS^{RA}_{spe}$$

The implementation and the specification may fail to generate the same traces after certain message comparisons. The specification will guarantee that the message received and validated by one principal should be the same as the message sent by other principal, while these messages would be different in the implementation due to the ill-design of a protocol. Hence, we can explicitly check the equality of the two messages in traces generated by the implementation [7,9], which is another way to encode the correspondence assertion.

Definition 7 (Action terms[3]). *Let α and β be actions, with $f_v(\alpha) \subseteq f_v(\beta)$, and let s be a trace. We use $s \models \alpha \hookleftarrow \beta$ to represent that for each ground substitution ρ, if $\beta\rho$ occurs in s, then there exists one $\alpha\rho$ in s before $\beta\rho$. A configuration satisfies $\alpha \hookleftarrow \beta$, denoted by $\langle s, P\rangle \models \alpha \hookleftarrow \beta$, if each trace s' generated from $\langle s, P\rangle$ satisfies $s' \models \alpha \hookleftarrow \beta$.*

Characterization 1. *[Authentication for the RA protocol] Given the formal description of the RA protocol, the recipient is correctly authenticated to the originator, if $\langle \epsilon, SYS^{RA}_{imp}\rangle \models \overline{b5}\, x \hookleftarrow \overline{acc}\, x$.*

4 Model Checking by the Pushdown System

4.1 Encoding as Pushdown Model

To analyze recursive protocols with a pushdown system, the restrictions for a process are, (i) a system is restricted to contain at most one recursive process; (ii) the expression that defines the recursive process is sequential.

When analyzing protocols in bounded sessions, fresh messages that processes generate are bounded. We can fix a set of distinguished symbols to describe them [7]. However, for the analysis of recursive protocols, fresh messages can be unbounded. We represent an unbounded number of fresh messages by nested binders. With the restrictions of a single recursive process, the same context (stack content) will not be repeated; thus freshness will be guaranteed.

Definition 8 (Pushdown system). *A pushdown system $\mathcal{P} = (Q, \Gamma, \Delta, c_0)$ is a quadruple, where Q contains the control locations, and Γ is the stack alphabet. A configuration of $\mathcal{P}$ is a pair (q, ω) where $q \in Q$ and $\omega \in \Gamma^*$. The set of all configurations is denoted by $\mathtt{conf}(\mathcal{P})$. With $\mathcal{P}$ we associated the unique transition system $\mathcal{I}_{\mathcal{P}} = (\mathtt{conf}(\mathcal{P}), \Rightarrow, c_0)$, whose initial configuration is c_0.*

Δ is a finite subset of $(Q \times \Gamma) \times (Q \times \Gamma^*)$. If $((q, \gamma), (q', \omega)) \in \Delta$, we also write $\langle q, \gamma\rangle \hookrightarrow \langle q', \omega\rangle$. For each transition relation, if $\langle q, \gamma\rangle \hookrightarrow \langle q', \omega\rangle$, then $\langle q, \gamma\omega'\rangle \Rightarrow \langle q', \omega\omega'\rangle$ for all $\omega' \in \Gamma^*$.

We define a set of messages used for the pushdown system as follows,

Definition 9 (Messages in the pushdown system)

$$pr ::= x \mid \top \mid \mathtt{m}[\,] \mid \mathtt{m}[pr, \ldots, pr]$$
$$M, N, L ::= pr \mid (M, N) \mid \{M\}_L \mid \mathcal{H}(M)$$

Two new messages are introduced. $\top$ is a special name, substituting a variable that can be substituted to an unbounded number of names. $\mathtt{m}[\,]$ is a *binder marker*, representing nested binders, together with the stack depth. For instance, $\mathtt{A}[\mathtt{A}[\mathtt{Null}]]$ is represented by $\mathtt{A}[\,]$, with two stack elements in the stack.

Definition 10 (compaction). *Given a parametric trace $\hat{s}$, a compaction $\hat{tr}$ is a parametric trace by cutting off redundant actions with the same labels in $\hat{s}$.*

We represent the parametric model with at most one sequential recursive process by the pushdown system as follows,

- control locations are pairs $(R, \hat{tr})$, where R is a finite set of recursive messages, and $\hat{tr}$ is a compaction.
- stack alphabet only contains a symbol $\star$.
- initial configuration is $\langle(\emptyset, \epsilon), \varepsilon\rangle$, where ϵ represents an empty parametric trace, and ε represents an empty stack.
- Δ is defined by two sets of translations, the translations for the parametric rules, and the translations for the refinement step.

An occurrence of **0** in the last sequence process of a recursive process means a return point of the current process. We will replace it to a distinguished marker, **Nil**, when encoding a parametric system to the pushdown system.

The key encodings of the parametric transitions are as follows, in which $\hat{tr}$ and $\hat{tr}'$ are compactions of $\hat{s}$ and $\hat{s}'$, respectively.

1. For parametric transition rules except $PIND$ rules, $\langle(R, \hat{tr}), \omega\rangle \hookrightarrow \langle(R, \hat{tr}'), \omega\rangle$ if $\langle\hat{s}, P\rangle \longrightarrow_p \langle\hat{s}', P'\rangle$.

```
Solution 1 (state 415)
states: 416  rewrites: 67367 in 7689676981ms cpu (824ms real) (0
    rewrites/second)
ML1 --> (((k[Mk],name(1)),px(32))lk[px(31),name(1)],{(Mk,px(33)),px(32)}lk[px(
    31),name(1)]),{(Mk,px(31)),px(34)}lk[px(33),name(1)]
TR1 --> < a(1),o,H(lk[MA,name(1)],((MA,A[MA]),MN),null) > . < b(1),i,H(lk[MA,
    name(1)],((MA,A[MA]),MN),null) > . < b(3),o,H(lk[A[MA],name(1)],((A[MA],
    name(1)),N[MN]),H(lk[MA,name(1)],((MA,A[MA]),MN),null)) > . < s(1),i,H(lk[
    A[MA],name(1)],((A[MA],name(1)),N[MN]),H(lk[MA,name(1)],((MA,A[MA]),MN),
    null)) > . < s(2),o,{(k[Mk],name(1)),N[MN]}lk[A[MA],name(1)],{(Mk,MA),N[
    MN]}lk[A[MA],name(1)]),{(Mk,A[MA]),MN}lk[MA,name(1)] > . < a(2),i,{(Mk,A[
    MA]),MN}lk[MA,name(1)] > . < acc,o,{(Mk,A[MA]),MN}lk[MA,name(1)] >
STACK --> empty
```

Fig. 2. Snapshot of Maude Result for the Recursive Authentication Protocol

2. For $PIND$ rule, when $\mathbb{R}$ is firstly met, $\langle (R, \hat{tr}), \omega \rangle \hookrightarrow \langle (R, \hat{tr}'), \omega \rangle$ if $\langle \hat{s}, P \rangle \longrightarrow_p \langle \hat{s}', P' \rangle$, where $\mathbb{R}(\tilde{pr}) \triangleq P$; Otherwise $\langle (R, \hat{tr}), \omega \rangle \hookrightarrow \langle (R, \hat{tr}), \star\omega \rangle$.
3. $\langle (R, \hat{tr}), \gamma \rangle \hookrightarrow \langle (R, \hat{tr}), \varepsilon \rangle$ if $\langle \hat{s}, \mathbf{Nil} \rangle$ is met.

In the refinement step, we need to satisfy *rigid messages* by unifications [10,9]. A rigid message is the pattern of a requirement of an input action that can be satisfied by messages generated only by legitimate principals. We distinguish two kinds of rigid messages, *context-insensitive*, and *context-sensitive*.

Definition 11 (Context-sensitive/insensitive rigid messages). *Context-sensitive rigid messages are rigid messages that contain binder markers, while context-insensitive rigid messages do not contain any binder markers.*

Intuitively, a context-sensitive rigid message has an bounded number of candidate messages within the current context to unify with, while a context-insensitive one has an unbounded number of candidate messages to unify with.

The transition relations for the refinement step in Δ are defined as follows.

4. $\langle (R, \hat{tr}), \omega \rangle \hookrightarrow \langle (R, \hat{tr}\hat{\rho}), \omega \rangle$, if N is context-sensitive and $\hat{\rho}$-unifiable in $R \cup el(\hat{s}_1)$.
5. $\langle (R, \hat{tr}), \omega \rangle \hookrightarrow \langle (R \cup N', \hat{tr}\hat{\rho}'), \omega \rangle$, if N is context-insensitive and $\hat{\rho}$-unifiable to N' in $el(\hat{s}_1)$, and $\hat{\rho}'$ is the substitution that replaces different messages in N and N' with $\top$.

4.2 Implementing in Maude

We implemented the pushdown system above by Maude [14]. It describes model generating rules by rewriting, instead of constructing directly. The reachability problem can be checked at the same time while a model is being generated. We tested the RA protocol by our Maude implementation. A counterexample is automatically detected. The result snapshot is in Fig. 2, in which `MA`, `MN`, and `Mk` are binder markers. `name(1)` is the server name S. It describes attacks showed in Fig. 3, which actually represents infinitely many attacks. An intruder intercepts the message sent by S, splits it, and sends the parted message to A_0. The minimal one is,

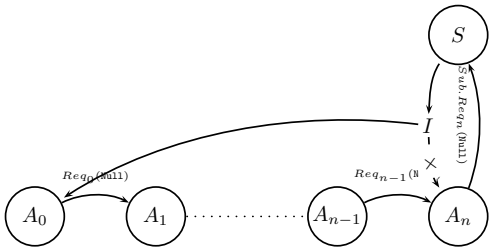

Fig. 3. The Attack of the RA Protocol

$$
\begin{aligned}
A_0 &\longrightarrow A_1 : && \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null}) \\
A_1 &\longrightarrow S : && \mathcal{H}_{K_{A_1S}}(A_1, S, N_{A_1}, \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null})) \\
S &\longrightarrow I(A_1) : && \{K_1, S, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_0, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_1, N_{A_0}\}_{K_{A_0S}} \\
I(A_1) &\longrightarrow A_0 : && \{K_0, A_1, N_{A_0}\}_{K_{A_0S}}
\end{aligned}
$$

This result obstructs that: (1) further update of the session key of A_0 is disabled, and (2) traceability of the session key of A_0 is violated, which are frequently required in the real-world security.

The implementation contains about 400 lines for the general structures and functions, and 32 lines for the protocol description. The test was performed on a Pentium M 1.4 GHz, 1.5 G memory PC. The flaw is detected at the last step.

protocols	states	times(ms)	flaws
recursive authentication protocol	416	824	detected

The reason of attacks is that S sends the message without any protections. One modification is that S protects the message it sends iteratively with long-term symmetric keys shared with principals. In the two-principal case,

$$
\begin{aligned}
A_0 &\longrightarrow A_1 : && \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null}) \\
A_1 &\longrightarrow S : && \mathcal{H}_{K_{A_1S}}(A_1, S, N_{A_1}, \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null})) \\
S &\longrightarrow A_1 : && \{\{K_1, A_2, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_0, N_{A_1}\}_{K_{A_1S}}, \\
& && \{K_0, A_1, N_{A_0}\}_{K_{A_0S}}\}_{K_{A_1S}} \\
A_1 &\longrightarrow A_0 : && \{K_0, A_1, N_{A_0}\}_{K_{A_0S}}
\end{aligned}
$$

The fixed protocol is checked secure by the same Maude implementation.

protocols	states	times(ms)	flaws
fixed recursive authentication protocol	416	1,068	secure

5 Related Work

G. Lowe proposed a taxonomy that elucidates four levels of authentication [15]. Let us suppose that in a session of a protocol, a sender A communicates with a receiver B.

- *Aliveness* of B guarantees that B attended the protocol.
- *Weak agreement* of B guarantees that B attended the protocol with A.
- *Non-injective agreement* of B guarantees that B attended the protocol with A, and two principals agreed on a set of messages $\mathcal{H}$.
- *Injective agreement* of B guarantees non-injective agreement of B, and that A corresponds to a unique run of B in the session.

Each level subsumes the previous one. This paper, together with other researches [12,3], took non-injective agreement as the standard authentication, which can be specified by the correspondence assertion.

Paulson took a weak form of key distribution property, and used Isabelle/HOL to prove that the correctness of the RA protocol with bounded number of principals [6]. Bella pointed out that non-injective agreement authentication and the weak form of key distribution "might be equivalent" [4]. However, we showed in this paper that the weak form of key distribution does not hold non-injective agreement, specified by the correspondence assertion.

Bryans and Schneider adopted CSP to describe behaviors of the RA protocol with the same assumption as Paulson's. They considered the correspondence assertion between the server and the last principal who submitted the request, and used PVS to prove the correctness of the authentication for the RA protocol [16].

Basin et al. proposed an on-the-fly model checking method (OFMC) [17] for security protocol analysis. In their work, an intruder's messages are instantiated only when necessary, known as *lazy intruder*. Their research is similar to our work in analyzing authentication in bounded sessions without binders.

A *tree transducer-based* model was proposed for recursive protocols by Küsters, et al. [18]. The rules in this model are assumed to have linear left-hand sides, so no equality tests can be performed. Truderung generalized the limitation, and proposed a *selecting theory* for recursive protocols [19]. Both of the two works focused on the secrecy property of the RA protocol. Recently, Küsters and Truderung considered the arithmetic encryption algorithm for the RA protocol, detected the known attack [20] automatically [21]. Since we assume a perfect cryptography, this attack is out of our methodology.

6 Conclusion

This paper presented the pushdown model checking of authentication of the RA protocol. It extended our previous work [7], allowing to analyze protocols with at most one recursive procedure. Our Maude implementation successfully detected a previously unreported attack that violates authentication in the sense of corresponding assertion of the RA protocol automatically. This result shows the effect of the subtle difference among security definitions.

Acknowledgements. The authors thank Prof. Kazuhiro Ogata for fruitful discussions. This research is supported by the 21st Century COE "Verifiable and Evolvable e-Society" of JAIST, funded by Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

1. Woo, T.Y., Lam, S.S.: A Semantic Model for Authentication Protocols. In: Proceedings of the S&P 1993, pp. 178–194. IEEE Computer Society Press, Los Alamitos (1993)
2. Abadi, M., Gordon, A.D.: A Calculus for Cryptographic Protocols: The Spi Calculus. In: Proceedings of the CCS 1997, pp. 36–47. ACM Press, New York (1997)
3. Boreale, M.: Symbolic Trace Analysis of Cryptographic Protocols. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 667–681. Springer, Heidelberg (2001)
4. Bella, G.: Inductive Verification of Cryptographic Protocols. PhD thesis, University of Cambridge (2000)
5. Bellare, M., Rogaway, P.: Provably Secure Session Key Distribution: The Three Party Case. In: Proceedings of the STOC 1995, pp. 57–66. ACM Press, New York (1995)
6. Paulson, L.C.: Mechanized Proofs for a Recursive Authentication Protocol. In: Proceedings of the CSFW 1997, pp. 84–95. IEEE Computer Society Press, Los Alamitos (1997)
7. Li, G., Ogawa, M.: On-the-Fly Model Checking of Security Protocols and Its Implementation by Maude. IPSJ Transactions on Programming 48, 50–75 (2007)
8. Li, G., Ogawa, M.: On-the-Fly Model Checking of Fair Non-repudiation Protocols. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 511–522. Springer, Heidelberg (2007)
9. Li, G.: On-the-Fly Model Checking of Security Protocols. PhD thesis, Japan Advanced Institute of Science and Technology (2008)
10. Li, G., Ogawa, M.: Authentication Revisited: Flaw or Not, the Recursive Authentication Protocol. Technical Report IS-RR-2008-002, Japan Advanced Institute of Science and Technology (2008)
11. Sangiorgi, D., Walker, D.: The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2003)
12. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-key Using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
13. Bull, J.A., Otway, D.J.: The Authentication Protocol. Technical report, Defence Research Agency, UK (1997)
14. Clavel, M., Durán, F., Eker, S., Lincolnand, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual (Version 2.2) (2005)
15. Lowe, G.: A Hierarchy of Authentication Specifications. In: Proceedings of the CSFW 1997, pp. 31–43. IEEE Computer Society Press, Los Alamitos (1997)
16. Bryans, J., Schneider, S.: CSP, PVS and a Recursive Authentication Protocol. In: Proceedings of the DIMACS FVSP 1997 (1997)
17. Basin, D.A., Mödersheim, S., Viganò, L.: OFMC: A Symbolic Model Checker for Security Protocols. International Journal of Information Security 4(3), 181–208 (2005)
18. Küsters, R., Wilke, T.: Automata-based Analysis of Recursive Cryptographic Protocols. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 382–393. Springer, Heidelberg (2004)
19. Truderung, T.: Selecting Theories and Recursive Protocols. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 217–232. Springer, Heidelberg (2005)
20. Ryan, P., Schneider, S.: An Attack on a Recursive Authentication Protocol: A Cautionary Tale. Information Processing Letters 65, 7–10 (1998)
21. Küsters, R., Truderung, T.: On the Automatic Analysis of Recursive Security Protocols with XOR. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 646–657. Springer, Heidelberg (2007)