Athena: a New Efficient Automatic Checker for Security Protocol Analysis [*]

Dawn Xiaodong Song
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
skyxd@cs.cmu.edu

Abstract

We propose an efficient automatic checking algorithm, Athena, for analyzing security protocols. Athena incorporates a logic that can express security properties including authentication, secrecy and properties related to electronic commerce. We have developed an automatic procedure for evaluating well-formed formulae in this logic. For a well-formed formula, if the evaluation procedure terminates, it will generate a counterexample if the formula is false, or provide a proof if the formula is true. Even when the procedure does not terminate when we allow any arbitrary configurations of the protocol execution, (for example, any number of initiators and responders), termination could be forced by bounding the number of concurrent protocol runs and the length of messages, as is done in most existing model checkers.

Athena also exploits several state space reduction techniques. It is based on an extension of the recently proposed Strand Space Model [25] which captures exact causal relation information. Together with backward search and other techniques, Athena naturally avoids the state space explosion problem commonly caused by asynchronous composition and symmetry redundancy. Athena also has the advantage that it can easily incorporate results from theorem proving through unreachability theorems. By using the unreachability theorems, it can prune the state space at an early stage, hence, reduce the state space explored and increase the likely-hood of termination. As shown in our experiments, these techniques dramatically reduce the state space that needs to be explored.

[*]This research is sponsored by the National Science Foundation under Grant No. CCR-9803774. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Government.

1. Introduction

A security protocol is a communication protocol that uses cryptography to achieve goals such as authentication and key distribution. Because of the subtlety of security protocols, experience has shown that the protocols can be flawed even when designed carefully. Thus, it is necessary to develop rigorous ways to analyze these protocols.

Many researchers have worked on applying formal techniques to the analysis of security protocols. They have developed logics of knowledge and belief such as BAN logic [2] and GNY logic [7]; semi-automatic and fully automatic tools such as the NRL Analyzer [14], the Interrogator Model [16], FDR [12], Murφ [17], Brutus [5], and Revere [9]; and theorem provers such as Isabelle [19]. Automatic checkers have the practical advantage that they are easy to use and do not need the assistance of experienced users. Unfortunately, current automatic checkers suffer from the state space explosion problem, mainly due to asynchronous composition and symmetry redundancy. Most automatic checkers are also limited to checking the properties of a security protocol under certain configurations of the protocol execution, e.g. with two initiators and two responders.

Thayer, Herzog and Guttman recently proposed the Strand Space Model (*SSM*) and demonstrated how to use *SSM* to prove certain security properties manually, for example authentication and secrecy [25]. *SSM* has the advantage that it contains the exact causal relation information which makes proofs concise. Inspired by their work, we have developed a new algorithm, *Athena*, for analyzing security protocols automatically.

We have designed a logic based on SSM that can express formally various security properties including authentication, secrecy and properties related to electronic commerce. We have also developed an automatic procedure for evaluating well-formed formulae in this logic. We provide a way of formally reduce the infinite-state-space problem into a finite-state-space problem which can be verified using model checking. Hence, for a well-formed formula, if the evaluation procedure terminates, then it will generate a counterexample if the formula is false, or provide a proof if the formula is true. Although the evaluation procedure is not guaranteed to terminate, experience shows that it does terminate for many useful protocols. In those cases when the procedure does not terminate for arbitrary configurations of the protocol execution, termination can always be forced by bounding the number of concurrent protocol runs and the length of messages. This is similar to the bounds in current model checkers such as FDR, Murφ and *Brutus*.

Athena also exploits several state space reduction techniques. First, the state transition is not asynchronously composed of independent process transitions, hence, avoid the state space explosion caused by asynchronous composition. Second, the state structures and state transitions capture exact causal relations, hence, achieve compact and efficient state representations. Third, Athena takes advantage of symbolic state transitions instead of explicit state search, by allowing a state to contain free variables. A state $s(\vec{x})$ with free term variables, $\vec{x}$, represents a class of variable-free states, $\{[\vec{\sigma}/\vec{x}]s(\vec{x})\}$, where $\vec{\sigma}$ is a substitution of term values. A state transition between two states which contain free variables represents a set of state transitions between two variable-free states. Thus, Athena can represent states and state transitions much more efficiently. As a special case of this, it naturally avoids the symmetry redundancy problem. Finally, Athena uses backward search instead of forward search. With forward search, all the participating principals have to be pre-stated. Our approach starts with a simple initial strand and then add new strands only when necessary according to exact causal relations. As demonstrated in our experiments, these techniques greatly reduce the state space that needs to be explored comparing with other current approaches.

Athena also has the advantage that it can easily incorporate results from theorem proving through unreachability theorems. By using the unreachability theorems, we can prune the state space at an early stage, hence, reduce the state space explored even further and increase the likelyhood of termination.

The paper is organized as follows. We first review some basic notions and properties of strand spaces (section 2). We then introduce a logic to reason about strand spaces and show how to use this logic to specify security properties (section 3), and explain the automatic evaluation procedure (section 4). Next, we discuss the advantages of our approach and compare it with other approaches (section 5). Finally, we conclude in section 6. Proof sketches of most propositions in this paper are omitted and can be found in [23].

2. Background

This section is a review of concepts developed by Thayer, Herzog and Guttman [25]. First, we explain the notion of *terms* that are used to represent the messages in the protocols. Then, we explain the notion of *strands, strand spaces* and *bundles*, and show how to represent protocols using strands. Finally, we give the formal description of the *penetrator model*.

2.1. Message Terms

The set of atomic terms is the union of a set of "*Text*" terms *T* and a set of "*Key*" terms *K*, where

- Text terms *T* contain several different types of terms, such as *Principal-names, Nonces,* or *Bank-account-number*.
- Key terms *K* contains a set of keys disjoint from *T*. In asymmetric crypto systems, K^{-1} represents *K*'s opposite member in a public-private key pair. In a symmetric key system, $K^{-1} = K$.

The set of *terms* (A) is defined inductively as follows:

- If *m* is a Text term or a Key term, then *m* is a term.
- If *m* is a term, k is a Key term, then $\{m\}_k$ is a term. (This represents encryption.)
- If m_1 and m_2 are terms, then $m_1 \cdot m_2$ is a term. (This represents concatenation.)

We use the *free encryption* assumption, where

$$\{m\}_k = \{m'\}_{k'} \Leftrightarrow m = m' \wedge k = k'.$$

Thayer, Herzog and Guttman defined the *subterm relation* $\sqsubseteq$: a term a_1 is a *subterm* of term a_2 if a_1 appears in a_2; We define the *interm relation* $\Subset$, such that a_1 is a *interm* of a_2 if a_1 can be extracted from a_2 without the application of the decryption operation. The formal definition of the two relations are as follows.

- *subterm relation* $\sqsubseteq$
  - $a \sqsubseteq t$ *for* $t \in T$ *iff* $a = t$;
  - $a \sqsubseteq k$ *for* $k \in K$ *iff* $a = k$;
  - $a \sqsubseteq \{g\}_k$ *iff* $a \sqsubseteq g \vee a = \{g\}_k$;
  - $a \sqsubseteq g \cdot h$ *iff* $a \sqsubseteq g \vee a \sqsubseteq h \vee a = g \cdot h$.
- *interm relation* $\Subset$
  - $a \Subset t$ *for* $t \in T$ *iff* $a = t$;
  - $a \Subset k$ *for* $k \in K$ *iff* $a = k$;
  - $a \Subset \{g\}_k$ *iff* $a = \{g\}_k$;
  - $a \Subset g \cdot h$ *iff* $a \Subset g \vee a \Subset h$.

2.2. SSM : Strands, Strand spaces and Bundles

The notions in this subsection are mainly from the paper [25]. We extend them slightly to make them applicable to electronic commerce protocols.

Actions. The set of actions *Act* that principals can take during an execution of a protocol include extermal actions such as *send* and *receive*, and user-defined internal actions such as *debit, credit,* etc.. In the rest of the paper, we will only use *send* and *receive* for simplicity.

Events. An *event* is a pair $\langle$*action, argu*$\rangle$, where *action* is in *Act*, and *argu* is in (A) and is the argument of the action. For simplicity, we denote $\langle$*send, a*$\rangle$ and $\langle$*receive, a*$\rangle$ respectively as *signed terms* $\langle +a \rangle$ and$\langle -a \rangle$. We represent the set of finite sequences of signed terms as $(\pm A)^*$.

Strands and Strand Spaces. A *protocol* defines the sequence of events for each role of the participant. A *strand* represents a sequence of actions of an instance of a role.

A strand space is a set Σ with a trace mapping tr: $\Sigma \to (\pm A)^*$.

1. A *node* is a pair $\langle s, i \rangle$, with $s \in \Sigma$ and *i* an integer satisfying $1 \leq i \leq$ length(tr(s)). We say $n = \langle s, i \rangle$ belongs to the strand *s*, denoted as $n \in s$. Clearly, every node belongs to a unique strand. The set of nodes is denoted by N.

2. If $n = \langle s, i \rangle \in N$, then index$(n) = i$ and strand$(n) = s$. If $(tr(s))_i = \langle \sigma a \rangle$, where σ is one of the symbols $+, -$, then term$(n) = a$.

3. If $n_1, n_2 \in N$, then $n_1 \to n_2$ means that term$(n_1) = +a$ and term$(n_2) = -a$. This represents that n_1 sends a message *a* and n_2 receives the message.

4. If $n_1, n_2 \in N$, then $n_1 \Rightarrow n_2$ means that n_1, n_2 occur in the same strand with index(n_2) = index$(n_1) + 1$. This represents an event n_1 followed immediately by n_2 in the same strand.

5. A term *t originates* from a node $n \in N$ *iff sign(n)*$= +$*; t* $\sqsubseteq$ *term(n)*; and whenever n' precedes *n* on the same strand, *t* $\not\sqsubseteq$ *term(*n'*)*.

6. A term *t uniquely-originates* from node *n iff t originates* on a unique $n \in N$. Nonces and other freshly generated terms are usually uniquely-originated.

We will also use N to refer to the directed graph (N, E) whose vertices are nodes and $E = (\to \cup \Rightarrow)$ is the set of edges that combines both types of relations $n_1 \to n_2$ and $n_1 \Rightarrow n_2$.

Bundles. A bundle represents the protocol execution under some configuration.

A bundle $C = (N_C, E)$ is a subgraph of N, where $E \subseteq (\to \cup \Rightarrow)$ is the set of the edges and $N_C \subseteq N$ is the set of nodes incident with the edges in E, and the following properties hold:

- C is non-empty and finite;
- If $n_1 \in C$ and *sign*$(n_1) = -$, then there is a unique n_2 such that $n_2 \to n_1 \in C$;
- If $n_1 \in C$ and $n_2 \Rightarrow n_1$, then $n_2 \Rightarrow n_1 \in C$;
- C is acyclic.

We say a strand $s \in C$ if for every node $n \in s$, $n \in C$.

Causal Precedence. Let *S* be a strand space, nodes n_1,n_2 $\in$ S. Define $n_1 \preceq_S n_2$ *iff* there is a sequence of zero or more edges of type $\to$ and $\Rightarrow$ leading from n_1 to n_2 in *S*. The relation $\preceq_S$ expresses a *causal precedence*.

Lemma 2.1. *Suppose C is a bundle, then* $\preceq_C$ *is a partial order, i.e. a reflexive, antisymmetric, transitive relation. Every non-empty subset of the nodes in* C *has* $\preceq_C$*-minimal members.*

The proof of this lemma can be found in [25].

2.3. Protocol specification using strands

A protocol usually contains several *roles*, such as *initiators, responders* and *servers*. The sequence of actions of each role is predefined by the protocol with a list of parameters, such as principal names and nonces. This can be specified as a *trace type*, denoted as *role[parameter list]*. A binding of the role and the parameter list gives an instance-trace of the role. A legal execution of a protocol forms a bundle, in which the strands of the legitimate principals are restricted to the predefined trace types, *role[parameter list]*. The strands of the legitimate principals are referred to as *regular strands*. The bundle also contains strands which are mapped to penetrator traces. These strands are referred to as *penetrator strands*. We explain them in more details in the next subsection. We now give an example of the Needham-Schroeder protocol [18] with the fix given by Gavin Lowe in [12]. We will refer to this protocol as *NSL* in the rest of the paper. Using the standard notation, the protocol is defined as follows:

1. $A \to B : \{N_a \cdot A\}_{K_B}$
2. $B \to A : \{N_a \cdot N_b \cdot B\}_{K_A}$
3. $A \to B : \{N_b\}_{K_B}$

There are two roles in this protocol: initiator and responder. The strands of the two roles are the following :

$$\begin{tikzcd}[row sep=small]
\underline{Resp[A,B,N_a,N_b]} & & \underline{Init[A,B,N_a,N_b]} \\
1:\langle -\{N_a \cdot A\}_{K_b} \rangle \arrow[d, Rightarrow] & & 1:\langle +\{N_a \cdot A\}_{K_b} \rangle \arrow[d, Rightarrow] \\
2:\langle +\{N_a \cdot N_b \cdot B\}_{K_a} \rangle \arrow[d, Rightarrow] & & 2:\langle -\{N_a \cdot N_b \cdot B\}_{K_a} \rangle \arrow[d, Rightarrow] \\
3:\langle -\{N_b\}_{K_b} \rangle & & 3:\langle +\{N_b\}_{K_b} \rangle
\end{tikzcd}$$

where the parameter list contains *A* and *B* as principal names, N_a and N_b as nonces. N_a *uniquely-originates* on the first node of the initiator's strand $Init[A, B, N_a, N_b]$. N_b *uniquely-originates* on the second node of the responder's strand $Resp[A, B, N_a, N_b]$. A responder's strand with the binding of $[A_0, B_0, N_{a0}, N_{b0}]$ is the following :

$$\begin{tikzcd}[row sep=small]
\underline{Resp[A_0,B_0,N_{a0},N_{b0}]} \\
\langle -\{N_{a0} \cdot A_0\}_{K_{b0}} \rangle \arrow[d, Rightarrow] \\
\langle +\{N_{a0} \cdot N_{b0} \cdot B_0\}_{K_{a0}} \rangle \arrow[d, Rightarrow] \\
\langle -\{N_{b0}\}_{K_{b0}} \rangle
\end{tikzcd}$$

2.4. The Penetrator Model

We use the same penetrator model as the one in [25]. The penetrator P has a set of initial knowledge *init-info(P)* which usually contains the principal names and the keys that are known initially to the penetrator, denoted as K_p. K_p usually contains all the public keys, all the private keys of the penetrator, and all the symmetric keys K_{px}, K_{xp} initially shared between the penetrator and principals playing by the protocol rules. It can also contain some keys to model known-key attacks.

A penetrator can intercept messages, generate messages that are computable from its initial knowledge and the messages it intercepts. These actions are modeled by a set of *penetrator strands*.

A penetrator strand is one of the following, where *g* and *h* are *terms*:

- M[t]. Atomic message: ⟨+t⟩ where t ∈ *init-info(P)* and t ∈ T.
- F[g]. Flushing: $\langle -g \rangle$.
- T[g]. Tee: $\langle -g, +g, +g \rangle$.
- C[g,h]. Concatenation: $\langle -g, -h, +g \cdot h \rangle$.
- R[g,h]. Separation into components: $\langle -g \cdot h, +g, +h \rangle$.
- K[k]. Key:$\langle +k \rangle$ where $k \in K_p$.
- E[k,h]. Encryption: $\langle -k, -h, +\{h\}_k \rangle$.
- D[k,h]. Decryption: $\langle -k^{-1}, -\{h\}_k, +h \rangle$.

The different types of the penetrator traces are called *penetrator roles*, which ranges over {M,F,T,C,S,K,E,D}. It is also possible to extend the set of penetrator traces to model some special ability of the penetrator if needed.

3. The Logic

We first introduce a logic to reason about strand spaces and bundles. Then we show how to use this logic for formal specification of various security properties.

3.1. Syntax

The syntax of the terms consists of *node constants* $(n, n_1, \ldots)$, *strand constants* $(s, s_1, \ldots)$, *bundle constant* $(c, c_1, \ldots)$, and *bundle variable* $(C, C_1, \ldots)$.

Propositional formulas are defined as follows:

- $n \in s$, $n \in c$, $n \in C$, $s \in c$, $s \in C$ are (atomic) propositional formulas;
- $\neg f_1$ and $f_1 \wedge f_2$ are propositional formulas if f_1 and f_2 are propositional formulas.

Finally, well-formed formulas (wffs) are:

- $f, \neg F_1, F_1 \wedge F_2$;
- $\forall C.f$, where f is a propositional formula, which doesn't contain any other free variable than C;

where f is a propositional formula, F_1 and F_2 are wffs, and C is a bundle variable.

Notice, that in a wff $\forall C.f$, f needs to be a propositional formula and cannot contain any other variables than C. We also use the obvious abbreviations:

$$\begin{array}{rclrcl} f_1 \vee f_2 & \equiv & \neg(\neg f_1 \wedge \neg f_2) & f_1 \Rightarrow f_2 & \equiv & \neg f_1 \vee f_2 \\ f_1 \Leftrightarrow f_2 & \equiv & f_1 \Rightarrow f_2 \wedge f_2 \Rightarrow f_1 & \exists C.f & \equiv & \neg \forall C.\neg f \end{array}$$

3.2. Semantics

Let the set of nodes be $\mathcal{N}$. For a given protocol, the set of the regular strands and the penetrator strands is $\mathcal{S}_p$; the execution traces of a protocol *p* form a set of bundles, denoted as $\mathcal{D}_p$. Thus, for a given protocol *p*, the model M is a pair $(\{\mathcal{N}, \mathcal{S}_p, \mathcal{D}_p\}, \mathcal{I})$, where $\mathcal{I}$ is the interpretation. The semantics of the logic is given as follows :

- $\mathcal{I}(n), \mathcal{I}(s), \mathcal{I}(c)$ are a node, a strand and a bundle in $\mathcal{N}, \mathcal{S}_p, \mathcal{D}_p$ respectively.

- $M \models n \in s$ *iff* $\mathcal{I}(n) \in \mathcal{I}(s)$. We can similarly define $M \models n \in c, M \models s \in c$ as well.

- If f is a propositional formula or a wff, then $M \models \neg f$ *iff* $M \not\models f$.

- If f_1 and f_2 are propositional formulas or wffs, then $M \models f_1 \wedge f_2$ *iff* $M \models f_1$ and $M \models f_2$.

- $M \models \forall C.f$ *iff* $M \models [C/c_0]f$ for any bundle c_0 in the model.

3.3. Specifying security properties in the logic

Our logic can specify a variety of security properties including electronic commerce properties. However, here we mainly focus on the authentication and secrecy properties. The properties are presented in a similar way as in the paper [25] except that we formalize the properties into the wffs in our logic.

Authentication

Gavin Lowe [13] proposed agreement properties for authentication protocols. A protocol guarantees a participant B (say, as the responder) *agreement* for certain binding $\vec{x}$ if each time a principal B completes a run of the protocol as a responder using $\vec{x}$, supposedly with A, then there is a unique run of the protocol with the principal A as initiator using $\vec{x}$, supposedly with B.

A weaker *non-injective agreement* does not ensure uniqueness, but requires only each time a principal B completes a run of the protocol as responder using $\vec{x}$, supposedly with A, then there exists a run of the protocol with the principal A as initiator using $\vec{x}$, supposedly with B.

The non-injective agreement property can be specified using the logic as:

$$\forall C.Resp(\vec{x}) \in C \Longrightarrow Init(\vec{x}) \in C,$$

where $Resp(\vec{x})$ and $Init(\vec{x})$ are the responder and the initiator strand with binding $\vec{x}$. For example, in the *NSL* protocol, the *non-injective agreement* property can be specified as

$$\forall C.Resp[A, B, N_a, N_b] \in C \Longrightarrow Init[A, B, N_a, N_b] \in C.$$

Because of the freshness of the nonces generated in the protocol run, usually the agreement property can be proved after the non-injective agreement property is proved, with the argument that there can't be two strands $Init(\vec{x}) \in C$ since the nonces in $Init(\vec{x})$ are uniquely originated from only one strand, i.e. in *NSL* protocol, N_a is *uniquely-originated* in the strand $Init[A, B, N_a, N_b]$.

Secrecy

A value *v* is secret in a strand space *S* if, for every bundle *C* that contains *S*, there does not exist a node $n \in C$ such that *term*$(n) = v$. For example, when *S* is a responder strand, we can specify the secrecy property as :

$$\neg\exists C.(Resp(\vec{x}) \in C \wedge node(+v) \in C)$$

4. The Model Checking Algorithm

This section introduces a model checking algorithm for wffs. We focus on the most interesting case: how to evaluate a wff of the form $\forall C.f$.

Lemma 4.1. *If $\mathcal{H}$ is an algorithm that decides the validity of any wff of the form $\forall C.f_1 \Rightarrow f_2$ in a model M in finite steps, where f_1 is a conjunction and f_2 is a disjunction of the atomic propositions, then there exists an algorithm to evaluate any formula of the form $\forall C.F$ in the model M in finite steps, for any propositional formula F.*

The proof sketch of this lemma can be found in the appendix.

Hence, it is sufficient to just explain the procedure $\mathcal{H}$. Due to the space limit, we only show the procedure $\mathcal{H}$ for two cases:

$$\mu_1 = \forall C.s_1 \in C \Rightarrow s_2 \in C, \text{ and}$$
$$\mu_2 = \exists C.s_1 \in C,$$

where s_1 is a strand constant, s_2 is a regular strand constant. These cases can be used for evaluating agreement properties and secrecy properties as mentioned in section 3. The other cases are simple extensions from them.

In the following subsections, we first introduce the notions and state structures used in the model checking algorithm. We then explain formally how we can reduce a problem in an infinite space to a problem of model checking which is restricted to a finite state space. We then describe in more details the model checking algorithm, in particular the *next state transition function*. We will also point out various techniques we exploit for state space reduction. Finally, we explain how we can use *unreachability theorems* to further reduce state space explored.

4.1. Goals and Goal-bindings

Intuition: We start with a strand space graph which only contains s_1, and then try all possibilities of adding nodes and strands to complete a bundle (an exhaustive search). To complete a bundle, the graph needs to be backward closed under $\Rightarrow$ and $\rightarrow$, and acyclic. To make the graph backward closed under $\rightarrow$, it means that any term that is received by a node must have been sent by another node in the same graph. Thus, we introduce a notion of a *goal* to represent the received terms, and a notion of a *goal-binding* to represent that a goal term is first sent by a node. We apply unification to search for all possible goal-bindings. When all goals are bound, all received terms are connected to their first senders. Hence, the graph will be backward closed under $\rightarrow$. For any node in the graph, we add all the nodes, that precede it in the same strand, to the graph. Thus, the graph is backward closed under $\Rightarrow$. Simple cycle-detection can check whether a graph is acyclic.

Goals. A *goal* is a pair (t, n), where $sign(n) = -$, $t \Subset Term(n)$ and t is not a concatenation of other two terms. The *goal-set* of a bundle C is the set of all the *goals* in C, denoted as $G(C)$.

Goal Binding. Suppose that C is a bundle, then a goal (t, n) is *bound* to node n' if n' is a $\preceq_C$-minimal member of the set

$$\psi = \{m \in C \mid t \Subset term(m),\ sign(m) = +, \text{ and } m \preceq n\}.$$

We say that $((t, n), n')$ is the *goal binding* for (t, n) and denote it as $n' \overset{t}{\twoheadrightarrow} n$. The node n' is called a *binder* of *(t,n)*. We will also write $n' \twoheadrightarrow n$ to denote $n' \overset{t}{\twoheadrightarrow} n$ for some t, or, formally, $\twoheadrightarrow = \bigcup_{t \in T} \overset{t}{\twoheadrightarrow}$.

Proposition 4.2. *Let* C *be a bundle. For any goal* $(t, n) \in G(C)$*, there exists a node* $n' \in C$ *where the goal* (t, n) *is bound.*

Proof sketch. Since *sign(n)* $= -$, and $n \in C$, there must exist a node n'', where $n'' = \langle -Term(n) \rangle$ and $n'' \in C$. So $t \Subset Term(n'')$. Therefore, the set ψ is not empty, where

$$\psi = \{m \in C \mid t \Subset Term(m) \text{ and } sign(m) = + \text{ and } m \preceq n\}.$$

As shown in Lemma 2.1, ψ has at least a $\preceq_C$-minimal member n'. Hence, (t, n) can be bound to n'. ☐

A bundle contains the information about the sequence of actions of each principal ("$\Rightarrow$") and the information about who sends messages to whom ("$\rightarrow$"). Because the penetrator can always intercept a message from a principal and forward it to another principal, the information about who sends messages to whom ("$\rightarrow$") is not important any more. Given a received term, it is not important who sends it but rather who sends it first. Therefore, only the goal-binding information ("$\twoheadrightarrow$") is necessary. This is the intuition why we introduce the relation "$\twoheadrightarrow$". In the state structure, we do not keep the information about the relation "$\rightarrow$" but only the goal-binding relation "$\twoheadrightarrow$".

4.2. State Structures

Semi-bundles. A semi-bundle $h = (N_h, E_{\Rightarrow})$ is a subgraph of N, where $E_{\Rightarrow} \subseteq \Rightarrow$ is the set of edges, N_h is the set of nodes incident to the edges in $E_{\Rightarrow}$, and the following properties hold:

- N_h is non-empty and finite;
- If $n_1 \in N_h$ and $n_2 \Rightarrow n_1$, then $n_2 \Rightarrow n_1 \in E_{\Rightarrow}$;
- h is acyclic.

Notice that the notion of a semi-bundle differs from the notion of a bundle in the aspect that a semi-bundle is backward closed under $\Rightarrow$, and not necessarily backward closed under $\rightarrow$.

States. A state is a tuple $\langle S, G, \twoheadrightarrow, \precsim \rangle$, where

- S is a semi-bundle containing strands of the principals;
- G is the set of unbound goals of S;
- $\twoheadrightarrow$ is the relation for the goal-bindings;
- $\precsim = (\twoheadrightarrow \cup \Rightarrow)^*$ — a reflexive and transitive closure of $\twoheadrightarrow$ and $\Rightarrow$ together.

  In other words, $n_1 \precsim n_2$ *iff* there exists a sequence of nodes $\{n'_i\}_{i=1}^k$, such that $n_1 = n'_1$, $n_2 = n'_k$, and for any $i \in [1, k-1]$, $n'_i \Rightarrow n'_{i+1}$ or $n'_i \twoheadrightarrow n'_{i+1}$. Hence, the relation $\precsim$ is a partial order on the nodes in S.

Mathematically, G and $\precsim$ can be computed from S and $\twoheadrightarrow$, but we keep them in the state structure for efficiency reasons.

If $l = \langle S_l, G_l, \twoheadrightarrow_l, \precsim_l \rangle$ is a state and c is a bundle, we say $l \in c$ *iff* $S_l \in c$.

Proposition 4.3. *If* G *is empty in a state* $\langle S, G, \twoheadrightarrow, \precsim \rangle$*, then there exists a bundle* C *such that* $S \in C$*, and* S *and* C *contain the same strands except that* C *might contain some more penetrator strands of type* R *or* T *than* S*.*

4.3. The Model Checking Algorithm

Bundle Sets. We define the *bundle set* of a state *l* to be the set of bundles that contain *l*, denoted as $\Psi(l) = \{c : bundle \mid l \in c\}$.

Next State Transition. If *l* is a state, the next state transition, $\mathcal{F} : state \rightarrow$ *a set of states*, is a function which maps *l* to the set of its next states, denoted as $L' = \mathcal{F}(l)$.

We say $\mathcal{F}$ is *complete-inclusive* if for any state l it satisfies the following properties :

1. L' is finite.
2. $\Psi(L') = \Psi(l)$, where $\Psi(L') = \bigcup_{l' \in L'} \Psi(l')$.

Unreachable states. If $\mathcal{F}(l)$ is empty for a state l, we say that l is *unreachable*. This means that there is not any bundle containing l, usually because l contains unsolvable goals.

As was mentioned earlier, we focus on explaining the algorithm for evaluating two simple wffs:

$$\mu_1 = \forall C. s_1 \in C \Rightarrow s_2 \in C, \text{ and}$$
$$\mu_2 = \exists C. s_1 \in C,$$

where s_1 is a strand constant, s_2 is a regular strand constant. The initial step of the model checking algorithm is to compute the initial state $l_0 = \langle S_0, G_0, \twoheadrightarrow_0, \precsim_0 \rangle$, where S_0 is the semi-bundle which only contains s_1, G_0 is the goal set of S_0, the relation $\twoheadrightarrow_0$ is empty, and $\precsim_0$ is the corresponding partial order on S_0. From the definition of $\Psi(l_0)$, we can derive:

$$\forall C. s_1 \in C \Rightarrow s_2 \in C \Leftrightarrow \forall c \in \Psi(l_0). s_2 \in c. \qquad (4.1)$$

$$\exists C. s_1 \in C \Leftrightarrow \exists c. c \in \Psi(l_0). \qquad (4.2)$$

The main procedure is a reachability analysis with a given next state transition $\mathcal{F}$. We will explain the next state transition $\mathcal{F}$ in the next subsection. *searchI* is the procedure for evaluating the formula μ_1, and *searchII* for μ_2. We give the pseudo code for *searchI* as follows. *searchII* is similar to *searchI*, and the pseudo code can be found in [23].

```
proc searchI(initialState){
    L = {initialState};
    while($\neg empty(L)$) do{
        $l = choose(L)$;
        $L = L \setminus l$;
        $L' = \mathcal{F}(l)$;
        if $\neg empty(L')$ then{
            for each $l' = \langle S', G', \twoheadrightarrow', \precsim' \rangle \in L'${
                if $s_2 \notin l'$ then{
                    if $empty(G')$ then return false;
                    else $L = add(l', L)$;
                }
            }
        }
    }
    return true;
}
```

choose(L) is a function which returns an element in the set L. We always pick a state which contains the least number of strands in the set L first.

We now give some intuition how we can reduce an infinite-state-space problem to a problem in a finite state space. First, from the Equation 4.1, we transform μ_1 into the equivalent formula,$\forall c \in \Psi(l_0). s_2 \in c.$, where $\Psi(l_0)$ is the set of bundles which contain l_0, as defined before. Then μ_1 is true, if we can find a set of states L, which satisfies the following property:

- $\forall l \in L. s_2 \in l$;
- $\Psi(l_0) = \Psi(L)$.

Thus, we reduce the problem in the infinite state space $\Psi(l_0)$ into a finite state space L. The reachability analysis is a way of searching for L. Once the search procedure terminates, the non-empty leaves of the search tree comprise the set L. We formalize these arguments into the correctness theorem below.

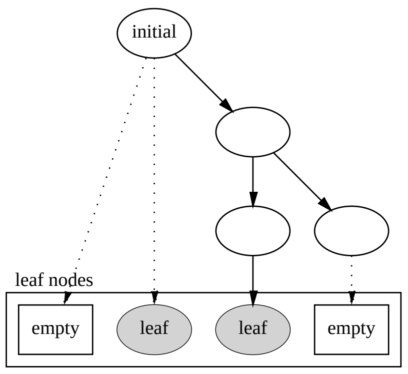
Theorem 4.4. *Let l_0 be an initial state, and $\mathcal{F}$ be a* complete-inclusive *next state transition function. If* searchI*(l_0) terminates and returns* true*, then the formula μ_1 is true; if it returns* false*, then μ_1 is false. The same holds for* searchII *and μ_2.*

Proof sketch. We only give the proof for *searchI* on μ_1. The proof for *searchII* on μ_2 can be derived similarly.

1. If *searchI*(l_0) returns false, then there exists a state $l = \langle S, G, \twoheadrightarrow, \precsim \rangle$ such that G is empty and $s_2 \notin l$. According to the Proposition 4.3, we can obtain at least a bundle which contains S and has the same regular strands as S. Therefore we find the counterexample C, $s_1 \in C$ and $s_2 \notin C$. Hence, μ_1 is false.

2. When *searchI*(l_0) returns true, the reachability analysis explores a state search tree T. An example of the state tree is in Figure 1. The root is the initial state l_0. The children of a tree node n is the next states $\mathcal{F}(n)$. Let L be the set of non-empty leaves of T. If L is empty, then there is no bundle which contains s_1, so μ_1 is true. If L is not empty, from the pseudo code we can see that for any $l \in L$, $s_2 \in l$. So for any $l \in \Psi(L)$, $s_2 \in l$. According to the complete-inclusive property of the next state transition $\mathcal{F}$, $\Psi(l_0) = \Psi(L)$. So $\forall l \in \Psi(l_0). s_2 \in l$. Finally, according to the Equation 4.1, we can see that μ_1 is true.

$\square$



A solid arrow represents the next state transition relation, whereas the dashed arrow stands for a sequence of next state transition relations.

Figure 1. State Search Tree.

4.4. The next state transition algorithms

We first give the intuition of the next state function $\mathcal{F}$. For a state $l = \langle S, G, \twoheadrightarrow, \precsim \rangle$, we pick a goal $g = (t, n) \in G$ from the unbound goal set. We then find all possible ways of binding g to a node. For each way of binding, we construct a new *next state*. The value of $\mathcal{F}(l)$ is the set of all such next states. If there is no possible way of binding g, then the state l is unreachable and $\mathcal{F}(l)$ returns an empty set.

Now we give a more detailed description of $\mathcal{F}$. To find all possible ways of binding a goal g, we use a unification procedure *U*. For a *goal term t*, $U(t)$ returns all possible positions that can bind t, and the corresponding most general substitutions, denoted as $\{([r, i], \gamma_{[r,i]})\}$. $[r, i]$ denotes a *position*, which represents the node of index *i* in a strand of role *r*. Role *r* can be any regular role or penetrator role. Each position $[r, i]$ can be bound to a strand s', either an existing strand in S or a new strand, with the substitution $\gamma_{[r,i]}$. Hence, a node $n' = \langle s', i \rangle$ will bind the goal *g*. For each of such n', we construct a new *next state* $l' = \langle S', G', \twoheadrightarrow', \precsim' \rangle$, where $\twoheadrightarrow'$ is the same as $\twoheadrightarrow$ with the added binding $n \overset{t}{\twoheadrightarrow} n'$. If s' is already in S, then S' is the same as S with the added n' and its preceding nodes in strand s', if they are not already in s' yet. If s' is a new strand, then S' is the same as S with the added strand s', which contains all the nodes up to n'. Then G' and $\precsim'$ can be updated appropriately. If there is a conflict in updating G' and $\precsim'$, e.g. there is a cycle in $\precsim'$, then l' is an unreachable state and will not be returned as a next state. The value of $\mathcal{F}(l)$ is the set of all such next states l'.

Because we only add (and never delete) nodes and strands in the next state transition, and since the next state transition covers all possibilities of binding a goal, we can see that $\mathcal{F}$ is *complete-inclusive*. The proof sketch can be found in [23].

Notice that due to the most general substitution in a goal-binding, the nodes and strands in a state can contain free variables. A semi-bundle $S(\vec{x})$ in a state s with free variables, $\vec{x}$, represents a set of variable-free semi-bundles, $\{[\vec{\sigma}/\vec{x}]s(\vec{x})\}$, where $\vec{\sigma}$ is a substitution of values. Thus, the state $s(\vec{x})$ represents a set of variable-free states. Hence, we can represent states and state transitions much more efficiently Also, notice that we only add nodes and strands when necessary due to the exact causal relation, which reduces the state space even further.

4.5. Unreachability Analysis

In order to reduce the search space even further, we can use *unreachability theorems* to prove early that a state is unreachable. Recall that a state is unreachable if there is no bundle containing it. For practical purposes, we need to be able to express such theorems as computable predicates θ, that is, if $\theta(s)$ is true, then the state s is unreachable. Thus, we can simply eliminate s from the set of states immediately.

The unreachability theorems can be specific to a particular protocol, or can be general theorems that are not restricted to any concrete example. The latter can be proven once and for all and included in the core of the tool. This let the model checker easily incorporate results from theorem proving techniques and greatly reduce the state space explored. We give two examples of the unreachability theorems as follows.

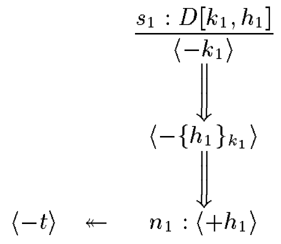
Proposition 4.5. *Let C be a bundle, let k* $\in K \setminus K_p$. *If k never originates in a regular node, then k* $\not\sqsubseteq$ *term(p) for any penetrator node* $p \in C$.

The proof of this proposition is in [25].

If a protocol is well-typed, in other words, all messages specified in a protocol have fixed types, we can always compute a number which is the maximum number of nested encryption operations in a message that can be generated by a legitimate principal, denoted as E_m. Then the following proposition holds.

Proposition 4.6. *If a state* l *contains the following nodes as shown in the figure:*
n_1 *is the binder of a term* t_1, n_1 *is on a penetrator strand* $s_1 = D[k_1, h_1]$, $t_1 \Subset h_1$*;*
t_1 *contains a number of nested encryptions which is greater than or equal to* E_m.
Then l *is unreachable, which means there is no any bundle which will contain* l.

This proposition is used to eliminate an infinite expanding of type D strand.


$$\begin{tikzcd}
 & \underline{s_1 : D[k_1, h_1]} \\
 & \langle -k_1 \rangle \arrow[d, Rightarrow] \\
 & \langle -\{h_1\}_{k_1} \rangle \arrow[d, Rightarrow] \\
\langle -t \rangle & n_1 : \langle +h_1 \rangle \arrow[l, twoheadrightarrow]
\end{tikzcd}$$

5. Discussion

Our approach has a number of advantages over previous techniques for automatic analysis of security protocols.

First, unlike most model checkers, our approach can reason about problems of infinite state space. When the evaluation procedure terminates, it will have found a mapping from the infinite state space to a finite state space while preserving the validity of well-formed formulae. The mapping is from the set of bundles to the non-empty leaf states of the search tree as described in section 4.3. Thus, by analyzing the finite state space, Athena can derive a proof that a security property holds under any configuration of the protocol execution.

Second, the state transition in Athena is not asynchronously composed of independent process transitions, but based on goal-bindings. Therefore, we avoid the state space explosion caused by asynchronous composition, while others suffer severely from this problem. Most other approaches are based on a trace-based model, where the global state is a list of independent local processes of the principals, and the global state transition is the asynchronous composition of the local process transitions. Hence, concurrent events are modeled by allowing all possible interleaving, and the state space grows exponentially with the number of principals in the protocol execution [20, 21]. In practice, these approaches have difficulties handling protocol executions with more than two initiators and responders. Although some researchers have explored observations that reduce the number of states and paths to be checked [22], and others are pursuing partial order reduction techniques, these techniques will still require a large number of states and paths to be checked unnecessarily.

Third, our states and state transitions, especially the goals and goal-bindings, capture the exact causal relations. By eliminating unnecessary information in state structures, our state representation is more compact and efficient than most other approaches. Each state in our model actually represents a set of traces in a trace-based model by linearizing the paths in the graph of the state.

Forth, unlike most other approaches, Athena takes advantage of symbolic state transitions instead of explicit state search. In explicit state search, state transitions have to be made from one explicit state to another. With security protocols, however, explicit states can be naturally grouped. A group of states can transit to the next group of states, similar to symbolic state transitions. In our approach, we allow a state to contain free variables. A state $s(\vec{x})$ with free term variables, $\vec{x}$, represents a class of variable-free states, $\{[\vec{\sigma}/\vec{x}]s(\vec{x})\}$, where $\vec{\sigma}$ is a substitution of term values. A state transition between two states which contain free variables represents a set of state transitions between two variable-free states. Thus, Athena can represent states and state transitions much more efficiently. As a special case of this, we naturally avoid the symmetry redundancy problem [3], while others suffer from it. For example, in most approaches, the model is composed of a number of replicated components, i.e. two initiators and two responders. A state can be identical to a second state up to a substitution of names and variable values. Thus, it is enough to just analyze one of the states because the analysis will produce the same outcome for both. Although some researchers are pursuing symmetry reduction techniques for explicit state search, it is still difficult to do aggressive symmetry reduction.

#init.	*#resp.*	*Murφ*	*Brutus (Red off)*	*Brutus (Red on)*	*Athena*
1	1	1,706	1,208	146	19
2	2	514,550	***	186,340	19

Brutus (Red on) means Brutus with partial order and symmetry reduction.
Brutus (Red off) means Brutus without partial order and symmetry reduction.

Table 1. Number of states explored in analyzing NSL protocol.

Fifth, Athena uses backward search instead of forward search. With forward search, all the participating principals have to be pre-stated. Our approach starts with a simple initial strand and then add new strands only when necessary according to the exact causal relation. Hence, we reduce the state space explored by avoiding the exploration of many unnecessary states and paths.

Sixth, our approach can use unreachability theorems to prove early that a state is unreachable, hence, prune the state space. The unreachability theorems can be specific to a particular protocol, or can be general theorems that are not restricted to any concrete example. The latter can be proven once and for all and included in the core of the tool. This let the model checker incorporate results from theorem proving techniques easily and systematically.

Our experimental results demonstrate the advantages of using Athena. For the NSL protocol, Athena explored 19 states and proved the agreement property of the protocol under any configuration. The sketch of the analysis can be found in the appendix. As shown in table 1, the number of states explored by Murφ and *Brutus* grows rapidly as the number of initiators and responders increase. For example, with two initiators and responders, Murφ and *Brutus* explore over 10,000 times as many states as Athena. We have also used Athena to find the known attacks in the Needham-Schroeder protocol [18] and the TMN protocol [24]. We have also used Athena to prove certain properties of the 1KP protocol [1] and the Kerberos protocol [10, 11]. Detailed results about these experiments can be found in [23].

6. Conclusions

We propose an efficient automatic checking algorithm, Athena, for analyzing security protocols. Athena incorporates a logic that can express security properties including authentication, secrecy and properties related to electronic commerce. We have developed an automatic procedure for evaluating well-formed formulae in this logic. For a well-formed formula, if the evaluation procedure terminates, it will generate a counterexample if the formula is false, or provide a proof if the formula is true. Athena also exploits several state space reduction techniques. It naturally avoids the state space explosion problem commonly caused by asynchronous composition and symmetry redundancy. It also has the advantage that it can easily incorporate results from theorem proving through unreachability theorems to further reduce the state space explored and increase the likely-hood of termination. As shown in our experiments, we successfully rediscover known flaws or provide proofs of properties for protocols with exploring only tens of states.

Acknowledgements

This research has benefited from discussions with other members of Ed Clarke's research group. The author also thanks Jon Millen for his helpful suggestions.

References

[1] M. Bellare, J. Garay, R. Hauser, A. Herberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. *i*KP - a family of secure electronic payment protocols. In *Proceedings of the 1st USENIX Workshop on Electronic Commerce*, July 1995.

[2] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, DEC Systems Research Center, February 1989.

[3] E. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[5] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *In Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[6] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1989.

[7] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *In Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, California*, May 1990.

[8] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[9] D. Kindred and J. M. Wing. Fast, automatic checking of security protocols. In *USENIX 2nd Workshop on Electronic Commerce*, 1996.

[10] J. Kohl and B. Neuman. The kerberos network authentication service. Internet Request For Comment RFC-1510, 1993.

[11] J. Kohl, B. Neuman, and T. Ts'o. The evolution of the kerberos authentication service. IEEE Computer Society Press, 1994.

[12] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[13] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 31–43, 1997.

[14] C. Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of the 1994 Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1994.

[15] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology - Asiacrypt '94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–150. Springer-Verlag, 1995.

[16] J. Millen. The Interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 251–260. IEEE Computer Society Press, 1995.

[17] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murφ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.

[18] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[19] L. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 70–83, 1997.

[20] D. Peled. All from one, one for all: on model checking using representatives. In C. Courcoubetis, editor, *Proceedings of the Fifth Workshop on Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423, Elounda, Greece, June 1993. Springer-Verlag.

[21] D. Peled. Combining partial order reductions with on-the-fly model-checking. In D. L. Dill, editor, *Proceedings of the Sixth Workshop on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 377–390, Stanford, CA, USA, June 1994. Springer-Verlag.

[22] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, 1998.

[23] D. Song. Inside Athena. Technical Report. Carnegie Mellon University. CMU-CS-99-125, 1999.

[24] M. Tatebayashi, N. Matsuzaki, and D. Newman. Key distribution protocol for digital mobile communication systems. In *Proc. CRYPTO'89*, 1990.

[25] F. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, 1998.

[26] T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1993.

Appendix A : Proofs of the Propositions

Lemma 4.1. If $\mathcal{H}$ is an algorithm that decides the validity of any wff of the form $\forall C.f_1 \Rightarrow f_2$ in a model M in finite steps, where f_1 is a conjunction and f_2 is a disjunction of the atomic propositions, then there exists an algorithm to evaluate any formula of the form $\forall C.F$ in the model M in finite steps, for any propositional formula F.

Proof sketch. Since F is a propositional formula, it can always be transformed into a Conjunctive Normal Form: $F = f_1 \wedge f_2 \wedge \ldots \wedge f_n$, where $f_i, i = 1, \ldots, n$, are disjunctions of the atomic propositions and their negations. Because universal quantifiers distribute over conjunctions,

$$\forall C.F \equiv (\forall C.f_1) \wedge (\forall C.f_2) \wedge \cdots \wedge (\forall C.f_n).$$

For each $i \in (1, \ldots, n)$, f_i can be written as $(\vee_k \alpha_k) \vee (\vee_l \neg\alpha_l)$, where α_k and α_l are atomic propositions. This is equivalent to $(\wedge_l \alpha_l) \Rightarrow (\vee_k \alpha_k)$. So f_i can be evaluated in finite steps by $\mathcal{H}$. Hence, it is easy to see that we can decide F in finite steps. □

Appendix B : The NSL Example

We show a sketch how to prove the agreement property of the NSL protocol using Athena. Due to the space restriction, we omit some steps. The formula that needs to be checked is

$$\forall C.\mathit{Resp}[A_0, B_0, N_{a0}, N_{b0}] \in C \Longrightarrow \mathit{Init}[A_0, B_0, N_{a0}, N_{b0}] \in C.$$

The initial state l_0 is as

$$\begin{tikzcd}
\underline{\mathit{Resp}[A_0, B_0, N_{a0}, N_{b0}]} \\
n_1 : \langle -\{N_{a0} \cdot A_0\}_{K_{b0}} \rangle \arrow[d, Rightarrow] \\
n_2 : \langle +\{N_{a0} \cdot N_{b0} \cdot B_0\}_{K_{a0}} \rangle \arrow[d, Rightarrow] \\
n_3 : \langle -\{N_{b0}\}_{K_{b0}} \rangle
\end{tikzcd}$$

To bind the goal $(n_3, \langle -\{N_{b0}\}_{K_{b0}} \rangle)$, there are three possibilities as follows.

1. l_1 :

$$\begin{tikzcd}
\underline{\mathit{Resp}[A_0, B_0, N_{a0}, N_{b0}]} & \underline{\mathit{Init}[A_x, B_0, N_{ax}, N_{b0}]} \\
\langle -\{N_{a0} \cdot A_0\}_{K_{b0}} \rangle \arrow[d, Rightarrow] & \langle +\{N_{ax} \cdot A_x\}_{K_{b0}} \rangle \arrow[d, Rightarrow] \\
\langle +\{N_{a0} \cdot N_{b0} \cdot B_0\}_{K_{a0}} \rangle \arrow[d, Rightarrow] & \langle -\{N_{ax} \cdot N_{b0} \cdot B_0\}_{K_{ax}} \rangle \arrow[d, Rightarrow] \\
\langle -\{N_{b0}\}_{K_{b0}} \rangle & \langle +\{N_{b0}\}_{K_{b0}} \rangle \arrow[l, two heads]
\end{tikzcd}$$

2. l_2 :

$$\begin{tikzcd}
\underline{\mathit{Resp}[A_0, B_0, N_{a0}, N_{b0}]} & \underline{E[K_{b0}, N_{b0}]} \\
\langle -\{N_{a0} \cdot A_0\}_{K_{b0}} \rangle \arrow[d, Rightarrow] & \langle -K_{b0} \rangle \\
\langle +\{N_{a0} \cdot N_{b0} \cdot B_0\}_{K_{a0}} \rangle \arrow[d, Rightarrow] & \langle -N_{b0} \rangle \arrow[d, Rightarrow] \\
n_4 : \langle -\{N_{b0}\}_{K_{b0}} \rangle & \langle +\{N_{b0}\}_{K_{b0}} \rangle \arrow[l, two heads]
\end{tikzcd}$$

3. l_3 :

$$\begin{tikzcd}
\underline{\mathit{Resp}[A_0, B_0, N_{a0}, N_{b0}]} & \underline{D[K_I, t_1 \cdot N_{b0} \cdot t_2]} \\
\langle -\{N_{a0} \cdot A_0\}_{K_{b0}} \rangle \arrow[d, Rightarrow] & \langle -K_I \rangle \\
\langle +\{N_{a0} \cdot N_{b0} \cdot B_0\}_{K_{a0}} \rangle \arrow[d, Rightarrow] & \langle -\{t_1 \cdot \{N_{b0}\}_{K_{b0}} \cdot t_2\}_{K_I} \rangle \arrow[d, Rightarrow] \\
\langle -\{N_{b0}\}_{K_{b0}} \rangle & \langle +t_1 \cdot \{N_{b0}\}_{K_{b0}} \cdot t_2 \rangle \arrow[l, two heads]
\end{tikzcd}$$

where t_1, t_2 are free term variables.

Later steps will show that l_1 lead to the correct execution of the protocol which contains the strand $\mathit{Init}[A_0, B_0, N_{a0}, N_{b0}]$. According to proposition 4.6, we can see that there is no bundle that can contain l_3 since in this protocol $E_m = 1$. We now compute the next states of l_2.

l_4 :

$$\begin{tikzcd}
\underline{\mathit{Resp}[A_0, B_0, N_{a0}, N_{b0}]} & \underline{E[K_{b0}, N_{b0}]} & \underline{D[K_I, t_3 \cdot N_{b0} \cdot t_4]} \\
& & \langle -K_I \rangle \\
\langle -\{N_{a0} \cdot A_0\}_{K_{b0}} \rangle \arrow[d, Rightarrow] & \langle -K_{b0} \rangle & n_5 : \langle -\{t_3 \cdot N_{b0} \cdot t_4\}_{K_I} \rangle \arrow[d, Rightarrow] \\
\langle +\{N_{a0} \cdot N_{b0} \cdot B_0\}_{K_{a0}} \rangle \arrow[d, Rightarrow] & \langle -N_{b0} \rangle \arrow[d, Rightarrow] & \langle +t_3 \cdot N_{b0} \cdot t_4 \rangle \arrow[l, two heads] \\
\langle -\{N_{b0}\}_{K_{b0}} \rangle & \langle +\{N_{b0}\}_{K_{b0}} \rangle \arrow[l, two heads] &
\end{tikzcd}$$

where t_3, t_4 are free term variables. Later steps will show that l_4 is unreachable. Finally, we finish the proof after exploring 19 states with the conclusion that NSL protocol preserves the non-injective agreement property. Hence, the agreement property holds as well, as mentioned in section 3. The more detailed analysis of this example can be found in [23].