# Athena: a novel approach to efficient automatic security protocol analysis *


[†]Dawn Song [‡]Sergey Berezin [†]Adrian Perrig

[†] Dept. of Computer Science
UC Berkeley
Berkeley, CA, 94720
dawnsong@cs.berkeley.edu
perrig@cs.berkeley.edu

[‡] Dept. of Computer Science
Carnegie Mellon University
berez@cs.cmu.edu



**Abstract**

We propose a new efficient automatic verification technique, Athena, for security protocol analysis. It uses a new efficient representation — our extension to the Strand Space Model, and utilizes techniques from both model checking and theorem proving approaches. Athena is fully automatic and is able to prove the correctness of many security protocols with arbitrary number of concurrent runs. The run time for a typical protocol from the literature, like the Needham-Schroeder protocol, is often a fraction of a second.

Athena exploits several different techniques that enable it to analyze infinite sets of protocol runs and achieve such efficiency. Our extended Strand Space Model is a natural and efficient representation for the problem domain. The security properties are specified in a simple logic which permits both efficient proof search algorithms and has enough expressive power to specify interesting properties. The automatic proof search procedure borrows some efficient techniques from both model checking and theorem proving. We believe that it is the right combination of the new compact representation and all the techniques that actually makes Athena successful in fast and automatic verification of security protocols.



*This research is suported in part by the Defense Advanced Research Projects Agency under DARPA contract N6601-99-28913 (under supervision of the Space and Naval Warfare Systems Center San Diego), by the National Science foundation under grant FD99-79852, and by the United States Postal Service under grant USPS 1025 90-98-C-3513. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or any of its agencies, DARPA, NSF, USPS.

# 1 Introduction

Security protocols are communication protocols that use cryptography to achieve goals such as authentication and key distribution. They are the basis of any secure system, and usually have to be redesigned as new systems and applications emerge. However, many examples have shown that security protocols may contain subtle errors that remain undiscovered for years, even when the protocols are carefully designed. Therefore, we cannot rely on ad hoc and informal ways of reasoning, and it is crucial to apply formal methods to security protocol analysis.

One of the promising directions in this area is the use of automatic tools. Automatic tools have the practical advantage that they are easy to use, and they have been successfully applied to find flaws in some proposed protocols. Unfortunately, previously available tools severely suffer from the state space explosion which prevents them from analyzing complicated protocols. This is mainly due to the complexity of the intruder behavior, asynchronous composition, and symmetry redundancy in the traditional verification approaches. Most of these tools are also limited to checking the properties of a security protocol under small configurations, e.g. with two initiators and two responders.

In this paper we propose a new automatic verification algorithm, *Athena*. Athena can provide proofs of properties of a security protocol under arbitrary configurations, and exploits several state space reduction techniques which greatly reduce the state space explosion problem.

## 1.1 Related Work

Previous automatic tools for security protocol analysis include general-purpose model checkers such as FDR [18, 15] and Mur$\varphi$ [26], and special-purpose model checkers, for example, the Interrogator [25] and Brutus [10]. These tools start with an initial state of a protocol execution and then exhaustively search through all possible sequences of actions of both legitimate principals and a modeled attacker to see whether an attack could happen. All these tools have been successfully applied to find attacks on protocols. But they all suffer from two main problems:

1. **Bounded number of principles.** All of these tools have to specify in advance the maximum number of principals that can participate in the protocol, which means that they can only check whether a protocol is correct for a limited number of participants. Even if they do not find an attack on the protocol, it is still possible that the protocol might have an attack with a higher number of participants. Lowe has overcome this problem in some cases by proving manually that a small number of participants is enough to prove the correctness for arbitrary number of runs [18, 20].
2. **State space explosion problem.** Although these tools explore a number of state reduction techniques, they still suffer from the state space explosion problem. This restricts their applicability to protocols with only a small number of participants, e.g. three or five, which send and receive a small number of messages in each protocol run.

The above techniques are based on the traditional *trace-based model* (TBM), where each principal is modeled by a *process*. The global state space of the protocol is a Cartesian product of local state spaces of individual processes. The state transition of each local process is based on sending or receiving a message. The processes are composed *asynchronously*, and the global transition relation is an interleaving of local transitions. Due to such parallel composition and the interleaving semantics, the number of states and transitions to be explored grows exponentially with the number of participants involved in the protocol [30, 31]. Hence, these approaches suffer severely from the state space explosion problem, and are often only able to verify protocols with small number of participants, e.g. three or five. This problem is especially acute for the explicit state enumeration techniques.

Multiple concurrent runs of a security protocol may yield many principals with identical roles, and in this case the transition graph often possesses a lot of symmetries. Although some reduction techniques have been used to reduce the search space, including partial order and symmetry reductions [8], a large number of states and transitions still have to be checked unnecessarily.

The NRL Protocol Analyzer [22] is another special-purpose tool that uses a theorem proving approach for security protocol analysis. It starts from an insecure state and performs a backward search trying to prove that this insecure state is unreachable. It can use many theorem proving techniques such as inductive methods. The advantage of this approach is that it can prove a protocol correct for arbitrary number of participants. However, it often requires non-trivial amount of human interaction and expertise, and the running time could be much slower than in the model checking approach [24]. The NRL Protocol Analyzer reduces symmetry redundancy by using symbolic variables. But it also uses the traditional TBM and asynchronous composition, and hence, is still not optimal in terms of the state space explosion problem.

There are also other approaches which use general purpose theorem provers such as Isabelle [29]. These approaches require more expertise with theorem provers and more human interaction, and have the disadvantage that they cannot generate counterexamples directly.

Apart from the above approaches, there are also some tools [4, 5, 17] based on belief logics such as BAN logic [6] and GNY logic [13], that have been used to find flaws in some protocols.

## 1.2 Overview of Our Techniques

We use our extension to the *Strand Space Model* (SSM) instead of the traditional TBM to represent protocol executions. Thayer, Herzog and Guttman proposed SSM for protocol representation and demonstrated how to use SSM to prove certain security properties manually, for example authentication and secrecy [36]. (Notice that SSM is a new model for representing protocol executions and has nothing to do with strands in the context of category theory or other mathematical theories.) Because SSM has the advantage that it contains the exact causal relation information, the authors have been able to derive much simpler proofs of a protocol's properties in comparison with the traditional TBM. However, their proof technique requires a lot of human insight and will be difficult to automate. We have extended their model and developed a new algorithm, *Athena*, for analyzing security protocols automatically.

We have designed a new logic suitable for SSM that can express various security properties, including authentication, secrecy, and electronic commerce properties. We have also developed an automatic procedure for evaluating well-formed formulas in this logic. If the evaluation procedure terminates, it generates either a proof or a counterexample, depending on the validity of the formula. The validation procedure is not guaranteed to terminate, but it terminates for the protocols we experimented with. Even when the procedure does not terminate for a general case (when we allow arbitrary configurations of the protocol execution), termination can 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, Mur$\varphi$, and Brutus.

We formulate our verification procedure in terms of a proof search in a very specialized proof system. Athena first transforms the security property to be verified into an *initial sequent* which contains an *initial state*. It then applies a small set of *inference rules* with certain decision procedures to the states, building a proof tree, until it either completes the proof or refutes a sequent. In the latter case Athena reports the protocol to be incorrect, and the state of the refuted sequent represents a counterexample, or a successful attack on the protocol.

One main difference between Athena and previous approaches is that Athena uses fundamentally different representation of protocol executions. As described above, most of the existing approaches are based on TBM, in which each principal is modeled by a *process*, and its local transition is based on sending/receiving a message. The protocol model in TBM is the asynchronous parallel composition of these processes. Instead, Athena uses our extension to SSM, a much more compact *state* structure based on *semi-bundles* and *goal-bindings*. The goal-binding is the *causal relation* "$\twoheadrightarrow$" that captures the exact information about the origins of messages in a protocol execution. A set of protocol runs that differ only in the order of interleaving executions of individual parties is in fact represented by one state in Athena, and Athena can reason about all such executions simultaneously. This form of the state structure allows us to develop efficient state search procedures avoiding the exponential growth of the state space due to asynchronous composition.

Similarly to the NRL Protocol Analyzer, Athena also takes advantage of symbolic state representation, in contrast to the explicit representation used in most of previous model checking approaches. In our approach, we allow a state to contain free variables. In a way, a state is parameterized by these free variables and, therefore, represents a (possibly infinite) set of concrete protocol executions. Thus, Athena can represent states and state transitions efficiently, and in particular, naturally reduces the symmetry redundancy problem.

To reduce the search space further, our approach can use *pruning theorems* to prove early that some states do not contribute to the final result, and we can prune such states from the search space immediately. Pruning theorems can be either specific to a particular protocol, or general theorems that are not restricted to any concrete example. In the latter case they can be proven once and for all and included in the core of the tool. This lets our model checker incorporate results from theorem proving easily and systematically. Note that the interface for supplying user-defined pruning theorems is mainly for convenience reasons for advanced users. For non-expert users, it is not required and often unnecessary to supply their own pruning theorems. Currently Athena contains two built-in pruning theorems and they are shown to be effective and sufficient in most cases in our experiments.

Given a security property formula, Athena first transforms this formula into an *initial state*. (Note that as explained later in section 4, this initial state is not an insecure state but rather a state which represents a possibly infinite set of protocol executions which satisfy certain properties.) The proof search then starts with this initial state, and new events and participants are added only when necessary according to the exact causal relation. Hence, we reduce the search space by avoiding the exploration of many unnecessary states and paths. In contrast, with forward search all the participating principals have to be pre-stated, which means that one might have to explore many more unnecessary states and paths.

As demonstrated by our experiments (Section 5), all these techniques dramatically reduce the state space explored, and our tool outperforms previous automatic approaches. We discuss various techniques used in Athena in more detail in Section 4.

The paper is organized as follows. We first review some background and the notion of SSM (section 2). Then we introduce a logic to reason about strand spaces and show how to use this logic to specify security properties (section 3). Next we explain our verification algorithm (section 4), show some experimental results and further discussion (section 5), and finally conclude (section 6).

# 2 Strand Space Model

This section is primarily a review of concepts developed by Thayer, Herzog and Guttman [36]. First, we explain the notion of *terms* that are used to represent the messages in the protocols. Then, we introduce *strands, strand spaces*, and *bundles*, and show how to represent protocols using strands. Finally, we give a formal description of the *penetrator model*.

## 2.1 Message Terms

The set of atomic terms is the union of a set of *Text terms* $\mathcal{T}$ and a set of *Key terms* $\mathcal{K}$, where

- Text terms from $\mathcal{T}$ contain several different types of terms, such as principal names, nonces, or bank account numbers.
- Key terms from $\mathcal{K}$ contains a set of keys disjoint from $\mathcal{T}$. In asymmetric crypto systems, $K^{-1}$ (for $K \in \mathcal{K}$) represents $K$'s opposite member in a public-private key pair. In symmetric key systems, $K^{-1} = K$.

The set of all *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 and $k$ 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

$$\{m\}_K = \{m'\}_{K'} \iff m = m' \wedge K = K'.$$

Thayer, Herzog and Guttman defined in [36] the *subterm relation* $\sqsubseteq$: a term $a_1$ is a *subterm* of term $a_2$ if $a_1$ appears in $a_2$. We also define an *interm relation* $\Subset$, such that $a_1$ is an *interm* of $a_2$ if $a_1$ can be extracted from $a_2$ without application of the decryption operation. The formal definition of the two relations is the following.

- *subterm relation* $\sqsubseteq$:
  - $a \sqsubseteq t$ *for* $t \in \mathcal{T}$ *iff* $a = t$; or
  - $a \sqsubseteq k$ *for* $k \in \mathcal{K}$ *iff* $a = k$; or
  - $a \sqsubseteq \{g\}_k$ *iff* $a \sqsubseteq g \vee a = \{g\}_k$; or
  - $a \sqsubseteq gh$ *iff* $a \sqsubseteq g \vee a \sqsubseteq h \vee a = gh$.

- *interm relation* $\Subset$:
  - $a \Subset t$ *for* $t \in \mathcal{T}$ *iff* $a = t$; or
  - $a \Subset k$ *for* $k \in \mathcal{K}$ *iff* $a = k$; or
  - $a \Subset \{g\}_k$ *iff* $a = \{g\}_k$; or
  - $a \Subset gh$ *iff* $a \Subset g \vee a \Subset h \vee a = gh$.

## 2.2 SSM : Strands, Strand Spaces and Bundles

The notions in this subsection are mainly from the paper [36]. 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 external 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, and denote them $+$ and $-$ respectively. That is, we will always assume $Act = \{+, -\}$.

**Events.** An *event* is a pair $\langle action,\ a\rangle$, where $action \in Act$, and $a \in A$ is the argument of the action from the set of terms $A$. Since we only have *send* $(+)$ and *receive* $(-)$ actions, we denote events as *signed terms* $+a$ and $-a$. The set of events, or signed terms, is denoted by $\pm A$, and the set of finite sequences of signed terms is $(\pm A)^*$.

**Strands and Strand Spaces.** A protocol defines a sequence of events for each principal's role. A strand represents a sequence of a principal's actions in a particular protocol run, and is an instance of a role.

A strand space is a set of strands $\Sigma$ with a trace mapping

$$\mathsf{tr} :\ \Sigma \to (\pm A)^*.$$

1. A *node* is a pair $\langle s, i\rangle$, with $s \in \Sigma$ and $i$ an integer satisfying $1 \leq i \leq \mathrm{length}(\mathsf{tr}(s))$. We say that $n = \langle s, i\rangle$ *belongs* to the strand $s$, denoted by $n \in s$. 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 $\mathsf{index}(n) = i$ and $\mathsf{strand}(n) = s$. If $(\mathsf{tr}(s))_i = \sigma a$, where $\sigma \in \{+, -\}$, then $\mathsf{term}(n) = a$ and $\mathsf{sign}(n) = \sigma$. In other words, a node is a particular event in a given strand, and we will often use nodes as events where it is unambiguous.

3. If $n_1, n_2 \in N$, then $n_1 \to n_2$ means that $n_1 = +a$ and $n_2 = -a$ for some term $a$. This represents sending a message $a$ from $n_1$ to $n_2$.
4. If $n_1, n_2 \in N$, then $n_1 \Rightarrow n_2$ means that $n_1$ and $n_2$ occur in the same strand with $\mathsf{index}(n_2) = \mathsf{index}(n_1) + 1$. This represents an event $n_1$ followed immediately by $n_2$.
5. A term $t$ *originates* from a node $n \in N$ iff $\mathsf{sign}(n) = +$ and $t \sqsubseteq \mathsf{term}(n)$, and whenever $n'$ precedes $n$ on the same strand, $t \not\sqsubseteq \mathsf{term}(n')$.
6. A term $t$ *uniquely originates* from node $n$ iff $t$ originates on a unique node $n$. 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 a protocol execution under some configuration.

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

- $C$ is a non-empty, finite and acyclic graph;
- If $n_1 \in C$ and $\mathsf{sign}(n_1) = -$, then there is a unique $n_2$ such that $n_2 \to n_1$ is an edge in $C$;
- If $n_1 \in C$ and $n_2 \Rightarrow n_1$, then $n_2 \Rightarrow n_1$ is an edge in $C$;

We say that a strand $s$ belongs to $C$ ($s \in C$) if for any node $n \in s$, $n \in N_C$. In some cases we treat a bundle as a set of strands $\{s \mid s \in C\}$. Note, that this set is closed under the *substrand* relation. A strand $s$ is called a *substrand* of $s'$ (denoted $s \subseteq s'$) if $s'$ contains all the nodes of $s$.

**Causal Precedence.** Let $\Sigma$ be a strand space. For nodes $n_1$, $n_2 \in \Sigma$, define $n_1 \preceq_\Sigma n_2$ *iff* there is a sequence of zero or more edges of $\to$ and $\Rightarrow$ leading from $n_1$ to $n_2$ in $\Sigma$. The relation $\preceq_\Sigma$ expresses a *causal precedence*. In other words, $\preceq_\Sigma$ is a reflexive and transitive closure of $\to$ and $\Rightarrow$:

$$\preceq_\Sigma = (\to \cup \Rightarrow)^*.$$

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

The proof of this lemma can be found in [36].

## 2.3 Protocol Specification Using Strands

A protocol usually contains several *roles*, such as *initiators, responders* and *servers*. The sequence of actions of each principal type is predefined by the protocol and represented as a *role*, or a *parameterized strand*. Parameters usually include principal names and nonces. We denote a role with parameters by *role*[*parameter list*]. Instantiating parameters of a role yields a particular strand representing a trace of a principal in a protocol run. A legal execution of a protocol forms a bundle, in which the strands of the legitimate principals are restricted to the predefined traces. The strands of the legitimate principals are referred to as *regular strands*. A bundle can also contain *penetrator strands*. We explain them in more detail in the next subsection. We now give an example of the Needham-Schroeder protocol [27] with the fix given by Gavin Lowe in [18]. We will refer to this protocol as *NSL* in the rest of the paper. Using the standard notation, the protocol is as follows.

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

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

$$
\begin{array}{ccc}
\underline{\mathit{Resp}[A, B, N_a, N_b]} & \qquad & \underline{\mathit{Init}[A, B, N_a, N_b]} \\
1 : \langle -\{N_a \cdot A\}_{K_b} \rangle & & 1 : \langle +\{N_a \cdot A\}_{K_b} \rangle \\
\Downarrow & & \Downarrow \\
2 : \langle +\{N_a \cdot N_b \cdot B\}_{K_a} \rangle & & 2 : \langle -\{N_a \cdot N_b \cdot B\}_{K_a} \rangle \\
\Downarrow & & \Downarrow \\
3 : \langle -\{N_b\}_{K_b} \rangle & & 3 : \langle +\{N_b\}_{K_b} \rangle
\end{array}
$$

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 $Init[A, B, N_a, N_b]$, and $N_b$ uniquely originates on the second node of the responder's strand $Resp[A, B, N_a, N_b]$. A responder's strand instantiated with parameters $[A_0, B_0, N_{a0}, N_{b0}]$ is the following :

$$
\begin{array}{c}
\underline{\mathit{Resp}[A_0, B_0, N_{a0}, N_{b0}]} \\
\langle -\{N_{a0} \cdot A_0\}_{K_{b0}} \rangle \\
\Downarrow \\
\langle +\{N_{a0} \cdot N_{b0} \cdot B_0\}_{K_{a0}} \rangle \\
\Downarrow \\
\langle -\{N_{b0}\}_{K_{b0}} \rangle
\end{array}
$$

## 2.4 Instantiation and Substitution

In our verification algorithm described in Section 4 we use unification to instantiate roles in a protocol. This section defines the basic notions of *substitution*, *unification*, and the *most general unifier*.

**Definition 2.2.** A *substitution* $\sigma = [\vec{t}/\vec{x}]$ is a mapping of variables $\vec{x}$ to terms $\vec{t}$. The substituted terms do not have to be closed and may contain free variables that can later be replaced by other substitutions.

*Applying a substitution* $\sigma$ to a particular term $t'$ yields a new term $\sigma(t')$ obtained from $t'$ by simultaneously replacing all free variables $\vec{x}$ in $t'$ with terms $\sigma(\vec{x})$.

We will also apply a substitution to strands, roles, and sets of strands or roles with the obvious meaning. In particular, applying a substitution to a role creates a strand which is an instance of that role. The formal parameters of the role act in this case as free variables.

**Definition 2.3.** A substitution $\sigma$ is called a *unifier* of terms $t$ and $t'$ if $\sigma(t) = \sigma(t')$.

**Definition 2.4.** A substitution $\sigma$ is a *most general unifier* (MGU) of $t$ and $t'$ if it is a unifier, and for any other unifier $\sigma'$ there exists a substitution $\gamma$ such that $\sigma' = \sigma \circ \gamma$.

## 2.5 The Penetrator Model

We use a similar penetrator model as the one in [36]. The penetrator $P$ has a set of initial knowledge $\mathsf{init\text{-}info}(P)$ which usually includes the principal names and the set of keys $K_p$ that are known initially to the penetrator. $K_p$ usually contains all the public keys, all the private keys of the penetrator, and all the symmetric keys initially shared between the penetrator and principals playing by the protocol rules. It can also contain keys to model known-key attacks.

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

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

- $\mathrm{M}[t]$. Atomic message: $\langle +t \rangle$ where $t \in \mathsf{init\text{-}info}(P)$ and $t \in \mathcal{T}$.
- $\mathrm{F}[g]$. Flushing: $\langle -g \rangle$.
- $\mathrm{T}[g]$. Tee: $\langle -g, +g, +g \rangle$.
- $\mathrm{V}[g, h]$. Concatenation: $\langle -g, -h, +gh \rangle$.
- $\mathrm{R}[g, h]$. Separation into components: $\langle -gh, +g, +h \rangle$.
- $\mathrm{K}[k]$. Key:$\langle +k \rangle$ where $k \in K_p$.
- $\mathrm{E}[k, h]$. Encryption: $\langle -k, -h, +\{h\}_k \rangle$.
- $\mathrm{D}[k, h]$. Decryption: $\langle -k^{-1}, -\{h\}_k, +h \rangle$.

Here the notation $\langle -t_1, \ldots, -t_k, +t'_1, \ldots, +t'_l \rangle$ for a strand means that the terms $t_1, \ldots, t_k$ can be received in any order, and terms $t'_1, \ldots, t'_l$ can be sent in any order, but all terms $t_i$ must be received before any of the $t'_j$ is sent. That is, for instance, both strands

$$-g \Rightarrow -h \Rightarrow +gh$$

and

$$-h \Rightarrow -g \Rightarrow +gh$$

are $\mathrm{V}[g, h]$ type of penetrator strands.

Athena has flexibility in incorporating new penetrator models easily by modifying the penetrator strands and the initial knowledge of the penetrator. In this paper we assume that the set of penetrator roles is $\Pi = \{\mathrm{M}, \mathrm{F}, \mathrm{T}, \mathrm{V}, \mathrm{R}, \mathrm{K}, \mathrm{E}, \mathrm{D}\}$.

# 3 Logic and Model

In this section, we introduce a formal model and a logic to reason about security protocols using the SSM representation, and show how various security properties are expressed in this logic.

## 3.1 Syntax

The syntax of terms consists of *strand constants* $(s, s_1, \dots)$ and *bundle variables* $(C, \dots)$. A strand constant may represent a partial strand, and in particular, a single node can be considered as a strand constant.

Propositional formulas are defined as follows:

- $s \in C$ is an (atomic) propositional formula;
- $\neg f_1$ 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 variable than $C$.

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

Notice, that in a wff $\forall C.\, f$, the formula $f$ needs to be propositional, and cannot contain any other variables than $C$. In addition, we allow only negative occurrences of atomic formulas in $f$ that reference penetrator strands. An occurrence of an atomic formula $\phi$ in $f$ is called *negative* if $\phi$ appears under the scope of odd number of negations. This restriction ensures the validity of counterexamples that our algorithm generates; namely, it is needed to apply Proposition 4.2 from Section 4.2.

We also use the obvious abbreviations:

$$\begin{array}{rcl@{\qquad}rcl}
f_1 \vee f_2 & \equiv & \neg(\neg f_1 \wedge \neg f_2) & f_1 \implies f_2 & \equiv & \neg f_1 \vee f_2 \\
f_1 \iff f_2 & \equiv & (f_1 \implies f_2) \wedge (f_2 \implies f_1) & \exists C.f & \equiv & \neg\forall C.\neg f \\
\mathcal{W} \subseteq C & \equiv & \bigwedge_{s \in W} s \in C & & &
\end{array}$$

## 3.2 Semantics

Let the set of nodes be $N$. For a given protocol $p$, define the set of all strands for this protocol (both regular and penetrator ones) as $\Sigma_p$; its execution traces form a set of bundles $\mathcal{D}_p$. A *model* $M_p$ for a given protocol $p$ is a tuple $M_p = (N,\ \Sigma_p,\ \mathcal{D}_p,\ \mathcal{I})$, where $\mathcal{I}$ is the interpretation of strand constants and bundle variables. We write $M' = M[\mathcal{I}(C) \leftarrow c]$ for some bundle $c$ to denote a new model $M'$ which is identical to $M$ except that $\mathcal{I}'(C) = c$. The semantics of formulas in our logic is the following:

- $\mathcal{I}(s) \in \Sigma_p$ and $\mathcal{I}(C) \in \mathcal{D}_p$ are a strand and a bundle respectively, that are assigned to the constant $s$ and the variable $C$ by the interpretation $\mathcal{I}$.
- $M \models s \in C$ *iff* $\mathcal{I}(s) \in \mathcal{I}(C)$.
- 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* $\forall c \in \mathcal{D}_p.\, M[\mathcal{I}(C) \leftarrow c] \models f$.

## 3.3 Specifying Security Properties in the Logic

Our logic can express a variety of security properties, including ones for authentication, secrecy, and electronic commerce. In this paper we mainly focus on the authentication and secrecy. We use similar ways for representing security properties as in [36], however, we formulate them using well-formed formulas in our logic.

#### Authentication.

Gavin Lowe [19] proposed agreement properties for authentication protocols. A protocol guarantees an *agreement property* for a participant $B$ (e.g. acting as a responder) for a certain vector of parameters $\vec{x}$, if each time the 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 $A$ initiating a session with the same parameters $\vec{x}$, supposedly with $B$.

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

The non-injective agreement property can be specified in our logic as:

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

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

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

Here $\vec{x} = [A, B, N_a, N_b]$. Because of the freshness of the nonces generated in the protocol run, usually the agreement property can be proven after the non-injective agreement property is proven, with the argument that there cannot be two strands $Init(\vec{x}) \in C$ since the nonces in $Init(\vec{x})$ are uniquely originated from only one strand. Namely, in the *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 set $\mathcal{W}$ if for every bundle $C$ that contains $\mathcal{W}$ there is no way for the intruder to receive $v$ in cleartext; that is, the strand $\mathrm{F}[v]$ does not appear in any $C$:

$$\forall C.\, \mathcal{W} \subseteq C \implies \neg\mathrm{F}[v] \in C.$$

For example, when $\mathcal{W}$ contains only a single responder strand $Resp(\vec{x})$, we can specify the secrecy property as:

$$\forall C.\, \{Resp(\vec{x})\} \subseteq C \implies \neg\mathrm{F}[v] \in C.$$

# 4 Verification Algorithm

## 4.1 The Intuition

In this section, we explain how to check the validity of a well-formed formula $F$ in our logic. Mathematically, we state our verification problem as a model checking problem: given a protocol $P$ (as a model), check that it satisfies a formula $F$. The actual verification algorithm, however, is significantly different from the classical model checking algorithms. We only consider the most interesting case of $F \equiv \forall C.\, f$ in this paper, because the other cases are relatively straightforward. Notice that the $\forall$-quantifier distributes over conjunction. Since any propositional formula $f$ can be put in a *conjunctive normal form* (CNF), we assume that $f$ has the form

$$f \equiv \bigwedge_i (\bigvee_j \neg\phi_{ij} \vee \bigvee_k \psi_{ik}),$$

where $\phi_{ij}$ and $\psi_{jk}$ are atomic formulas. Then we can transform the entire formula $F$ into a special form:

$$\begin{aligned} F &\equiv \forall C.\, \bigwedge_i (\bigvee_j \neg\phi_{ij} \vee \bigvee_k \psi_{ik}) \\ &\equiv \bigwedge_i \forall C.\, (\bigvee_j \neg\phi_{ij} \vee \bigvee_k \psi_{ik}) \\ &\equiv \bigwedge_i \forall C.\, (\bigwedge_j \phi_{ij} \implies \bigvee_k \psi_{ik}). \end{aligned}$$

Therefore, we only need to discuss the verification procedure for a formula of the form

$$F \equiv \forall C.\, (\bigwedge \Phi \implies \bigvee \Upsilon), \tag{4.1}$$

where $\Phi = \{\phi_j\}$ and $\Upsilon = \{\psi_k\}$ are sets of formulas. Note that, due to the syntactic restriction, the atomic formulas in the set $\Upsilon$ can only contain regular strands (see Section 3.1).

If $\Phi$ is empty, then $F \equiv \forall C.\, (\bigvee \Upsilon)$, and the entire formula is trivially false, because each $\psi \in \Upsilon$ is false for the empty bundle. Hence, we can assume that the antecedent of the implication is never empty. If the set of formulas $\Upsilon$ is empty, then our formula becomes

$$F \equiv \forall C.\, (\bigwedge \Phi \implies \mathsf{false}),$$

which is a particular case of Formula 4.1 and is covered by our decision procedure.

We represent our verification problem in a sequent form:

$$P;\, \Gamma \vdash \Delta, \tag{4.2}$$

where $\Gamma$ and $\Delta$ are *sets of strands* from the assumptions $\Phi$ and conclusions $\Upsilon$ of $F$ in Formula 4.1 respectively. That is,

$$\Gamma = \{s \mid (s \in C) \in \Phi\} \qquad \Delta = \{s \mid (s \in C) \in \Upsilon\}.$$

Moreover, $\Delta$ is either empty or contains only regular strands, whereas $\Gamma$ is always non-empty and may contain penetrator strands in addition to regular ones.

We write $\Delta \cap C \neq \emptyset$ to denote that there is a strand in $\Delta$ which is a substrand of some strand in $C$.

The semantics of the sequent is defined as follows:

$$[\![P;\, \Gamma \vdash \Delta]\!]_P \equiv \forall C \in \mathcal{D}_P.\, \Gamma \subseteq C \implies \Delta \cap C \neq \emptyset. \tag{4.3}$$

That is, for any bundle $C$ representing a run of the protocol $P$, if $\Gamma \subseteq C$, then $\Delta \cap C \neq \emptyset$. It is easy to see that this sequent is true if and only if the original formula $F$ (4.1) is true for the protocol $P$.

According to the semantics, in order to prove this sequent we need to consider all possible bundles that contain $\Gamma$ and check that they all have at least one strand from $\Delta$. We formulate our algorithm as a proof search procedure in a very specialized proof system.

First, we encode bundles in a special representation called *state*. Roughly speaking, a state can be viewed as a collection of properties of bundles, and we say that a state $l$ *represents* the set of bundles satisfying these properties. The *initial state* $l_0(P, \Gamma)$ specifies that any bundle it represents must satisfy the protocol $P$ and contain all of the strands from $\Gamma$. Therefore, to prove our original property, it is sufficient to prove that any bundle represented by the initial state has common strands with $\Delta$:

$$\frac{l_0(P,\, \Gamma) \vdash_P \Delta}{P;\, \Gamma \vdash \Delta}\,\mathsf{init}.$$

The new sequent $l_0 \vdash_P \Delta$ can either be solved by a decision procedure directly if it applies (rule $\mathsf{final}$), or split into multiple sequents $l_1 \vdash_P \Delta, \ldots, l_n \vdash_P \Delta$ which are then proven separately (rule $\mathsf{split}$). The precise definitions and formal semantics of the sequents are given in Section 4.2. To simplify the notation, we will omit the subscript $P$ in the sequent $l \vdash_P \Delta$ when the protocol $P$ is unambiguous from the context.

Thus, the overall verification procedure is a proof search in a very specialized proof system described in detail in Section 4.3, coupled with a separate algorithm for splitting a state in the split rule that we introduce in Section 4.4. The resulting verification procedure allows further optimizations which we outline in Section 4.5.

## 4.2 Compact State Representation

Recall that a bundle is an acyclic graph backward closed under "$\Rightarrow$" and "$\rightarrow$". It is easy to keep the graph acyclic and backward closed under "$\Rightarrow$" while completing it to a bundle, since each strand is finite and usually small. The difficult problem is to make it backward closed under "$\rightarrow$", because generally there might be an infinite number of possible ways of doing it, and we need to consider all possibilities. (Closing under "$\rightarrow$" means that every received message must be sent by some other node in the bundle.)

One of the reasons for infinite number of possibilities is unbounded forwarding of messages by the intruder: a node $n$ may receive a message either directly from $n'$, or there may be arbitrary many intermediate nodes that forward the same message from $n'$ to $n$. As we will show later, only the first sender matters, and such forwarding does not change the validity of formulas. Hence, we introduce the *semi-bundle* structure and the *goal-binding* mechanism which effectively ignores the infinite message forwarding along with other redundancies in protocol executions, such as the order of interleaving actions. Note that some theorem proving techniques might be able to use *forwarding theorems* to achieve similar goals [29], but here we simply use the compact state structures without applying any forwarding theorems in the protocol analysis process.

**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.

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

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

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

We denote the *goal binding* relation as $n' \overset{t}{\twoheadrightarrow} n$. The node $n'$ is called a *binder* of $(t, n)$. When $t$ is clear from the context, we write $n' \twoheadrightarrow n$ to denote $n' \overset{t}{\twoheadrightarrow} n$ for simplicity.

It is easy to show that any goal in a bundle has a binder.

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

- $S$ is a semi-bundle;
- $G$ is the set of unbound goals of $S$; and
- $\twoheadrightarrow$ is the relation for the goal-bindings.

Note that $G$ is redundant and can be computed from $S$ and $\twoheadrightarrow$. We keep it in the state structure for the convenience of illustration.

We say that a strand $s$ is in a state $l = \langle S, G, \twoheadrightarrow \rangle$, denoted $s \in l$, whenever $s \in S$; similarly, $\mathcal{W} \subseteq l$ for a set of strands $\mathcal{W}$ means $\mathcal{W} \subseteq S$. We also extend the notion of substitution from Section 2.4 to states in the obvious way. For a substitution $\sigma$, define

$$\sigma(\langle S,\, G,\, \twoheadrightarrow \rangle) = \langle \sigma(S),\, \sigma(G),\, \sigma(\twoheadrightarrow) \rangle,$$

where $\sigma(S)$ and $\sigma(G)$ are sets of corresponding elements in which all terms are substituted by $\sigma$, and

$$\sigma(\twoheadrightarrow) = \left\{ \ (\sigma(n'),\, \sigma(t),\, \sigma(n)) \ \middle|\ \text{where } n' \overset{t}{\twoheadrightarrow} n \ \right\}.$$

That is, $\sigma(n') \overset{\sigma(t)}{\twoheadrightarrow}_\sigma \sigma(n)$ iff $n' \overset{t}{\twoheadrightarrow} n$, where $\twoheadrightarrow_\sigma = \sigma(\twoheadrightarrow)$.

**Bundle Sets.** We define the *bundle set* $\Psi(l)$ of a state $l = \langle S_l, G_l, \twoheadrightarrow_l \rangle$ for a protocol $P$ as follows.

A bundle $c \in \Psi(l)$ iff

1. $S_l$ is a subgraph of $c$;
2. The $\twoheadrightarrow$ relation in $l$ is *consistent* with $\rightarrow$ in $c$; that is, for every $n' \overset{t}{\twoheadrightarrow}_l n$ in $S_l$ there is a path $n' = n_1, n_2, \ldots, n_k = n$ in $c$ such that
   (a) $n'$ sends a message $m$ such that $t \Subset m$, and
   (b) there is no $n''$ in $c$ that sends a message $m'$ such that $t \Subset m'$, and there is a path from $n''$ to $n'$.

   Intuitively, the term $t$ must *originate* in $n'$ and be forwarded to $n$.

Since the $\twoheadrightarrow_l$ is consistent with the $\rightarrow$ relation in any bundle $c \in \Psi(l)$, we can define the $\preceq_l$ relation between nodes in a state $l$ as the reflexive and transitive closure of $(\twoheadrightarrow_l \cup \Rightarrow)$, and it will be consistent with the $\preceq_c$ relation for any $c \in \Psi(l)$.

**Definition 4.1.** Semantics of the sequent $l \vdash \Delta$ (introduced in section 4.1) is defined as follows:

$$[\![ l \vdash_P \Delta ]\!] \equiv \forall C \in \Psi(l).\ \Delta \cap C \neq \emptyset. \tag{4.4}$$

**Proposition 4.2.** *If $G$ is empty in a state $l = \langle S, G, \twoheadrightarrow \rangle$, then there exists a bundle $C$ such that $C \in \Psi(l)$, and $S$ and $C$ contain the same strands except that $C$ might contain some additional penetrator strands of type* R, V, *or* T. *(Note, that $S$ may already contain some penetrator strands.)*

*Proof sketch.* Let $h$ be the graph which contains $S$ and the edges defined by the relation "$\twoheadrightarrow$". For each $n_1 \twoheadrightarrow n_2$ in $h$, we can add new penetrator strands of type $\mathrm{R}$, $\mathrm{V}$ and $\mathrm{T}$ and the correspondent "$\rightarrow$" edges, such that we can find a path from $n_1$ to $n_2$ connected by a sequence of edges of "$\rightarrow$" and "$\Rightarrow$". We then eliminate the edge $n_1 \twoheadrightarrow n_2$. After we remove all the "$\twoheadrightarrow$" edges in this way, we get a graph $h'$ which only contains edges of"$\rightarrow$" and "$\Rightarrow$". It is easy to prove that $h'$ is a bundle. Since we only add penetrator strands of type $\mathrm{R}$, $\mathrm{V}$ or $\mathrm{T}$ in the transformation, $h'$ and $S$ have the same regular strands. $\square$

## 4.3 The Proof System and Proof Search Procedure

The skeleton of our verification algorithm is an automatic proof search procedure for the initial sequent $P;\, \Gamma \vdash \Delta$ (Formula 4.2). Our proof system consists of three inference rules which we discuss in the rest of this subsection together with the actual proof search algorithm.

### 4.3.1 Inference Rules

**Soundness and Invertibility.** An inference rule is called *invertible* if, whenever the conclusion is provable, then all of the premisses are provable. To simplify the presentation, we use an equivalent dual definition of the invertibility: whenever one of the premisses of a rule is proven false, its conclusion is also false.

The invertibility property is useful for disproving a sequent and providing a *counterexample* — a successful attack on the protocol. It also allows to search for the proof without backtracking. In the following, we prove that each inference rule is *sound* and *invertible*.

**Init rule.**

In the first step of the proof search we convert an initial sequent $P;\, \Gamma \vdash \Delta$ (Formula 4.2) to another type of sequent $l \vdash \Delta$, where $l$ is a *state*:

$$\frac{l_0(P,\, \Gamma) \vdash \Delta}{P;\, \Gamma \vdash \Delta}\,\mathsf{init}, \tag{4.5}$$

The *initial state* $l_0(P,\, \Gamma)$ specifies that any bundle it represents must satisfy the protocol $P$ and contain all the strands from $\Gamma$. Let $S_\Gamma$ be the smallest semi-bundle that contains $\Gamma$; that is, $S_\Gamma$ is the backward closure of $\Gamma$ over the $\Rightarrow$ relation. Then the initial state is $l_0(P,\, \Gamma) = \langle S_\Gamma,\, G_\Gamma,\, \emptyset\rangle$, where the set of unbound goals $G_\Gamma$ is computed from $S_\Gamma$. We do not bind any goals in the initial state, therefore the $\twoheadrightarrow$ relation is empty.

From the construction of the initial state $l_0 = l_0(P,\, \Gamma)$ we can see that it satisfies the following properties:

$$\begin{array}{l} \Gamma \subseteq l_0 \\ \forall C.\, \Gamma \subseteq C \implies C \in \Psi(l_0). \end{array}$$

It is easily to see that the $\mathsf{init}$ rule is sound and invertible.

**Final rule.**

A sequent $l \vdash \Delta$ is called a *leaf sequent* if $\Delta \cap l \neq \emptyset$. When $l \vdash \Delta$ is a leaf sequent, we apply the final inference rule:

$$\frac{\Delta \cap l \neq \emptyset}{l \vdash \Delta}\ \mathsf{final}, \tag{4.6}$$

The invertibility of this rule is trivial, and the soundness follows from the fact that any bundle $C \in \Psi(l)$ is a supergraph of $S_l$, thus, $\Delta$ has a non-empty intersection with any $C \in \Psi(l)$.

If the final rule does not apply and $G = \emptyset$, then the current sequent $l \vdash \Delta$ is considered proven false, or *refuted*. This follows directly from the Proposition 4.2. Recall that $\Delta$ contains only regular strands. By Proposition 4.2 there exists a bundle $C \in \Psi(l)$ such that for any regular strand $s$, $s \in S_l$ if and only if $s \in C$. Since no single strand from $\Delta$ is in $S_l$, we conclude that it cannot be in $C$ either, which means $\exists C \in \Psi(l). C \cap \Delta = \emptyset$. Therefore, $l \vdash \Delta$ is false. Since all of our inference rules are invertible, this means that the original sequent is also false, and the refuted sequent represents a *counterexample*, or a *successful attack* on the protocol.

**Split rule.**

When the final rule does not apply and $G \neq \emptyset$, we apply the split rule:

$$\frac{l_1 \vdash \Delta \quad \cdots \quad l_n \vdash \Delta, \quad \text{where } \{l_1, \ldots, l_n\} = \mathcal{F}(l)}{l \vdash \Delta}\ \mathsf{split}. \tag{4.7}$$

This rule splits the state $l$ in the current sequent into several *next states* $l_1, \ldots, l_n$ using the *next state function* $\mathcal{F}$. This yields a set of new sequents which are then proven separately. The next state function $\mathcal{F}(l)$ is explained in more detail later in Section 4.4. Note that $\mathcal{F}(l)$ could be empty; in this case the split rule successfully finishes the proof of the sequent. When $\mathcal{F}(l) = \emptyset$, the state $l$ contains a contradiction, so its bundle set is empty, and, therefore, the sequent is vacuously true. We will return to this special case in Section 4.5 when we discuss some optimizations to the proof search algorithm.

The split rule is sound and invertible when $\mathcal{F}$ is *complete-inclusive*.

**Definition 4.3.** We say that $\mathcal{F}$ is *complete-inclusive* if for any state $l$ it satisfies the following properties:

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

The proof of soundness for the split rule can be easily derived from the semantics of the sequent when the function $\mathcal{F}$ is complete-inclusive.

In addition, we prove that the split rule is *invertible* for a complete-inclusive $\mathcal{F}$. That is, if any of the assumption sequents $l_i \vdash \Delta$ is proven false, then the conclusion $l \vdash \Delta$ is also false. The idea behind the proof is the following. A sequent $l_i \vdash \Delta$ is false implies that there exists a bundle $C$ from $\Psi(l_i)$ which has no common strands with $\Delta$. Since $\Psi(l_i) \subseteq \Psi(l)$, then $C \in \Psi(l)$, and therefore, $l \vdash \Delta$ is also false.

### 4.3.2 The Proof Search Procedure

The proof search procedure is very simple. At the beginning we transform the initial sequent $P; \Gamma \vdash \Delta$ into our "working" sequent $l_0 \vdash \Delta$ with the init rule. Then we iteratively apply the following procedure to the current sequent. First we try to apply the $\mathsf{final}$ rule, and if it applies, the current sequent is proven. Otherwise we check if the set of unbound goals in the current sequent is empty ($G = \emptyset$), and if it is, the current sequent is proven false. Since all the rules in our proof system are invertible, the original sequent is also false, and the proof search terminates with a failure, returning the current sequent as a counterexample.

If the $\mathsf{final}$ rule does not apply and $G \neq \emptyset$, then we apply the $\mathsf{split}$ rule, and if it generates any new sequents, continue the proof search for each of the new sequents.

Since our proof system is so simple and all the inference rules are invertible, there is no need to backtrack. In the implementation we also apply several optimizations to the straightforward proof search procedure. We perform a *breadth-first search* from the initial sequent and at each step optionally merge the identical sequents that are generated in different branches of the proof tree. Thus, we effectively construct a *proof DAG*.

Now we state the correctness of the proof search procedure as a theorem.

**Theorem 4.4.** *Let $P; \Gamma \vdash \Delta$ be an initial sequent, and $\mathcal{F}$ be a complete-inclusive next state function. If the proof search procedure described in this section terminates with a complete proof, then the initial sequent is true (soundness); if it refutes a sequent anywhere in the proof, then the initial sequent is false (invertibility).*

The proof is a straightforward structural induction over the proof tree that relies on the soundness and invertibility of each inference rule.

## 4.4 The Next State Function

When the current sequent $l \vdash \Delta$ cannot be proven or refuted in one step, which means the set of unbound goals $G$ in the current sequent is non-empty, we split the current state by binding one of the goals in $G$. We call the splitting procedure *the next state function* $\mathcal{F}$, which is the basis of the $\mathsf{split}$ rule. $\mathcal{F}$ picks a goal $g \in G$ and binds it in all possible ways, creating the set of next states with refined $\twoheadrightarrow$ relation and possibly new strands in the semi-bundle. Note that the order of picking goals *does* affect the efficiency of the algorithm. However, our experience shows that a few simple built-in heuristics work well.

The set of unbound goals in each new state, $G_{l_i}$, might not be smaller than $G_l$, since the newly introduced strands may have new unbound goals; in fact, $G_{l_i}$ may even grow in size. However, the bundle set for each new state does not increase, and usually decreases, since we introduce a new property for the bundles to satisfy: the new goal binding.

If there is no possible way of binding the chosen goal $g$, then the state $l$ is *contradictory* and $\mathcal{F}(l)$ returns an empty set.

We now define $\mathcal{F}$ formally and show that it is complete-inclusive.

**Definition 4.5.** A *position* is a pair $[r, i]$, where $r$ is a role (a parameterized strand) of either a legitimate principal from the given protocol $P$ or a penetrator, and $i$ is an index of a node in $r$. In other words, a position specifies a particular node in a particular role.

**Definition 4.6.** A substitution $\sigma$ is called a *unifier* of a term $t$ and a position $[r, i]$ if the corresponding node $n = \langle r, i \rangle$ in $r$ has a form $n = \langle +t'' \rangle$, and $\sigma$ is a unifier of $t$ and $t'$ for some

$t' \Subset t''$; that is $\sigma(t) = \sigma(t')$. In other words, the node $\sigma(n) = \sigma(\langle r, i\rangle)$ represents sending a term containing $\sigma(t)$.

**Definition 4.7.** A substitution $\sigma$ is a *most general unifier* (MGU) of $t$ and $[r, i]$ if it is a unifier, and for any other unifier $\sigma'$ of $t$ and $[r, i]$ there exists a substitution $\gamma$ such that $\sigma' = \sigma \circ \gamma$.

For a protocol $P$ and a term $t$ we define a *set of unifiers* $U_P(t)$ as follows:

$$U_P(t) = \left\{ \ ([r, i],\, \sigma) \ \middle|\ \sigma \text{ is a MGU of } t \text{ and } [r, i] \text{ for some } [r, i] \in P \cup \Pi \ \right\},$$

where $\Pi$ is the set of penetrator roles (see Section 2.5). The set $U_P(t)$ is always finite, since any protocol has only a finite number of both regular and penetrator roles, and each role has finitely many nodes.

To compute the set of next states $\mathcal{F}(l)$ for a state $l = \langle S,\, G,\, \twoheadrightarrow\rangle$ in a protocol $P$, we first pick an unbound goal $g \in G_l$. We then compute the set $U_P(t)$ for the term $t$ and for each element $u = ([r, i],\, \sigma) \in U_P(t)$ construct the *set of next states* $L'_u$ as follows.

- Let $s_u = (\Rightarrow^{-1})^*[\sigma(\langle r,\, i\rangle)]$ be a (possibly partial) strand that ends with the node $\sigma(\langle r,\, i\rangle)$ and is backward closed under the $\Rightarrow$ relation; that is, it consists of a node $\sigma(\langle r,\, i\rangle)$ and all the preceding nodes in the strand $\sigma(r)$.
- For any strand $s \in S$ from the original state $l$ we say that $\sigma(s)$ is a (possibly partial or extended) *instance* of $s_u$ if there exists a substitution $\gamma_s$ such that $\gamma_s(s_u) \cap \sigma(s) \neq \emptyset$, which means $\sigma(s)$ and $\gamma_s(s_u)$ have common nodes.
  For each strand $s \in S$ from $l$, if $\sigma(s)$ is an instance of $s_u$ with a substitution $\gamma_s$, then we construct a *new state* $l' = \langle S',\, G',\, \twoheadrightarrow'\rangle$ and include it into $L'_u$, where
  1. $S' = \sigma(S) \cup \{\gamma_s(s_u)\}$;
  2. $\twoheadrightarrow' = \sigma(\twoheadrightarrow) \cup \{\langle s',\, i\rangle \overset{\sigma(t)}{\twoheadrightarrow} \sigma(g)\}$), where $s' = \sigma(s) \cup \gamma_s(s_u)$
     (the goal $g$ is bound by the $i$-th node in the strand $s'$); and
  3. $G'$ is updated in accordance with $S'$ and $\twoheadrightarrow'$.
- We also construct an additional next state $l'' = \langle S'',\, G'',\, \twoheadrightarrow''\rangle$ by adding $s_u$ as a *new strand*, such that
  1. $S'' = \sigma(S) \cup s_u$;
  2. $\twoheadrightarrow'' = \sigma(\twoheadrightarrow) \cup \{\langle s_u,\, i\rangle \overset{\sigma(t)}{\twoheadrightarrow} \sigma(g)\}$)
     (the goal $g$ is bound by the last node of $s_u$); and
  3. $G''$ is updated in accordance with $S''$ and $\twoheadrightarrow''$.

The set of all next states is defined as $L' = \bigcup_{u \in U_P(t)} L'_u$. Each next state $l' \in L'$ is then analyzed for being *valid*, and the final result of $\mathcal{F}(l)$ is the set of all valid next states from $L'$.

Checking for validity of $l'$ involves several steps.

A state $l'$ is considered *invalid* if one of the following conditions holds:

- The $\preceq$-minimal relationship is not satisfied in the goal-binding. For example, there exists a positive (sending) node $n_1$ where $t$ binds to, and another node $n_2$, where $t \Subset \mathsf{term}(n_2)$ and $n_2 \preceq n_1$.

- There is a cycle in the resulting state over $\twoheadrightarrow$ and $\Rightarrow$ edges;
- The "unique origination" property of certain terms is not satisfied. For example, when a nonce $N_a$ is specified as "uniquely originated" on a particular role and a node $n_1$, and if $N_a \sqsubseteq \mathsf{term}(n_2)$ and $n_2 \preceq n_1$ in $l$, then the state is considered invalid.

Otherwise $l'$ is considered *valid*.

Notice that terms in sending nodes of penetrator roles $\mathrm{R}$, $\mathrm{T}$, and $\mathrm{V}$ are not $\preceq$-minimal, as required by the goal-binding, and the role $\mathrm{F}$ does not have sending nodes at all. If any of these roles are used to construct an element in $U_P(t)$, the corresponding new states will be invalid. Therefore, we can reduce the penetrator roles $\Pi$ in $U_P(t)$ definition to only four roles $\mathrm{M}$, $\mathrm{K}$, $\mathrm{E}$, and $\mathrm{D}$.

Since we only add (and never delete) nodes and strands to the next states, and since the next state transition covers all possibilities of binding a goal, and $L'$ is finite, we can see that $\mathcal{F}$ is *complete-inclusive*. The proof of this is straightforward and we omit it in this paper.

Notice that due to the most general substitution in goal binding, the nodes and strands in a state may contain free variables. This often allows us to prove a sequent with the minimal possible instantiation of the terms in each state and greatly helps to reduce the search space.

## 4.5 Optimization: Contradiction Analysis

If $\mathcal{F}(l)$ is empty for a state $l$ in the current sequent $l \vdash \Delta$, we say that $l$ is *contradictory*. This means that $\Psi(l) = \emptyset$, usually because $l$ contains unsolvable goals or its graph is cyclic, and therefore, the current sequent is vacuously true. When $l$ is contradictory, the $\mathsf{split}$ rule takes its extreme form and completes the proof of the current sequent:

$$\frac{\mathcal{F}(l) = \emptyset}{l \vdash \Delta}\ \mathsf{split}.$$

In our algorithm we can also easily and systematically incorporate results from theorem proving by using *pruning theorems*, which can prove that $l$ is contradictory early in the proof, thus, "pruning" the search space in the current branch of the proof tree. For practical purposes, we require that pruning theorems are expressed as computable predicates $\theta$ such that if $\theta(l)$ is true, then the state $l$ is contradictory. This allows us to introduce a new (admissible) inference rule as a special case of $\mathsf{split}$:

$$\frac{\theta(l) \text{ is true}}{l \vdash \Delta}\ \mathsf{prune}. \tag{4.8}$$

The pruning 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. Athena also provides an easy interface for the expert users who would like to add their own protocol-specific pruning theorems. Although in general, users do not need to input their own pruning theorems, we provide this easy interface to achieve extensibility of the tool, which enables Athena to be used even for exotic protocols and reduce the search space even further. So far, Athena has two built-in pruning theorems,

$$\begin{array}{ccc} & & \dfrac{s : \mathrm{D}[k, h]}{\langle -k \rangle} \\ & & \Downarrow \\ & & \langle -\{h\}_k \rangle \\ & & \Downarrow \\ \langle -t, n \rangle & \twoheadleftarrow & n' : \langle +h \rangle \end{array}$$

Figure 1: Illustration to the state $l$ in Proposition 4.9

described below as Propositions 4.8 and 4.9. These two theorems are shown to be effective and sufficient in almost all of our experiments.

**Proposition 4.8.** *Let $C$ be a bundle, and $k \in K \setminus K_p$ be a key of a legitimate principal. If $k$ never originates in a regular node, then $k \not\sqsubseteq \mathsf{term}(p)$ for any penetrator node $p \in C$.*

That is, when such a key $k$ occurs in a penetrator strand in a state $l$, then $l$ is contradictory. The proof of this proposition is in [36].

If all the encrypted messages specified in a protocol are well-typed, in other words, the protocol does not contain encryptions of terms with free variables which can match terms of an arbitrary type, then we can always compute the maximum nesting depth for the encryption operations in any message generated by a legitimate principal (denote this maximum depth by $E_m$). Then the following proposition holds.

**Proposition 4.9.** *If a state $l$ satisfies the following conditions (see Figure 1):*

1. *A node $n' = \langle +h \rangle \in \mathrm{D}[k,\, h]$ on a penetrator strand of type $\mathrm{D}$ binds some goal $(t, n)$, and*
2. *The nesting depth of encryption in $t$ is greater or equal to $E_m$,*

*then $l$ is contradictory.*

A proof sketch of this proposition can be found in the appendix. Note that in order to apply proposition 4.9, certain typing requirements in the protocol are needed, namely, all the encrypted messages specified in a protocol must be well-typed. For some protocols this might not hold, and in such cases we specify a threshold for the maximum depth of encryption nesting. A similar restriction is used in other model checkers such as FDR, Mur$\phi$, and Brutus. Athena then verifies the correctness of the protocol for an arbitrary configuration, but with the constraints that for any message in the protocol execution, the nesting depth of encryption is no greater than the threshold.

# 5 Experimental Results

We have implemented our verification technique in a tool called *Athena* [33]. The implementation language of Athena is Standard ML of New Jersey [14], which is freely available on both Windows and Linux platforms [1].

An input to Athena consists of a protocol description and a set of security properties in a simple and intuitive input language (and possibly, additional pruning theorems, if the user wants to supply his own pruning theorems). A description of a typical protocol from the literature together with its properties takes about half a page in this language, and can be written in a matter of minutes. The tool takes this input and runs completely automatically. If it terminates, it either finds a proof or a counterexample for each security property. In the latter case, a successful attack is constructed from the counterexample, which is illustrated graphically with the help of a graph drawing package *Dot* [2]. The Dot package is freely available on Windows, Linux and many other platforms [3].

The latest news on the development of Athena can be found at [35].

We have run several batches of experiments on a Pentium 133MHz PC with 32 MB RAM. The first batch is the analysis of various security properties of protocols from the collection by Clark and Jacob [7]. We have used Athena to check about 30 protocols from [7], and almost all of them have running time less than half a second. For example, for the Otway-Rees protocol [28], we have checked six formulas describing authentication properties and obtained the same results as in [12]. The longest execution time for a single formula is 0.38 second, exploring 23 states. On average, each formula takes 0.16 seconds and requires less than 20 states.

Another batch consists of 1641 asymmetric two party authentication protocols constructed by an automatic protocol generator, which is taken from a case study of an approach to automatic security protocol generation [32]. These protocols are already filtered by an attacker filter which eliminates flawed protocols suffering from simple impersonation and replay attacks. Athena analyzes all of the 1641 protocols in 103.8 seconds, with the average of 63.5 milliseconds per protocol, and reports 120 protocols to be correct in terms of mutual authentication.

We have also experimented with protocols for symmetric two party authentication and key distribution with a trusted third party generated by the automatic protocol generator with different sets of security properties [34]. The protocol generator output more than $11,000$ candidate protocols. Athena took less than two hours to analyze all of them.

Athena can be easily extended with new cryptographic primitives. In fact, adding hash functions and MAC functions only take a few hours of modifying the code from start to finish. In most of our experiments, Athena only needs to search from a dozen to a couple of hundreds of states. When Athena terminates with a proof which contains only a small state space, this automatically indicates that for a given protocol, it is sufficient to just check the correctness of the protocol with a small number of concurrent sessions. This is a side product of the automatic proof procedure in Athena and no manual proofs are involved.

# 6 Conclusion

We have presented a new efficient automatic verification technique, Athena, for the analysis of security protocols. It is based on our extension to the Strand Space Model, combined with techniques from both Model Checking and Theorem Proving. It uses a simple logic to specify various security properties and has an automatic proof search procedure for checking the validity of the well-formed formulas in the logic. When Athena terminates, it can either provide a counterexample if the formula does not hold, or generate a proof showing the correctness of the security protocol with arbitrary number of concurrent runs. Typical benchmark protocols from the literature usually take Athena less than a second to find a correctness proof or an attack on the protocol.

Athena exploits several different techniques that enable it to analyze infinite sets of protocol runs and achieve such efficiency. Unlike previous tools, Athena uses the extended Strand Space Model to represent protocol executions. The semi-bundle representation and the goal binding relation comprise a state structure which provides a very compact encoding for (usually infinite) sets of protocol runs. For example, the state structure eliminates the problem of infinite forwarding of messages by using the $\preceq$-minimal requirement in the goal binding. Because the state space in Athena is not an asynchronous composition of local processes, Athena naturally avoids the state space explosion problem caused by the asynchronous composition. The use of free variables and symbolic techniques allows each state to represent an infinite number of concrete protocol executions and reduce the state space explosion problem caused by symmetry redundancy.

As a new approach for security protocol analysis, Athena is still at a very young stage. We look forward to extending the approach in other directions.

# Acknowledgements

We would like to thank the research group of Edmund Clarke for valuable help. We would also like to thank Joshua Guttman, Jonathan Herzog, Jonathan Millen, John Mitchell, George Necula, Sylvan Pinsky, Javier Thayer, Doug Tygar and Jeanette Wing for their encouragement and discussion.

# References

[1] http://cm.bell-labs.com/cm/cs/what/smlnj/.

[2] http://www.research.att.com/sw/tools/graphviz/.

[3] http://www.research.att.com/sw/tools/graphviz/download.html.

[4] Stephen Brackin. Automatic formal analyses of cryptographic protocols. In *Proceedings of the 19th National Conference on Information Systems Security*, 1996.

[5] Stephen Brackin. Automatic formal analysis of two large commercial protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

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

[7] John Clark and Jeremy Jacob. A survey of authentication protocol literature: Version 1.0, November 1997.

[8] Ed Clarke, Somesh Jha, and Will Marrero. Using partial order and symmetry reductions in security protocol verification. Unpublished, 1999.

[9] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[10] E.M. 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.

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

[12] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *In Proceedings of 1998 Computer Security Foundations Workshop*, June 1998.

[13] 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.

[14] Michael R. Hansen and Hans Rischel. *Introduction to Programming using SML*. Addison-Wesley, 1999.

[15] N. Heintze, D. Tygar, J. Wing, and H. Wong. Model checking electronic commerce protocols. In *Proceedings of the USENIX 1996 Workshop on Electronic Commerce*, pages 146–164, 1996.

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

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

[18] 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.

[19] 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.

[20] Gavin Lowe. Towards a completeness result for model checking security protocols. *Journal of Computer Security*, 1999.

[21] A. Maneki. Honest functions and their applications of cryptographic protocols. In *the 12th IEEE Computer Security Foundation Workshop*, 1999.

[22] 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.

[23] 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.

[24] C. Meadows. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In *Proceedings of ESORICS*. Springer-Verlag, 1996.

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

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

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

[28] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21, 1987.

[29] 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.

[30] Doron Peled. All from one, one for all: on model checking using representatives. In Costas 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.

[31] Doron Peled. Combining partial order reductions with on-the-fly model-checking. In David 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.

[32] Adrian Perrig and Dawn Song. An initial approach to automatic generation of security protocols. In Proceedings of NDSS'00 (Network and Distributed System Security Symposium), 2000.

[33] Dawn Song. Athena: An automatic checker for security protocol analysis. In *Proceedings of the 12th Computer Science Foundation Workshop*, 1999.

[34] Dawn Song and Adrian Perrig. Looking for a diamond in a dessert – extending automatic protocol generation to three-party authentication and key distribution protocols. In *To Appear in the proceeding of IEEE Computer Security Foundation Workshop (CSFW'2000)*, July, 2000.

[35] Dawn Xiaodong Song. `http://www.cs.berkeley.edu/~dawnsong` and `http://www.cs.cmu.edu/~skyxd`.

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

[37] Jeannette M. Wing. A symbiotic relationship between formal methods and security. In *Workshops on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solution*. ONR and NSF, 1998.

[38] 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.

# A Proof of the Pruning Theorem

**Proposition 4.9.** *If a state $l$ satisfies the following conditions (see Figure 1):*

1. *A node $n' = \langle +h_1 \rangle \in \mathrm{D}[k_1,\ h_1]$ on a penetrator strand of type $\mathrm{D}$ binds some goal $(t, n)$, and*
2. *The nesting depth of encryption in $t$ is greater or equal to $E_m$,*

*then $l$ is contradictory.*

This proposition is used to eliminate an infinite expansion of type $\mathrm{D}$ strands.

*Proof Sketch.* Since $t$ contains a number of nested encryptions which is greater than or equal to $E_m$, the goal term $\{h_1\}_{k_1}$ can not be bound to any regular node. Therefore, it can only be bound to either a type $\mathrm{D}$ or a type $\mathrm{E}$ strand of the penetrator. If it is bound to a type $\mathrm{E}$ strand, there will be either a cycle in the graph, or the $\preceq$-minimal condition for $h_1$ will not be satisfied. If it is bound to a type $\mathrm{D}$ strand $s_2$, where $s_2 = \mathrm{D}[k_2, h_2]$ and $\{h_1\}_{k_1} \Subset h_2$, as shown in the graph, then the same argument for $\mathrm{D}[k_1, h_1]$ applies to the strand $\mathrm{D}[k_2, h_2]$. Thus, we derive a chain of type $\mathrm{D}$ strands. Since for any protocol run, a penetrator can only take finite number of operations, this chain has to stop at a finite length, say at strand $s_u = \mathrm{D}[k_u, h_u]$. Then $\langle -\{h_u\}_{k_u} \rangle$ can only be bound to a type $\mathrm{E}$ strand which will cause either a cycle in the graph, or the $\preceq$-minimal condition for some $h_i$ to be violated. This implies that $\Psi(l) = \emptyset$, therefore, $l$ is contradictory. $\square$

$$
\begin{array}{ccccccc}
 & & & & \underline{s_2 : \mathrm{D}[k_2,h_2]} & & \\
 & & & & \langle -k_2 \rangle & & \\
 & & \underline{s_1 : \mathrm{D}[k_1,h_1]} & & \Downarrow & & \\
 & & \langle -k_1 \rangle & & \langle -\{h_2\}_{k_2} \rangle & \twoheadleftarrow & \cdots \\
 & & \Downarrow & & \Downarrow & & \\
 & & \langle -\{h_1\}_{k_1} \rangle & \twoheadleftarrow & n_2 : \langle +h_2 \rangle & & \\
 & & \Downarrow & & & & \\
\langle -t,n \rangle & \twoheadleftarrow & n_1 : \langle +h_1 \rangle & & & &
\end{array}
$$