Under consideration for publication in Formal Aspects of Computing

An Inductive Approach to Strand Spaces

Yongjian Li

Chinese Academy of Sciences,
Institute of Software,
The State Key Laboratory of Computer Science &
The State Key Laboratory of Information Security

Jun Pang

Université du Luxembourg,
Faculté des Sciences de la Technologie et de la Communication,
Computer Science and Communications

Abstract. In this paper, we develop an inductive approach to strand spaces, by introducing an inductive definition for bundles. This definition provides us not only a constructive illustration for bundles, but also an effective and rigorous technique of rule induction to reason about properties of bundles. With this induction principle, we can prove that our bundle model is sound in the sense that a bundle is a causally well-founded graph. This approach also gives an alternative to rigorously prove a generalized version of authentication tests. To illustrate the applicability of our approach, we have performed case studies on verification of secrecy and authentication properties of the Needham-Schroeder-Lowe and Otway-Rees protocols. Our approach has been mechanized using Isabelle/HOL.

Keywords: Strand spaces, security protocols, theorem proving, inductive methods.

1. Introduction

Security protocols are designed to ensure security properties, including authentication, secrecy and key distribution, even with the presence of a penetrator who can perform malicious actions. A security protocol

Correspondence and offprint requests to: Yongjian Li, The State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, P.O.Box 8717, Beijing, China. e-mail: `lyj238@ios.ac.cn`
Preliminary versions of this work were presented at the 25th IFIP International Conference on Formal Techniques for Networked and Distributed System (FORTE 2005) [Li05] and the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2006) [LP06]. The first author is supported by grants (Nos. 60833001, 60603001, 60721061, 60496321, 60421001, 60973016) from National Natural Science Foundation of China and a grant (No. 2007AA01Z147) from the National High Technology Development 863 Program of China.

contains a list of messages exchanges among two or more agents. These messages are often encrypted using symmetric or asymmetric encryption. However, the design of these protocols is error-prone. Incorrectly designed protocols may become ideal entry points for various attacks. Therefore, we cannot only rely on informal ways of reasoning about their correctness. Following the seminal work of Dolev and Yao [DY83], a lot of research efforts have been devoted to developing formal approaches to security protocol analysis, including the BAN logic [BAN90], model checking [Mil95, Low96a, Low96b, Sch97, MMS97, Mea99], Paulson's inductive method [Pau97, Pau98], and the strand space method [JHG98, JHG99]. Among them, the strand space is one of the most successful and widely used formalisms.

A brief introduction to the strand space method. Within the strand space model, a strand for a legitimate regular agent represents a sequence of actions that the agent would receive or send as part of a run as his/her role of the protocol. A strand element is called node. Nodes can be either positive, representing the transmission of a term, or negative, representing the reception of a term. For the penetrator, the strand represents atomic deductions. More complex deductions can be formed by connecting several penetrator strands. Hence, a strand space is simply a set of strands with a trace mapping. Two kinds of causal relation (arrow), $\rightarrow$ and $\Rightarrow$, are introduced to impose a graphic structure on the nodes of the space. The relation $\preceq$ is defined to be the reflexive and transitive closure of these two arrows, modeling the causal order of the events in the protocol execution. The formal analysis based on strand spaces can be carried on the notion of bundles. A bundle is a causally well-founded set of nodes and the two arrows, which sufficiently formalizes a session of a protocol. In a bundle, it must be ensured that a node is included only if all nodes that proceed it are already included. For the strand corresponding to an agent in a given protocol run, we construct all possible bundles containing nodes of the strand. In fact, this set of bundles encodes all possible interactions of the environment with that agent in the run. Typically, for the protocol to be correct, each such bundle must contain one strand of each legitimate agent apparently participating this session, all agreeing on names of agents, nonces, and session keys. Penetrator strands may also be entangled in a bundle, even in a correct protocol, but they should not prevent legitimate parties from agreeing on the data values, or from maintaining the secrecy of the value chosen. The key to this approach is the fact that a bundle forms a finite, well-founded sets under the causal relation $\preceq$, and each non-empty subset of the bundle has a $\preceq$-minimal element.

In [GJ01, GJ02], a powerful idea, authentication tests, is introduced. It is basically a formalization of the basic challenge-response style primitive which is a building block for many protocols. An agent transmits a so-called *test component*, and later receives back another term which is in some transformed form of the component, then only a regular agent, but not the penetrator, can have transformed it. In favorable circumstances, it can only be one regular agent, the intended one, who has thereby been authenticated. The authentication tests themselves are easy to apply, but the proofs justifying them are very complicated. Special research efforts should be made to make the proofs precise and concise.

Although strand spaces provide us an intuitive and powerful framework to analyze security protocols, it seems that the mechanics of the proof tend to be quite intricate and not necessarily easy to be formalized. To the best of our knowledge, the strand space method has not been formalized in a theorem prover yet. There is still a deep gap between a pen-and-paper strand space theory and a mechanized theory library. Our work is essential not only to bridge this gap, but also to provide a mechanized proving methodology for security protocols on the groundwork of strand space theory.

Structure of the paper. The remainder of this paper is organized as follows: Section 2 provides a preliminary introduction for the classic strand space theory. Section 3 summarizes the difficulties in formalizing strand space theory and briefly introduces our motivations and contributions in this work. Section 4 shows our formalized strand space theory. Section 5 shows how to formalize and prove protocol properties based on our mechanized strand space theory, using the Needham-Schroeder-Lowe (NSL) protocol [Low96a] and the Otway-Rees protocol [OR87] as illustrative examples. We discuss related work in Section 6. Section 7 concludes the paper.

Presentation of the paper. As mentioned before, our work involves both improvements on the strand space theory itself and the formalization of the theory in a theorem prover in order to provide mechanical support for security protocol analysis using strand spaces. Because formalization is one of our main objectives in this paper and our implementation is tailored to Isabelle/HOL, we directly use parts of our Isabelle's theories to introduce definitions and lemmas to convey the main idea of the formalization.

Isabelle/HOL has a polymorphic type system as in ML [Pau96]. Type inference eliminates the need to specify types in expressions. Lemmas about lists, sets, etc., are polymorphic, and the prover uses the appropriate types automatically. Besides, a function in Isabelle/HOL syntax is usually defined in a curried form instead of a tupled form, that is, we often use the notation $f\ x\ y$ to stand for $f(x, y)$. The advantage of a curried function is to allow a partial function application [Pau96]. We use the notation $[\![A_1; A_2; ...; A_n]\!] \Longrightarrow B$ to mean that with assumptions $A_1, \ldots, A_n$, we can derive a conclusion B. Given a relation r, we write r^+ for the transitive closure of r, r^* for the transitive and reflexive closure of r, and $x\#xs$ for the list that extends xs by adding x to the front of xs, $[x_1, ..x_n]$ for a list $x_1\#..x_n\#[]$, $xs@ys$ for the result list by concatenating xs with ys, $xs!i$ for the i^{th} element of the list xs (counting from 0 as the first element), $\mathsf{set}\ xs$ for the set of all the elements in xs, $\mathsf{length}\ xs$ for the length of the list xs, and $\mathsf{last}\ xs$ for the last element of the list xs. More information on our choices of notations can be found in the appendix.

2. Preliminaries

2.1. Messages

The set of messages is defined using the following BNF notation:

$$\begin{array}{lllllll} h & ::= & \mathsf{Agent}\ A & \mid & \mathsf{Nonce}\ N & \mid & \mathsf{Key}\ K \quad \mid \\ & & \mathsf{MPair}\ h_1\ h_2 & \mid & \mathsf{Crypt}\ K\ h & & \end{array}$$

where A is an element from a set of agents, N from a set of nonces, and K from a set of keys. Here we use K^{-1} to denote the inverse key of K. $\mathsf{MPair}\ h_1\ h_2$ is called a composed message. $\mathsf{Crypt}\ K\ h$ represents the encryption of message h with K. We use the free encryption assumption, where $\mathsf{Crypt}\ K\ h = \mathsf{Crypt}\ K'\ h'$ if and only if $K = K'$ and $h = h'$. The set of all messages is denoted by $\mathsf{Messages}$. Terms of the form $\mathsf{Agent}\ A$, $\mathsf{Nonce}\ N$, or $\mathsf{Key}\ K$ are said to be atomic. The set of all atomic messages is denoted by Atoms. A message h is a text message if $h \neq \mathsf{Key}\ K$ for any K. The set of all atomic text messages is denoted by T.

In an asymmetric-key protocol model, an agent A has a public key $\mathsf{pubK}\ A$, which is known to all agents, and a private key $\mathsf{priK}\ A$. $\mathsf{pubK}\ A$ is the inverse key of $\mathsf{priK}\ A$ ($(\mathsf{priK}\ A)^{-1} = \mathsf{pubK}\ A$), and vice versa. In a symmetric-key model, each agent A has a symmetric key $\mathsf{shrK}\ A$. The inverse key of $\mathsf{shrK}\ A$ is itself ($(\mathsf{shrK}\ A)^{-1} = \mathsf{shrK}\ A$). We also assume that (1) asymmetric keys and symmetry keys are disjoint; (2) the functions shrK, pubK and priK are injective, e.g., if $\mathsf{shrK}\ A = \mathsf{shrK}\ A'$ then $A = A'$. The public key, private key, and shared key of an agent are long-term becuase the agent holds them forever. In contrast, some keys are created and used only in a session by some agents, and these keys are short-term. In the following, we abbreviate $\mathsf{Crypt}\ K\ h$ as $\{\!|h|\!\}_K$, and $\mathsf{MPair}\ h_1 \ldots \mathsf{MPair}\ h_{n-1}\ h_n$ as $\{\!|h_1, \ldots, h_{n-1}, h_n|\!\}$. Such abbreviations are supported in Isabelle by syntax translation [NPW02]. In order to reduce the number of $\{\!|$ or $|\!\}$ for readability, we abbreviate $\mathsf{Crypt}\ K\ (\mathsf{MPair}\ h_1 \ldots \mathsf{MPair}\ h_{n-1}\ h_n)$ as $\{\!|h_1, \ldots, h_{n-1}, h_n|\!\}_K$ in this paper.

Parts and synth. Two operators parts and synth are inductively defined on a message set H. Their definition is taken from [Pau97, Pau98] and tailored for our purposes. Usually, H contains a penetrator's initial knowledge and all messages sent by regular agents. The set $\mathsf{parts}\ H$ is obtained from H by repeatedly adding the components of compound messages and the bodies of encrypted messages. Formally, $\mathsf{parts}\ H$ is the least set including H and closed under projection and decryption.

$$\frac{g \in H}{g \in \mathsf{parts}\ H} \qquad \frac{\{\!|g|\!\}_K \in \mathsf{parts}\ H}{g \in \mathsf{parts}\ H}$$

$$\frac{\{\!|g, h|\!\} \in \mathsf{parts}\ H}{g \in \mathsf{parts}\ H} \qquad \frac{\{\!|g, h|\!\} \in \mathsf{parts}\ H}{h \in \mathsf{parts}\ H}$$

The parts operator is used to define the subterm relation $\sqsubset$ in strand spaces: $h_1 \sqsubset h_2 \equiv h_1 \in \mathsf{parts}\{h_2\}$. $h_1 \sqsubset h_2$ means that h_1 occurs in h_2. Here K is not regarded as occurring in $\{\!|g|\!\}_K$ unless K is a part of g.

The set $\mathsf{synth}\ H$ models the messages a spy could build up from elements of H by repeatedly adding agent names, forming compound messages and encrypting with keys contained in H. $\mathsf{synth}\ H$ is defined to be the least set that includes H, agents, and is closed under pairing and encryption. In later sections, the synth

operator is used when we describe an invariance of a penetrator strand, which characterizes a penetrator's ability to deduce knowledge from a message set.

$$\mathsf{Agent}\ A \in \mathsf{synth}\ H \qquad \frac{\mathsf{Key}\ K \in H \quad h \in \mathsf{synth}\ H}{\{\!|h|\!\}_K \in \mathsf{synth}\ H}$$

$$\frac{g \in \mathsf{synth}\ H \quad h \in \mathsf{synth}\ H}{\{\!|g, h|\!\} \in \mathsf{synth}\ H} \qquad \frac{g \in H}{g \in \mathsf{synth}\ H}$$

For instance, it is easy to verify that $\mathsf{Nonce}\ N \in \mathsf{parts}\ \{\{\!|\mathsf{Nonce}\ N, \mathsf{Agent}\ A|\!\}_{\mathsf{pubK}\ A}\}$, but $\mathsf{Key}\ (\mathsf{pubK}\ A) \notin \mathsf{parts}\ \{\{\!|\mathsf{Nonce}\ N, \mathsf{Agent}\ A|\!\}_{\mathsf{pubK}\ A}\}$. We can have $\mathsf{Nonce}\ N \sqsubset \{\!|\mathsf{Nonce}\ N, \mathsf{Agent}\ A|\!\}_{\mathsf{pubK}\ A}$, and the spy can generate the term $\{\!|\mathsf{Nonce}\ N, \mathsf{Agent}\ A|\!\}_{\mathsf{pubK}\ A} \in \mathsf{synth}\ \{\mathsf{Nonce}\ N,\ \mathsf{Key}\ (\mathsf{pubK}\ A)\}$.

2.2. Strands and Strand Space

The notions in this section are mainly taken from the paper [JHG98, JHG99], with slight extensions for formalization.

Actions. The set of actions that agents can take during an execution of a protocol include send and receive actions. We denote send and receive actions by a set of two signs $\mathsf{Sign} = \{+, -\}$, respectively.

Events. An event is a pair (σ, t), where $\sigma \in \mathsf{Sign}$ and $t \in \mathsf{Messages}$.

Strands and strand spaces. A protocol defines a sequence of events for each agent's role. A strand represents a sequence of an agent's actions in a particular protocol run, and is an instance of a role. A strand space is a mapping from a strand set Σ to a trace $\mathsf{SP} : \Sigma \Rightarrow (\mathsf{Sign} \times \mathsf{Messages})\ \mathsf{list}$.

- A node is a pair (s, i), with $s \in \Sigma$ and $0 \leq i < \mathsf{length}\ (\mathsf{SP}\ s)$. We use $n \in$ strand s to denote that a node $n = (s, i)$ belongs to the strand s. The set of all nodes is denoted as Domain.
- If $n = (s, i)$ and $(\mathsf{SP}\ s)!i = (\sigma, g)$, then we define $\mathsf{strand}\ n$, $\mathsf{index}\ n$, $\mathsf{term}\ n$ and $\mathsf{sign}\ n$ to be the strand, index, term and sign of the node n respectively, namely $\mathsf{strand}\ n = s$, $\mathsf{index}\ n = i$, $\mathsf{term}\ n = g$ and $\mathsf{sign}\ n = \sigma$. We call a node positive if it has sign $+$, and negative if it has sign $-$.
- If $n, n' \in \mathsf{Domain}$, the relation $n \Rightarrow n'$ holds between nodes n and n' if $n = (s, i)$ and $n' = (s, i+1)$. This represents event occurring on n followed by that occurring on n'.
- If $n, n' \in \mathsf{Domain}$, the relation $n \rightarrow n'$ holds for nodes n and n' if $\mathsf{strand}\ n \neq \mathsf{strand}\ n'$, $\mathsf{term}\ n = \mathsf{term}\ n'$, $\mathsf{sign}\ n = +$ and $\mathsf{sign}\ n' = -$. This represents that n sends a message and n' receives the message. Note that we place an additional restriction on the relation $\rightarrow$ than that in [JHG98, JHG99], we require $\mathsf{strand}\ n \neq \mathsf{strand}\ n'$, i.e., n and n' are in different strands, which means that actions of sending or receiving a message can only occur between different strands.
- A term g originates from a node $n \in \mathsf{Domain}$ iff $\mathsf{sign}\ n = +$ and $g \sqsubset \mathsf{term}\ n$, and whenever n' precedes n on the same strand, $g \not\sqsubset \mathsf{term}\ n'$.
- A term g uniquely originates from node n iff g originates on a unique node n. Nonces and other freshly generated terms are usually uniquely originated.

Bundles. A bundle $b = (N_b, E_b)$ is a finite subgraph of the graph $(\mathsf{Domain}, (\rightarrow \cup \Rightarrow))$, which represents a protocol execution under some configuration. N_b is the set of nodes, and E_b is the set of the edges incident with the nodes in N_b, and the following properties hold:

- b is a finite graph;
- If the sign of a node n is $-$, and $n \in N_b$, then there is a unique positive node n' such that $n' \in N_b$, $n' \rightarrow n$ and $(n', n) \in E_b$;
- If $n' \Rightarrow n$ and $n \in b$, then $n' \in N_b$ and $(n', n) \in E_b$.
- b is acyclic.

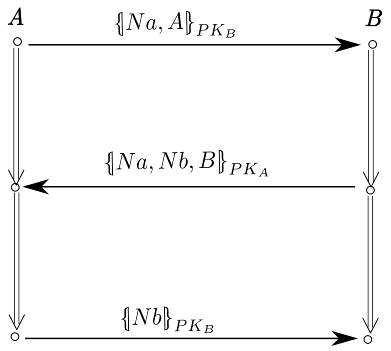

Fig. 1. The Needham-Schroeder-Lowe protocol.

Causal precedence. Let b be a graph, we define $m \prec_b n$ for $(m, n) \in E_b^+$, and $m \preceq_b n$ for $(m, n) \in E_b^*$. $\prec_b$ and $\preceq_b$ represent causal precedence between nodes in b.

From the definition of a bundle, we can derive that it is a casually well-founded graph [JHG98, JHG99].

Lemma 1. For a bundle b, b is casually well-founded graph, and every non-empty subset of the nodes in has $\prec_b$-minimal members.

2.3. Protocol Modeling using Strands

A protocol usually contains several roles, such as initiators, responders and servers. The sequence of actions of each regular agent acting some role in a protocol session is pre-defined by the protocol and represented as a parameterized strand. Parameters usually include agent names and nonces. Informally, we denote a parameter strand acting some role by $\mathsf{role}[\textit{parameter list}]$. The strands of the legitimate agents are referred to as regular strands.

A bundle can also contain penetrator strands. We explain them in more details in the next section. We now use the NSL protocol [Low96a, Low96b] (see Figure 1) as an example to illustrate the modelling strategy using strands. In this figure, we abbreviate $\mathsf{Agent}\ A$ as A, $\mathsf{Nonce}\ Na$ as Na, and $\mathsf{pubK}\ A$ as PK_A.

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

- $\mathsf{NSLInit}[s, \mathsf{Agent}\ A, \mathsf{Agent}\ B, \mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb]$, if the trace of s is $[(+, \{\!|\mathsf{Nonce}\ Na, \mathsf{Agent}\ A|\!\}_{\mathsf{pubK}\ B}), (-, \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{\mathsf{pubK}\ A}), (+, \{\!|\mathsf{Nonce}\ Nb|\!\}_{\mathsf{pubK}\ B})]$.
- $\mathsf{NSLResp}[s, \mathsf{Agent}\ A, \mathsf{Agent}\ B, \mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb]$, if the trace of s is $[(-, \{\!|\mathsf{Nonce}\ Na, \mathsf{Agent}\ A|\!\}_{\mathsf{pubK}\ B}), (+, \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{\mathsf{pubK}\ A}), (-, \{\!|\mathsf{Nonce}\ Nb|\!\}_{\mathsf{pubK}\ B})]$.

2.4. Penetrator

The symbol bad denotes the set of all penetrators. If an agent is not in the set bad, then it is regular. The strands of the penetrators are referred to as penetrator strands, which we show in the following paragraph. If a strand is not a penetrator one, it is referred to as a regular strand. A node is regular if it is at a regular strand.

There is a set of keys known to all penetrators initially, denoted as KP. KP contains the public keys of all agents, the private keys of all penetrators, and the symmetric keys initially shared between the penetrators and the server.

In the classic strand space theory, a penetrator can intercept messages and generate messages that are computable from its initial knowledge and the messages it intercepts. These actions are modeled by a set of penetrator strands, and they represent atomic deductions. More complex deduction actions can be formed by connecting several penetrator strands together. In our extension, we assume that penetrators share their initial knowledge and cooperate each other by composing their strands.

Definition 1. A penetrator' trace relative to KP is one of the following:

- $\mathsf{M}[a]$ (text message): $[(+, a)]$, where $a \in \mathsf{T}$.
- $\mathsf{K}[K']$ (key): $[(+, \mathsf{Key}\ K')]$, where $K' \in \mathsf{KP}$.
- $\mathsf{C}[g, h]$ (concatenation): $[(-, g), (-, h), (+, \{\!|g, h|\!\})]$.
- $\mathsf{S}[g, h]$ (separation): $[(-, \{\!|g, h|\!\}), (+, g), (+, h)]$.
- $\mathsf{E}[h, K]$ (encryption): $[(-, \mathsf{Key}\ K), (-, h), (+, \{\!|h|\!\}_K)]$.
- $\mathsf{D}[h, K]$(decryption): $[(-, \mathsf{Key}\ K^{-1}), (-, \{\!|h|\!\}_K), (+, h)]$.
- $\mathsf{F}[g]$(Flush): $[(-, g)]$.
- $\mathsf{T}[g]$(Tee): $[(-, g)\,, (+, g)\,, (+, g)]$.

Here, $\mathsf{M}[a]$ and $\mathsf{K}[K']$ do not imply that a penetrator can issue any unguessable term which is not in his initial knowledge, such as nonces and session keys. Because when we introduce secrecy or authentication properties about an unguessable term t for all penetrators, we usually assume that t uniquely originates from a regular strand, and this implicitly eliminates the possibility that any penetrator can originate t.

2.5. Specifying Security Properties

In this paper we mainly focus on the authentication and secrecy properties. We use similar ways for representing security properties as in [JHG98, JHG99].

Secrecy. A message g is secret for a protocol if in every bundle b of the protocol the penetrator cannot receive g in cleartext; that is, there is no node n in b such that $\mathsf{term}\ n = g$. Usually g is a long-term key of a regular agent, a nonce, or a session key issued by a server. For a key K, if $\mathsf{Key}\ K$ can be kept secret, then K is safe.

Authentication. Authentication properties are specified as follows: for an agent B (e.g. acting as a responder), for a certain vector of parameters $\overrightarrow{x}$, if each time B completes a run of the protocol as a responder using $\overrightarrow{x}$, supposedly with another agent A, then there is a run of the protocol with A acting as an initiator using $\overrightarrow{x}$, supposedly with B. And this is formalized as follows: there is a responder strand $\mathsf{Resp}[\overrightarrow{x}]$ and the i-th node of the strand is in a bundle b, then there is an initiator strand $\mathsf{Init}[\overrightarrow{x}]$ and some j-th node of the initiator strand is in b.

2.6. Authentication Tests

Consider an agent in a security protocol, it originates and transmits a message containing a new value a. [1] Later on, this agent receives a back in a different cryptographic context associated with a relevant key K. The agent can conclude that some other agent possessing K (or K^{-1}) must have received and transformed the message in which a was emitted. If K (or K^{-1}) is safe, this agent cannot be the penetrator, but a regular agent instead. A transforming edge is the action of changing the cryptographic form in which such a value a occurs. The authentication tests give sufficient conditions for transforming edges being the work of regular agents.

Outgoing tests. A uniquely originating value a may be transmitted only in an encrypted form $\{\!|h|\!\}_K$ from a node n, where K^{-1} is safe. If it is not received in the same context $\{\!|h|\!\}_K$, then a regular agent must have been responsible for the first time it appears in a different context. Figure 2 presents Proposition 19 of [GJ01] in a simplified form. We call the edge $n \Rightarrow^+ n'$ an *outgoing test* for a in $\{\!|h|\!\}_K$ because the encrypted unit goes out and the edge $m \Rightarrow^+ m'$ is a transforming edge for a.

[1] It is usually a nonce or session key.

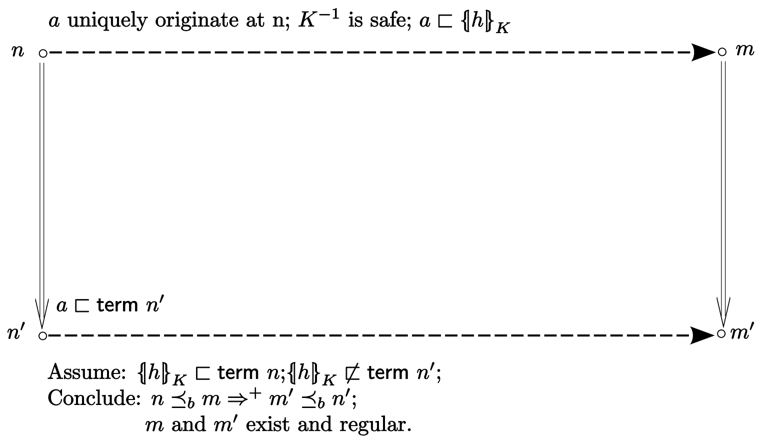
a uniquely originate at n; K^{-1} is safe; $a \sqsubset \{\!|h|\!\}_K$
n
m
$a \sqsubset$ term n'
n'
m'
Assume: $\{\!|h|\!\}_K \sqsubset$ term n;$\{\!|h|\!\}_K \not\sqsubset$ term n';
Conclude: $n \preceq_b m \Rightarrow^+ m' \preceq_b n'$;
m and m' exist and regular.

Fig. 2. Outgoing authentication tests.

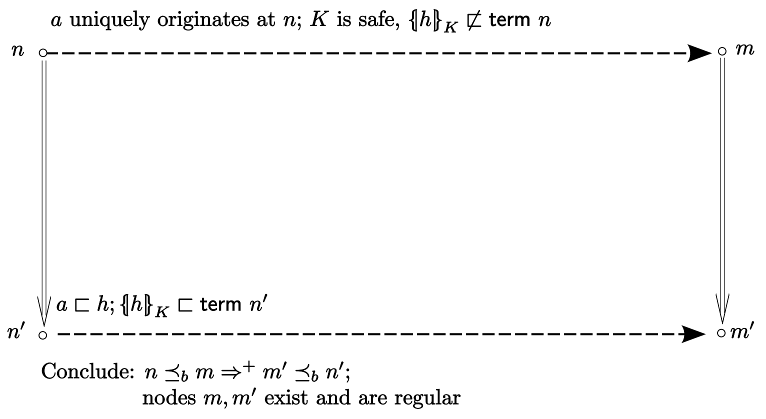
a uniquely originates at n; K is safe, $\{\!|h|\!\}_K \not\sqsubset$ term n
n
m
$a \sqsubset h$; $\{\!|h|\!\}_K \sqsubset$ term n'
n'
m'
Conclude: $n \preceq_b m \Rightarrow^+ m' \preceq_b n'$;
nodes m, m' exist and are regular

Fig. 3. Incoming authentication tests.

Incoming tests. Let a received within an encrypted form $\{\!|h|\!\}_K$ at node n'. If a was not sent in that context by another node n and K is safe, then a regular agent, which strand m stands for, must have been responsible when a entered into this context. We refer to the edge $n \Rightarrow^+ n'$ an *incoming test* because the encrypted unit comes in, as shown in Figure 3 representing Proposition 20 of [GJ01]. In public key cryptography, K serves as a signature key.

Unsolicited tests. A third, related but weaker, type of test is the unsolicited test. If a term $\{\!|h|\!\}_K$ is received, and K is safe, then $\{\!|h|\!\}_K$ originated on some regular strand. After all, it originated somewhere, and that cannot have been a penetrator strand if K is safe. Here we know only that the regular node originating $\{\!|h|\!\}_K$ is before the node on which it is received.

3. Motivations and Contributions

Based on the introduction of preliminary knowledge of strand spaces, we analyze the reason why it is so difficult to formalize strand spaces. In our opinion, the main crux lies in that the theory is proposed mainly for paper proof-reading without having mechanical support in mind.

Firstly, as the cornerstone in the strand space theory, we believe the definition of bundle should be refined. In its current form, the notion of bundle is just a sketchy property description of a graph about a protocol session, that is, a bundle is a causally well-founded finite graph. However, it does not tell us how this graph is constructed inductively. Such a sketchy definition introduces many meaningless instances for

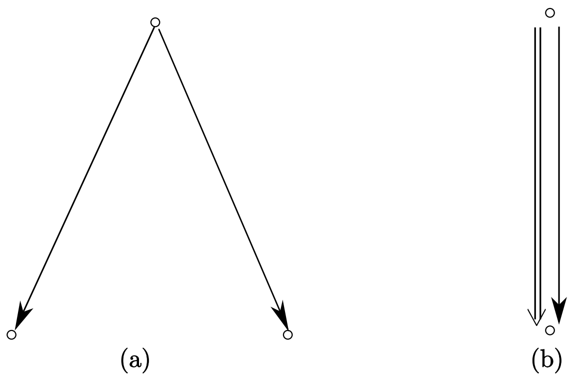

Fig. 4. Useless bundles.

protocol analysis. For example, bundles (a) and (b) in Figure 4 make no sense for any realistic protocol model, as bundle (a) allows an agent to send a message to two different agents at the same time and bundle (b) shows that an agent can send a message to himself ($n_1 \Rightarrow n_2$ requires that $\mathsf{strand}\ n_1 = \mathsf{strand}\ n_1$). We believe that we need a better definition of bundles to formalize the operational semantics of protocol steps.

Secondly, many definitions and proofs of the strand space method are informal and complicated, and it is far from being mechanized in a theorem prover. For example, the subterm relation [GJ02] is defined as follows: "A term t is a subterm of another term t', written $t \sqsubset t'$, if starting with t we can reach t' by repeatedly concatenating with arbitrary terms and encrypting with arbitrary keys". The definition of path [GJ02] is as follows: "A path p through a bundle $\mathfrak{C}$ is any finite sequence of nodes and edges $n_1 \longmapsto n_2 \longmapsto \ldots \longmapsto n_k$." Such definitions are too informal to be directly formalized. Later in this paper, we show that adopting rule induction gives a satisfactory definition for such relation, which is fully supported by the theorem prover Isabelle/HOL.

Besides, proofs of authentication tests are extremely complicated such that it is hard to follow even in paper proofs. The authors devoted almost 30 pages to the proofs of authentication tests in [GJ02]. The authentication tests guarantee the existence of the transforming edges in a bundle, which is done by a regular agent. However, they cannot directly prove the result for a general bundle. They have to propose the concepts of bundle equivalence, normal bundles, efficient bundles. The equivalence between two bundles means that they have the same regular nodes and the same uniquely originating message assumptions. Both normal and efficient bundles place restrictions on the behaviors of the penetrators. The former eliminates the redundancies of behaviors of the penetrators, while the latter has restriction on the way in which a penetrator gets a component from a regular node. Their proof is mainly composed of two parts: firstly they need prove that there exists an equivalent normal and efficient bundle for any bundle, and secondly that authentication tests guarantee the existence of the transforming edges hold for any normal efficient bundle. With the above two results, they prove that authentication tests also guarantee the existence of the transforming edges for a general bundle, because there is an equivalent normal and efficient bundle for the general bundle and they have the same behaviors on the regular nodes.

Unfortunately the definitions of normal and efficient bundles are rather complex, it is tedious to specify the graphical operations to construct an equivalent normal and efficient bundle from a general bundle. The corresponding proof of the existence of an equivalent normal and efficient bundle for a general bundle is difficult and tedious. Many of the arguments in this part are informal in [GJ02]. We believe that formalizing the concepts and proofs on authentication tests is the main obstacle to formalize the strand space theory. Special effort should be made to simplify and formalize these proofs.

The main contribution of this paper is twofold. The first contribution is to improve the strand space theory itself. We briefly discuss the novelties of our theory below:

- We introduce an inductive definition for bundles, which gives an accurate and formal definition for the operational semantics of protocol execution. It not only provides us a constructive illustration for a bundle, but also introduces an effective and rigorous technique of rule induction to prove properties of bundles. In the classic strand space theory, we can only rely on the implicit induction principle on finiteness of a graph.

- In particular, we use the induction principle on the structures of bundles to formally prove that there exists a path p in a bundle b from the node n at which a message m originates to a node n' containing such a message m. This path clearly formalizes the causal precedence from n to n'. Here b is a general bundle, not necessarily an efficient or a normal one. Our proof is simpler than the counterpart in the classic strand space theory because we can rely on the newly introduced induction principle on a bundle.
- We redefine the subterm relation and *test suite* formally by importing the formalization of message algebra [Pau98]. Two inductively defined operators on messages parts and synth are taken from Paulson's work [Pau98]. The subterm relation is derived from the operator parts in a straightforward way. A test suite G is on a relevant message f, any message g in G, which has f as a subterm, is a component. A component can neither be detached nor decrypted. We formalize an invariance of a penetrator's behaviors on G via the synth operator: a penetrator receives messages in synth G, and then the penetrator can only derive a message which is still in synth G. The operator synth properly characterizes a penetrator's ability to deduce knowledge from test suite. This result is not available in the classic strand space theory.
- A generalized version of authentication tests is proposed and its proof is given. Our formalization is based on a *test suite* G. If a message g originates on a node n in a bundle b such that $\mathsf{term}\ n \in \mathsf{synth}\ G$, and there exists another node n' such that $\mathsf{term}\ n' \notin \mathsf{synth}\ G$. We can safely conclude the existence of regular nodes m and m' such that $n \preceq_b m$, $m' \preceq_b n'$, $m \Rightarrow^+ m'$, $\mathsf{term}\ n \in \mathsf{synth}\ G$, $\mathsf{term}\ n' \notin \mathsf{synth}\ G$. That is to say, the regular edges $m \Rightarrow^+ m'$ is responsible for the transformation from a message in synth G to another one which is not in synth G. Nodes m and m' are regular because a penetrator strand cannot do the transformation based on the aforementioned invariance of a penetrator strand on a test suite. Note that the result of generalized authentication tests holds for a general bundle, and we need not introduce the tedious concepts of normal efficient bundles and the equivalence between bundles. We have avoided the corresponding proof of the existence of an equivalent normal and efficient bundle for the general bundle. Therefore, our proof for the generalized authentication tests is essentially simplified. Less than four pages are needed for the proof, and the proof can be mechanized in Isabelle/HOL.

  Our second contribution is to formalize the newly extended strand space theory in a theorem prover.

- We formalize our extended strand space theory in Isabelle/HOL [NPW02]. A formal theory `strand.thy` is provided as a library, which allows users to do machine-checked proofs on the groundwork of strand space theory. There are two guiding principles in our formalization. Firstly our induction techniques should be conveniently implemented by Isabelle/HOL. Isabelle/HOL is appropriate for this task because of its support for inductively defined sets and its automatic tools. Secondly, our mechanized proofs correspond to their pen-and-paper counterparts as closely as possible. This means that for a person who has completed these proofs manually, little extra effort should be required in order to let Isabelle/HOL check them. Isabelle/HOL offers Isar proof language [Wen99], which is the abbreviation of the phrase "Intelligible semi-automated reasoning". Isar is offered as an alternative proof language interface layer beyond traditional tactical scripts. In Isar proof language, our formal strand space theory is more readable for human beings.
- The formal theory `strand.thy` is applicable generally for case studies on real-world protocol analysis. Effective proof techniques of authentication tests in this library theory can be called to prove security properties of protocols under study. As case studies, some typical protocols such as the Needham-Schroeder-Lowe and the Otway-Rees protocols are formalized, and their security goals are formally proved. In a more recent work, we use our extended strand space theory to formally verify Kerberos V [LP07]. By our experience on case studies, it is shown that our formalized proof techniques can be effectively applied in mechanical protocol analysis. Based on our formal library theory, analyzing a new protocol typically requires about one week's effort.

4. A Formalized Strand Space Theory in Isabelle

4.1. Messages

We directly import the formalization in [Pau97, Pau98] to formalize messages.

```
types key = nat
datatype agent = Server | Friend nat | Spy
datatype msg = Agent agent | Nonce nat | Key key |
               MPair msg msg | Crypt key msg
```

The parts and synth operators are also directly imported from Paulson's formalization. We only show the definition of synth operator. The subterm relation $\sqsubset$ is derived naturally from parts operator as the following:

```
consts synth::msg set ⇒ msg set
inductive synth H
intros
Inj [intro]:g ∈ H ⟹ g ∈ synth H
Agent [intro]: Agent A ∈ synth H
MPair [intro]: ⟦g ∈ synth H; h ∈ synth H⟧ ⟹ ⦃g,h⦄ ∈ synth H
Crypt [intro]: ⟦g ∈ synth H; Key K ∈ H⟧ ⟹ ⦃g⦄_K ∈ synth H

constdefs parts_relation::msg ⇒ msg ⇒ bool (infix ⊏ 55)
parts_relation t1 t2 ≡ t1 ∈ parts{t2}
```

In the above formalization, the command (`infix ⊏ 55`) defines the function as an infix operator, and use the symbol $\sqsubset$ to represent the operator, while the number 55 determines the precedence of the operator. Namely, we can use $t_1 \sqsubset t_2$ to denote parts_relation t_1 t_2.

4.2. Strands and Strand Spaces

First we need to define the basic types such as Sign, signed_msg, Strand, node, and strand_ space. We need to fix a strand set Σ, and a strand space SP for discussion, then we declare them as two constants in our theory.

```
datatype Sign = positive (+ 100) | negative(- 100)
types signed_msg = Sign × msg
typedecl Strand
types node = Strand × nat
types strand_space = Strand ⇒ signed_msg list
consts strand_set::Strand set (Σ)
       SP::strand_space
```

Next we formalize the set of all the nodes Domain, define operators on a node, i.e. strand, index, term,sign, and define the causal relations $\rightarrow$ and $\Rightarrow$.

```
constdefs Domain::node set
       Domain ≡ {(s,i). s ∈ Σ ∧ i<length (SP s)}
constdefs strand::node ⇒ Strand
       strand n ≡ fst n
constdefs index::node ⇒ nat
       index n ≡ snd n
constdefs term::node ⇒ msg
       term n ≡ snd ((SP (fst n))!(snd n) )
constdefs sign::node ⇒ Sign
       sign n ≡ fst ((SP (fst n))!(snd n))

constdefs causal1:: (node × node) set
                 ⎧ (n1,n2). n1 ∈ Domain ∧ n2 ∈ Domain ∧         ⎫
       causal1 ≡ ⎨ sign n1 = + ∧ sign n2 = − ∧                  ⎬
                 ⎩ term n1 = term n2 ∧ strand n1 ≠ strand n2    ⎭
syntax _causal1::node ⇒ node ⇒ bool (infix → 100)
translations n1→n2 ≡ (n1, n2) ∈ causal1

constdefs causal2::(node × node) set
                 ⎧ (n1,n2). n1 ∈ Domain ∧ n2 ∈ Domain ∧  ⎫
       causal2 ≡ ⎨ (strand n1) = (strand n2) ∧          ⎬
                 ⎩ Suc (index n1) = index n2            ⎭
syntax _causal2::node ⇒ node ⇒ bool (infix ⇒ 50)
translations n1⇒n2 ≡ (n1, n2) ∈ causal2
```

In the Isabelle definition of $\rightarrow$ and $\Rightarrow$, we use the syntax and translations commands, which provides a simple mechanism for syntactic macros. A typical use of syntax translation is to introduce relational notation for membership in a set of pairs. For example $n_1 \rightarrow n_2$ is such a macro for abbreviating the predicate $(n_1, n_2) \in$ `causal1`. And the symbol $\Rightarrow$ is used for the causal relation `causal2`. Note that this symbol is also used for the function definition. But users need not be worried about the conflict because Isabelle is able to detect the different meanings of $\Rightarrow$ in different contexts.

4.3. Bundles

We first prepare some notions before introducing the definition of bundle. An edge is a pair of nodes such as (n_1, n_2), and a graph b is pair of a node set and an edge set (ns, es). We use nodes b and edges b to indicate the nodes and the edges of b, respectively. A bundle is a finite graph, which represents the protocol execution under some configuration. Rather than following the way of [JHG98, JHG99], we give a new inductive definition for the set of all the bundles. In this definition, there are five induction rules. Rule `Nil` specifies an empty bundle. The other rules specify bundle's nodes and edges augmented with protocol steps. The intuitive ideas behind the four induction rules are illustrated by Figure 5. More precisely,

- if b is a bundle, and there is a node n_2 with positive sign, which is not a part of b, and it has a $\Rightarrow$-predecessor n_1 in b, then we need to add n_2 into the nodes of b, and add a pair (n_1, n_2) into the edges of b, and the resulting graph is also a bundle;
- if b is a bundle, and there is a node n_2 with positive sign, which is not a part of b, and it has no $\Rightarrow$-predecessor (index $n_2 = 0$), then we need to add n_2 into the nodes of b, and the resulting graph is also a bundle;
- if b is a bundle, and there is a node n_2 with negative sign, which is not a part of b, and it has a $\rightarrow$-predecessor n_1 and $\Rightarrow$-predecessor n'_1 in b, and n_1 has no $\rightarrow$-successor in b, then we need to add n_2 into the nodes of b, and add two pairs (n_1, n_2) (n'_1, n_2) into the edges of b, and the resulting graph is also a bundle (condition $\forall n_3 \in$ nodes b $s.t.$ $(n_1, n_3) \notin$ edges b is used to specify that a node can have only one $\rightarrow$-predecessor, and it can eliminate meaningless bundles as shown in Figure 4(a));

- if b is a bundle, and there is a node n_2 with negative sign, which is not a part of b, and it has a $\rightarrow$-predecessor n_1, n_1 has no $\rightarrow$-successor in b, and n_2 has no $\Rightarrow$-predecessor, then we need to add n_2 into the nodes of b, (n_1, n_2) into the edges of b, and the resulting graph is also a bundle.

```
types edge = node × node
types graph = node set × edge set
constdefs nodes::graph ⇒ node set
       nodes b ≡ fst b
constdefs edges::graph ⇒ edge set
       edges b ≡ snd b

consts bundles::graph set
inductive bundles
intros
Nil: (∅, ∅) ∈ bundles
Add_Pos1: ⟦b∈bundles; sign n2=+; n2∈Domain; n2∉nodes b;
          0<index n2; n1 ∈ nodes b; n1 ⇒ n2⟧
          ⟹ ({n2}∪nodes b, {(n1, n2)}∪edges b)∈ bundles
Add_Pos2: ⟦b ∈ bundles; sign n2=+; n2∉nodes b; n2∈ Domain; index n2=0⟧
          ⟹ ({n2}∪nodes b, edges b) ∈ bundles
Add_neg1: ⟦b∈ bundles; sign n2=-; n2∉nodes b; 0<index n2; n1→n2;
          n1∈nodes b; ∀n3.n3∈nodes b⟶(n1, n3)∉edges b; n1'∈nodes b; n1'⇒n2⟧
          ⟹ ({n2}∪nodes b, {(n1, n2),(n1', n2)}∪edges b) ∈ bundles
Add_neg2: ⟦b∈ bundles; sign n2=-; n2∉nodes b; index n2=0; n1→n2;
          n1∈nodes b; ∀n3.n3∈nodes b⟶(n1, n3)∉edges b⟧
          ⟹({n2}∪nodes b, {(n1, n2)}∪edges b) ∈ bundles
```

Notice that the definition for relation $\rightarrow$ requires strand $n \neq$ strand n', and that for $\Rightarrow$ requires strand $n =$ strand n', the condition whether strand $n =$ strand n' holds can tell us whether (n, n') represents either a $\Rightarrow$ relation or a $\rightarrow$ relation.

Our motivation of introducing such a new definition is twofold. We want to have a more apparent and detailed way to specify the process of constructing a bundle, rather than just specify it as a causally well-defined graph. We also want to take advantage of the rule induction principle on bundles. Namely, a property P holds for every bundle provided that P is preserved under all the rules for constructing bundles. In more details, first we need to prove $P(\emptyset, \emptyset)$ to cover empty bundle, then for each other rules, we must prove an assertion of the form $P(b) \Longrightarrow P(\text{nodes } b \cup \{n\}, \text{edges } b \cup es)$, where n and es are the new node and edges added, and $(\text{nodes } b \cup \{n\}, \text{edges } b \cup es)$ is the resulting augmented bundle by the rule. More formally, such an inductive principle is formalized as an elimination rule `bundles.induct`, as shown below:

```
⟦      xa∈bundles;
       P (∅,∅);
       ⋀ b n1 n2.⟦b∈bundles; P b; sign n2=+; n2∈Domain; n2∉nodes b;
       0<index n2; n1∈nodes b; n1⇒n2⟧
       ⟹P ({n2}∪nodes b,{(n1,n2)}∪edges b);
       ⋀ b n2.⟦b∈bundles; P b; sign n2=+; n2∉nodes b; n2∈Domain; index n2=0⟧
       ⟹P ({n2}∪nodes b, edges b);
       ⋀ b n1 n1' n2.⟦b∈bundles; P b; sign n2=-; n2∉nodes b; n1→n2; 0<index n2;
       n1∈nodes b; (∀n3.n3∈nodes b⟶(n1,n3)∉edges b); n1'∈nodes b; n1'⇒n2⟧
       ⟹ P ({n2}∪nodes b,{(n1,n2),(n1',n2)}∪edges b);
       ⋀ b n1 n2.⟦b∈bundles; P b; sign n2=-; n2∉nodes b; index n2=0; n1→n2;
       n1∈nodes b; (∀n3.n3∈nodes b⟶(n1,n3)∉edges b)⟧
       ⟹P ({n2}∪nodes b,{(n1,n2)}∪edges b)
⟧ ⟹ P xa
```

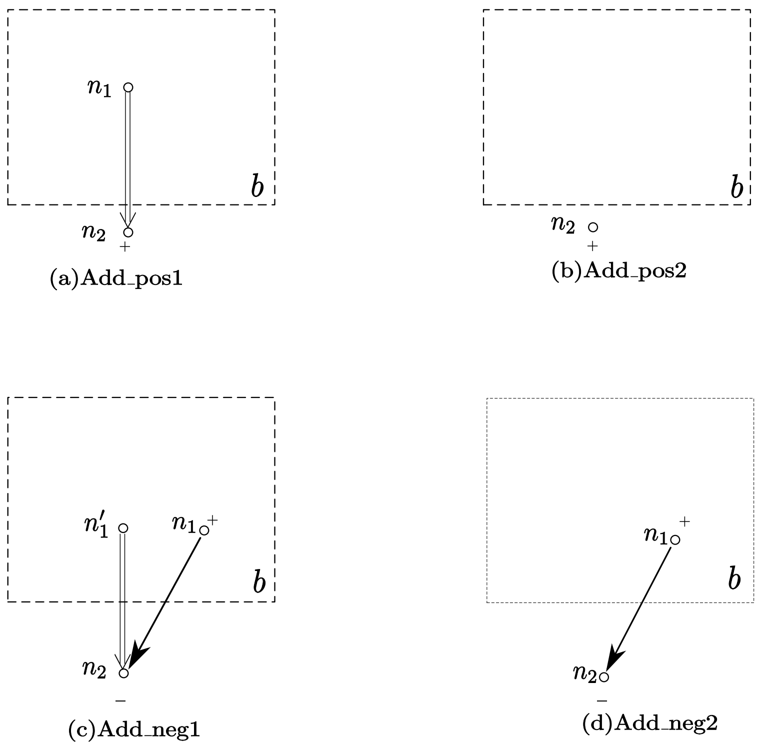

Fig. 5. Bundles' inductive specification.

Applying the induction rule `bundles.induct` yields five subgoals, each of which is for a rule in the definition of bundles. Note that the first goal P $(\emptyset,\emptyset)$ is straightforward, which can be proved automatically in Isabelle.

Our bundle model is sound in the sense that we can prove that a bundle is a well-founded graph. Firstly, a bundle is finite. Here, we use a predicate $\mathsf{finite}\ x$ to denote that x is a finite set.

Lemma 2. $[\![b \in \mathsf{bundles}]\!] \Longrightarrow \mathsf{finite}\ (\mathsf{nodes}\ b) \wedge \mathsf{finite}\ (\mathsf{edges}\ b)$

Lemma 3 and Lemma 4 specify that a bundle is up-wards closed under $\Rightarrow$ and $\rightarrow$.

Lemma 3. $[\![b \in \mathsf{bundles}; \mathsf{sign}\ n = -; n \in \mathsf{nodes}\ b; n' \rightarrow n]\!] \Longrightarrow n' \in \mathsf{nodes}\ b \wedge (n', n) \in \mathsf{edges}\ b$

Lemma 4. $[\![b \in \mathsf{bundles}; n_1 \Rightarrow n_2; n_2 \in \mathsf{nodes}\ b]\!] \Longrightarrow n_1 \in \mathsf{nodes}\ b \wedge (n_1, n_2) \in \mathsf{edges}\ b$

For convenience, we define $m \prec_b n \equiv (m,n) \in (\mathsf{edges}\ b)^+$ and $m \preceq_b n \equiv (m,n) \in (\mathsf{edges}\ b)^*$. Lemma 5 shows that a bundle is acyclic.

Lemma 5. $[\![b \in \mathsf{bundles}]\!] \Longrightarrow (m \prec_b n) \longrightarrow m \neq n$

From the above four lemmas on a bundle, it can be easily derived that the causal dependency relation determined by the bundle's edges is well-founded. Here, we use $\mathsf{wf}\ r$ to denote that r is a well-founded relation, which is defined in the Isabelle/HOL library.

Lemma 6. $[\![b \in \mathsf{bundles}]\!] \Longrightarrow \mathsf{wf}\ (\mathsf{edges}\ b)$

Given a well-founded relation r, the transitive closure of r is also well-founded.

Lemma 7. $[\![b \in \mathsf{bundles}]\!] \Longrightarrow \mathsf{wf}\ (\mathsf{edges}\ b)^+$

The next lemma is about the $\prec_b$-minimal elements in some non-empty set. Its proof needs one lemma in Isabelle library: $\mathsf{wf}\ r = (\forall M\ x. x \in M \longrightarrow (\exists z \in M. \forall y. (y,z) \in r \longrightarrow y \notin M))$, which says that every non-empty set b has a r-minimal element if r is well-founded.

Lemma 8. $[\![b \in \mathsf{bundles}; x \in M; x \in \mathsf{nodes}\ b]\!] \Longrightarrow \exists z \in M.\forall y.(y \prec_b z) \longrightarrow y \notin M$

The existence of minimal members in a non-empty node set of a bundle serves as the most important proof principle in the strand space theory, which clarifies the relation between the strand space theory to the work by Paulson [Pau98] and the work by Schneider [Sch97]. When we apply this proof principle in later sections, most of our arguments rely on the $\prec_b$-minimal elements in some set of nodes.

4.4. Freshness Assumption

Freshness assumptions refer to the facts that confidential items such as nonces and session keys uniquely originate from some node. In order to formalize them, we first need to introduce a predicate originate g n, which says that n is the first node where g occurs in the strand in which g lies. Second, we can define the uniquely_originate g n predicate saying that there is only one node which originates g.

```
constdefs originate::msg⇒node⇒bool
       originate g n ≡ sign n=+ ∧ g⊏term n ∧ (∀i.i<index n⟶g⋢term(strand n,i))

constdefs uniquely_originate::msg⇒node⇒bool
       uniquely_originate g n ≡ originate g n ∧ (∀n'.originate g n'⟶n=n')
```

If g uniquely originates from node n, and another node $n' \neq n$, then (1) if n' is in the same strand as n, then the index of n' is greater than that of n if g also occurs in the term of n'; (2) if n' and n are in different strands, then either the sign of the first node containing g is negative in the strand strand n' or g does not occur in any node of strand n'. This is captured by Lemma 9.

Lemma 9.
$[\![\mathsf{uniquely\_originate}\ g\ n; n \neq n']\!] \Longrightarrow$
$(\mathsf{strand}\ n = \mathsf{strand}\ n' \wedge (g \sqsubset \mathsf{term}\ n' \longrightarrow \mathsf{index}\ n < \mathsf{index}\ n'))\vee$
$(\mathsf{strand}\ n \neq \mathsf{strand}\ n' \wedge ((\exists i.\mathsf{sign}\ (\mathsf{strand}\ n', i) = - \wedge g \sqsubset \mathsf{term}\ (\mathsf{strand}\ n', i) \wedge (\forall j < i.g \not\sqsubset \mathsf{term}\ (\mathsf{strand}\ n', j)))$
$\vee(\forall i.g \not\sqsubset \mathsf{term}\ (\mathsf{strand}\ n', i))))$

For convenience, we define two predicates. One is non_originate g s, saying that message g does not originate from any node in strand s. The other is first_node_in_nonorigi_strand g n m, which says that g does not originate from the strand in which n and m lie, and m is the first node containing g in this strand. These two predicates are used to formalize the notion of a completely transforming path in the next section.

```
constdefs non_originate::msg⇒sigma⇒bool
non_originate t s ≡ ∀n.(strand n=s⟶
          (¬(t⊏term n) ∨ sign n=- ∨ (∃i.i<index n ∧ t⊏term (strand n,i))))
constdefs first_node_in_nonorigi_strand::msg⇒node⇒node⇒bool
first_node_in_nonorigi_strand g n m ≡ sign m=- ∧ g⊏term m ∧ strand m=strand n ∧
          (∀j<index m.g⋢term (strand m,j))
```

4.5. Path

In this section, we discuss a path in a bundle from a node m at which a message g uniquely originates to a node n which contains g as a subterm. Such a path shows the existence of the causal precedence from m to n in the bundle. Furthermore, we show there is a special path p such that for a positive node n' in p such that $g \sqsubset \mathsf{term}\ n'$, the set of all the nodes m' such that $m' \Rightarrow^+ n'$ and $g \sqsubset \mathsf{term}\ m'$ are in p, they can be regarded as a set of nodes which provide complete information to transform term n' in the strand strand n', so we call p a *completely transforming path*. To define a completely transforming path, we introduce a function slice_arr_cons s j len, which returns a consecutive node list $[(s, j), \ldots, (s, j + len)]$. It is defined as follows:

```
consts slice_arr_cons::sigma⇒nat⇒nat⇒node list
primrec
slice_arr_cons s i (len+1)=(s,i)#(slice_arr_cons s (i+1) len)
slice_arr_cons s j 0=[(s,j)]
```

Now we can inductively define the set of all completely transforming paths which starts from some node at which g uniquely originates in a bundle b.

```
consts complete_Path::msg⇒graph⇒((node list) set)
inductive complete_Path g b
intros
add1: ⟦non_originate g (strand m2); sign m2=+; g⊏term m2;
          first_node_in_nonorigi_strand g m2 m; p∈complete_Path g b; last p=m⟧
      ⟹p@(slice_arr_cons (strand m) (index m) (index m2-index m))∈ complete_Path g b
add2: ⟦uniquely_originate g m; strand m2=strand m; g⊏term m2;
          sign m2=+; ⟧
      ⟹(slice_arr_cons (strand m) (index m) (index m2-index m))∈ complete_Path g b
add3: ⟦m1→m2; m1∈nodes b; (m1,m2)∈edges b;
          p∈complete_Path g b; g⊏term m2; last p=m1⟧
      ⟹ p@[m2]∈complete_Path g b
```

The above definition uses three induction rules to specify how a completely transforming path is extended with the causal relation in b. Rule add1 says that if g does not originate from any node in the strand in which a positive node m_2 lies, and $g \sqsubset \mathsf{term}\ m_2$, m is the first node containing g as a subterm in the strand in which m_2 lies, and p is a completely transforming path in b of which the last element is m, then p can be appended with a node list $\mathsf{slice\_arr\_cons}\ (\mathsf{strand}\ m)\ (\mathsf{index}\ m)\ (\mathsf{index}\ m_2 - \mathsf{index}\ m)$. The resulting path is also a completely transforming path in b. Rule add2 says that if g uniquely originates at m, m and m_2 are in the same strand, and $g \sqsubset \mathsf{term}\ m_2$, then $\mathsf{slice\_arr\_cons}\ (\mathsf{strand}\ m)\ (\mathsf{index}\ m)\ (\mathsf{index}\ m_2 - \mathsf{index}\ m)$ is a completely transforming path in b. Rule add3 specifies that a completely transforming path in b is extended with an edge $m_1 \to m_2$ which is an edge of b.

Obviously, if p is a path through a bundle b such that $p \in \mathsf{complete\_Path}\ g\ b$, and g uniquely originates at n, then for any node n' in p, the causal order $n \preceq_b n'$ holds.

Lemma 10. $[\![b \in \mathsf{bundles};\ p \in \mathsf{complete\_Path}\ g\ b; \mathsf{uniquely\_originate}\ g\ n; n' \in \mathsf{set}\ p]\!] \Longrightarrow n \preceq_b n'$

Next lemma is on the complete information provided in such a completely transforming path p. Namely, if $p \in \mathsf{complete\_Path}\ g\ b$, m is a positive node containing g as a subterm, then all the nodes m' such that $m' \Rightarrow^+ m$ and $g \sqsubset \mathsf{term}\ m'$ lie in p, which are regarded as a set of nodes which provide complete information to transform $\mathsf{term}\ m$ in the strand in which m lies.

Lemma 11. $[\![b \in \mathsf{bundles};\ p \in \mathsf{complete\_Path}\ g\ b; m \in \mathsf{set}\ p; \mathsf{sign}\ m = +; g \sqsubset \mathsf{term}\ m]\!] \Longrightarrow (\forall m'. m' \Rightarrow^+ m \wedge g \sqsubset \mathsf{term}\ m' \longrightarrow m' \in \mathsf{set}\ p)$

If p is a path through a bundle b_1 such that $p \in \mathsf{complete\_Path}\ g\ b_1$, and b_1 is a subgraph of b_2, then p is still a completely transforming path through b_2.

Lemma 12. $[\![p \in \mathsf{complete\_Path}\ g\ b_1; \mathsf{nodes}\ b_1 \subseteq \mathsf{nodes}\ b_2; \mathsf{edges}\ b_1 \subseteq \mathsf{edges}\ b_2]\!] \Longrightarrow p \in \mathsf{complete\_Path}\ g\ b_2$

If g uniquely originates from n, n' is a node in bundle b, and g is a subterm of $\mathsf{term}\ n'$, then there exists a path $p \in \mathsf{complete\_Path}\ g\ b$ through b from n to n'.

Lemma 13.

$$[\![b \in \mathsf{bundles}]\!] \Longrightarrow \mathsf{uniquely\_originate}\ g\ n \longrightarrow$$
$$\forall n'. n' \in \mathsf{nodes}\ b \wedge g \sqsubset \mathsf{term}\ n' \longrightarrow (\exists p. p \in (\mathsf{complete\_Path}\ g\ b) \wedge \mathsf{last}\ p = n')$$

The proof itself is not easy to illustrate clearly due to rather complex proof contexts involved, and only a sketch of the proof is shown. In the next section, extracts of the mechanical proof version are given.

Proof. We apply induction to generate cases for each rule. Note that we have a set of induction hypotheses when we prove for each case except the base case `Nil`. We need to construct a path that satisfies our purpose. For notational convenience, we define predicate abbreviations for subgoals in some proof contexts as follows:

- $\mathsf{P}\ g\ b\ n\ n' \equiv n' \in \mathsf{nodes}\ b \wedge g \sqsubset \mathsf{term}\ n' \longrightarrow (\exists p. p \in (\mathsf{complete\_Path}\ g\ b) \wedge \mathsf{last}\ p = n')$;
- $\mathsf{Cons}\ g\ b\ n \equiv \mathsf{uniquely\_originate}\ g\ n \longrightarrow (\forall n'. \mathsf{P}\ g\ b\ n\ n')$.

1. Base case `Nil`: Let $b' = (\emptyset, \emptyset)$. The assertion $\mathsf{Cons}\ g\ b'\ n$ holds vacuously.
2. Case `Add_pos1`: Fix a bundle b and nodes n_1, n_2, such that $\mathsf{sign}\ n_2 = +$, $n_2 \notin \mathsf{nodes}\ b$, $0 < \mathsf{index}\ n_2$, $n_1 \in \mathsf{nodes}\ b$, $n_1 \Rightarrow n_2$. Assume $\mathsf{Cons}\ g\ b\ n$ and let $b' = (\{n_2\} \cup \mathsf{nodes}\ b, \{(n_1, n_2)\} \cup \mathsf{edges}\ b)$, we need to show $\mathsf{Cons}\ g\ b'\ n$. To prove this, we assume $\mathsf{uniquely\_originate}\ g\ n$, fix a node n', and assume $n' \in b'$ and $g \sqsubset \mathsf{term}\ n'$, then we need to show that $\exists p. p \in \mathsf{complete\_Path}\ g\ b' \wedge \mathsf{last}\ p = n'$.

   - If $n_2 = n'$, then there are two cases: (1) $n = n'$, then let $p = \mathsf{slice\_arr\_cons}\ (\mathsf{strand}\ n)\ (\mathsf{index}\ n)\ 0 = [n']$, by rule `add2`, we have $p \in \mathsf{complete\_Path}\ g\ b'$. Obviously, $\mathsf{last}\ p = n'$; (2) $n \neq n'$, then by Lemma 9, there are also two sub-cases.

     (a) $\mathsf{strand}\ n = \mathsf{strand}\ n'$ and $\mathsf{index}\ n < \mathsf{index}\ n'$. Let $p = \mathsf{slice\_arr\_cons}\ (\mathsf{strand}\ n)\ (\mathsf{index}\ n)\ (\mathsf{index}\ n' - \mathsf{index}\ n)$, i.e., $p = [(\mathsf{strand}\ n, \mathsf{index}\ n), \ldots, (\mathsf{strand}\ n, \mathsf{index}\ n')]$. By rule `add2`, we have $p \in \mathsf{complete\_Path}\ g\ b'$. Obviously, $\mathsf{last}\ p = n'$.

     (b) $\mathsf{strand}\ n \neq \mathsf{strand}\ n'$ and $g \sqsubset \mathsf{term}\ n'$. There exists an index i such that $\mathsf{sign}\ (\mathsf{strand}\ n', i) = -$ and $g \sqsubset \mathsf{term}\ (\mathsf{strand}\ n', i)$ and $\forall i_0 < i. g \not\sqsubset \mathsf{term}\ (\mathsf{strand}\ n', i_0)$. By definition, we have $\mathsf{first\_node\_in\_nonorigi\_strand}\ g\ n'\ (\mathsf{strand}\ n', i)$. By $g \sqsubset \mathsf{term}\ n'$ and $\mathsf{sign}\ n' = +$, we have $i < \mathsf{index}\ n'$, which means $i \leq \mathsf{index}\ n' - 1 = \mathsf{index}\ n_1$. By $n_1 \in \mathsf{nodes}\ b$, we know that $(\mathsf{strand}\ n', i) \in \mathsf{nodes}\ b$. By induction hypothesis, we have $\mathsf{Cons}\ g\ b\ n$, with assumption $\mathsf{uniquely\_originate}\ g\ n$, then there exists a path p_0 such that $p_0 \in \mathsf{complete\_Path}\ g\ b$ and $\mathsf{last}\ p_0 = (\mathsf{strand}\ n', i)$. By Lemma 12, we have $p_0 \in \mathsf{complete\_Path}\ g\ b'$. Let $p = p_0 @ \mathsf{slice\_arr\_cons}\ (\mathsf{strand}\ n')\ (i+1)\ (\mathsf{index}\ n' - i - 1)$. By rule `add1`, we can prove that $p \in \mathsf{complete\_Path}\ g\ b'$. Obviously, $\mathsf{last}\ p = n'$.

   - If $n_2 \neq n'$, then $n_2 \in \mathsf{nodes}\ b$. By induction hypothesis, we have $\mathsf{Cons}\ g\ b\ n$, with assumption $\mathsf{uniquely\_originate}\ g\ n$, then there exists a path p_0 such that $p_0 \in \mathsf{complete\_Path}\ g\ b$ and $\mathsf{last}\ p_0 = n'$. Let $p = p_0$. By Lemma 12, we have $p \in \mathsf{complete\_Path}\ g\ b'$.

3. Case `Add_pos2`: The case is simpler. Similar to the argument in case `Add_pos1`, we shall fix a bundle b and nodes n_2, such that $\mathsf{sign}\ n_2 = +$, $n_2 \notin \mathsf{nodes}\ b$, $\mathsf{index}\ n_2 = 0$, and assume $\mathsf{Cons}\ g\ b\ n$, and let $b' = (\{n_2\} \cup \mathsf{nodes}\ b, \mathsf{edges}\ b)$, we need to show $\mathsf{Cons}\ g\ b'\ n$. To prove this, we assume $\mathsf{uniquely\_originate}\ g\ n$, fix a node n', and assume $n' \in b'$ and $g \sqsubset \mathsf{term}\ n'$, then we need to show the following $\exists p. p \in (\mathsf{complete\_Path}\ g\ b') \wedge \mathsf{last}\ p = n'$. There are also two cases:

   - If $n_2 = n'$, then from facts $\mathsf{sign}\ n_2 = +$, $g \sqsubset \mathsf{term}\ n'$ and $\mathsf{index}\ n_2 = 0$, we have $\mathsf{originate}\ g\ n'$. With the fact $\mathsf{uniquely\_originate}\ g\ n$, it can only be $n = n'$, then let $p = \mathsf{slice\_arr\_cons}\ (\mathsf{strand}\ n)\ (\mathsf{index}\ n)\ 0$. The rest argument in this case is similar to (a) in case `Add_pos1`.
   - If $n_2 \neq n'$, the argument in this case is similar to the counterpart in case `Add_pos1`.

4. Case `Add_neg1`: Fix a bundle b and nodes n_1, n_2, such that $\mathsf{sign}\ n_2 = -$, $n_2 \notin \mathsf{nodes}\ b$, $0 < \mathsf{index}\ n_2$, $n_1 \in \mathsf{nodes}\ b$, $n_1 \rightarrow n_2$, $\forall n_3. n_3 \in \mathsf{nodes}\ b \longrightarrow \{(n_1, n3)\} \notin \mathsf{edges}\ b$, $n'_1 \in \mathsf{nodes}\ b$, $n'_1 \Rightarrow n_2$. Assume $\mathsf{Cons}\ g\ b\ n$, and let $b' = (\{n_2\} \cup \mathsf{nodes}\ b, \{(n_1, n_2), (n'_1, n_2)\} \cup \mathsf{edges}\ b$, we need to show $\mathsf{Cons}\ g\ b'\ n$. To prove this, we assume $\mathsf{uniquely\_originate}\ g\ n$, and fix a node n' such that $n' \in b'$ and $g \sqsubset \mathsf{term}\ n'$. Then we need to show $\exists p. p \in (\mathsf{complete\_Path}\ g\ b') \wedge \mathsf{last}\ p = n'$.

   - If $n_2 = n'$, then by $n_1 \rightarrow n_2$, we have $\mathsf{sign}\ n_1 = +$, and $\mathsf{term}\ n_1 = \mathsf{term}\ n_2$. Hence, $g \sqsubset \mathsf{term}\ n_1$. By $n_1 \in \mathsf{nodes}\ b$, and induction hypothesis $\mathsf{Cons}\ g\ b\ n$, with assumption $\mathsf{uniquely\_originate}\ g\ n$, then there exists a path p_0 such that $p_0 \in (\mathsf{complete\_Path}\ g\ b)$ and $\mathsf{last}\ p_0 = n_1$. By Lemma 12, we have $p_0 \in \mathsf{complete\_Path}\ g\ b'$. Let $p = p_0 @ [n']$. By rule `add3`, we have $p \in \mathsf{complete\_Path}\ g\ b'$. Obviously, $\mathsf{last}\ p = n'$.
   - If $n_2 \neq n'$, then $n_2 \in \mathsf{nodes}\ b$, this case is similar to the second sub-case of the case `Add_pos1`.

5. Case `Add_neg2`: Almost the same as case `Add_neg1`.

□

The existence of a completely transforming path is the most important result in our extended strand space model. Its proof relies on the induction principle on the structures of bundles, which differentiates our proof method from the classic strand space theory. The key lies in that we construct a path that satisfies our purpose by analyzing the structure of the bundle and the induction hypothesis. This result holds for a general bundle. We need not to introduce the so-called well-behaved bundles (such as *normal bundles*) and other special kinds of paths (such as *rising* and *falling paths*).

4.6. Extracts from the Proof of Lemma 13

To gain an impression of a mechanical proof, let us look at parts of the proof of Lemma 13. The full proof in Isabelle is too large to include, thus we only concentrate on some key points of the proof steps to illustrate mechanical proof commands and proof assistance provided by Isabelle/Isar. In particular, we show how Isar can help us to do structured proof, which makes the mechanized proof resemble the paper-and-pen one as much as possible.

A theorem proving process can be seen as a sequence of interactions between a user and a theorem prover. The user inputs proof commands, and the prover interprets and executes the commands, and responds with a proof state to show the fixed variables, premises, and goals to be solved. A proof is started by giving a goal to Isabelle, and finished until all goals are solved. At first, a goal is written by a lemma command. For Lemma 13, we write commands as shown below.

```
lemma path_exist:
assumes A: "b ∈ bundles"
shows C: "unique_originate a n⟶ (∀n'.n'∈nodes b ∧
             a⊏ term n' ⟶ (∃p. p∈complete_Path a b ∧ last p=n'))
```

The command assumes A_1 and ... and A_n shows B is an alias for the lemma notation $[\![A_1; A_2; \ldots; A_n]\!] \Longrightarrow B$.

In order to apply rule induction, we use the command "using A proof induct", then Isabelle automatically applies the induction rule `bundles.induct` in Section 4.3 as an elimination rule. The first premise in `bundles.induct` will be eliminated. Note that the property P' under investigation in this lemma is $\lambda b.\mathsf{unique\_originate}\ a\ n \longrightarrow (\forall\ n'.n' \in \mathsf{nodes}\ b \wedge a \sqsubset \mathsf{term}\ n' \longrightarrow (\exists p.p \in \mathsf{complete\_Path}\ a\ b \wedge \mathsf{last}\ p = n'))$. Isabelle will match P' with the schema variable `P` in the rule `bundles.induct`. Five subgoals corresponding to induction rules of bundles are yielded automatically by Isabelle. We only show the second goal corresponding to the case `Add_pos1`.

```
2. ⋀ b n1 n2.
⟦ b∈ bundles;  unique_originate a n ⟶ (∀n'. n'∈nodes b ∧ a⊏term n'⟶
 (∃p.p∈complete_Path a b∧ last p=n'));
 sign n2=+; n2∈Domain; n2∉ nodes b; 0<index n2; n1∈nodes b; n1⇒n2
⟧ ⟹
unique_originate a n ⟶
∀n'.n'∈nodes ({n2}∪ nodes b, {(n1, n2)}∪edges b) ∧a⊏term n'
⟶(∃p.p∈complete_Path a ({n2}∪nodes b, {(n1, n2)}∪edges b)∧last p = n'))
```

Notice that the second premise is the induction hypothesis "$P'\ b$", and the conclusion to be shown is "$P'\ (\{n2\} \cup \mathsf{nodes}\ b, \{(n1, n2)\} \cup \mathsf{edges}\ b)$". From this, we can clearly see that Isabelle can help us do the following work in an induction proof: selecting proper induction rule to execute, generating subgoals for base case and induction steps. Sometimes the subgoals are not so trivial, and it is error-prone for human to write them manually. Isabelle/Isar can automatically finish these tasks without any mistake. After the subgoals are created, our proof structure is naturally decomposed into five parts: one is for the base case *Nil*, the other four are for the induction cases: *Add_Pos*1, *Add_Pos*2, *Add_Neg*1, *Add_Neg*2.

For the subgoal `Add_pos1`, we use the following commands.

```
fix b n1 n2
assume a1:"b∈ bundles" and a2:"sign n2= +" and a3:"n2∈ Domain" and
      a4:"unique_originate a n ⟶ (∀n'.n'∈nodes b ∧ a⊏term n'⟶
      (∃p.p∈complete_Path a b∧last p=n'))" and
      a5:"n2∉nodes b" and a6:"0<index n2" and
      a7:"n1∈nodes b" and a8:"n1⇒n2"
let ?b'="({n2}∪nodes b, {(n1, n2)}∪edges b)"
show "unique_originate a n ⟶ (∀n'.n'∈nodes ?b' ∧ a⊏term n'
      ⟶(∃p.p∈complete_Path a ?b' ∧ last p=n'))"
proof(rule impI, rule allI, rule impI, erule conjE)
      fix n'
      assume b1:"unique_originate a n" and b2:"n'∈nodes ?b'" and b3:"a⊏ term n'"
      show "∃p.p∈complete_Path a ?b'∧last p=n'"
```

Commands fix and assume are used to add fixed variables and new assumptions into the current proof context, and show claims a new local goal and exports it to the corresponding proof context after the proof of the goal is finished. After Isabelle executes them, the resulting proof state is shown as follows:

```
proof (state):step 14
fixed variables: b, a, n, b=b, n1=n1, n2 =n2, n'=n'
prems
      unique_originate a n ⟶ (∀n'.n'∈nodes b ∧ a⊏term n'⟶
      (∃p.p∈complete_Path a b ∧ last p=n'))
      sign n2=+
      n2∈Domain
      n2∉nodes b
      0<index n2
      n1∈nodes b
      n1⇒n2
      unique_originate a n
      n'∈nodes ({n2}∪nodes b,{(n1, n2)}∪edges b)
      a⊏term n'
goal (show, 1 subgoal):
      1. ∃p.p∈complete_Path a ({n2}∪nodes b, {(n1, n2)}∪edges b)∧last p=n'
```

As shown in the proof for case *Add_Pos*1 in Section 4.5, we do case analysis on node `n'`, from the premise `b2:"n'∈nodes ?b'"`, we have two sub-cases: (1) `n'∈nodes b`, (2) `n'=n2`. From both cases, we shall prove the current goal. An overview of our proof is shown as follows:

```
proof -
      (*Two cases P1 and P2 are exhaustively distinguished*)
      from b2 have "n'∈nodes b ∨ n'=n2"
          by (unfold nodes_def,auto)
      (*each proof in one moreover block proves the current goal with each case Pi *)
      moreover
      {assume c1:"n'∈nodes b"
      ...... (*detail proof commands for the case when n'∈nodes b*)
      then have ?thesis by blast
      }
      moreover
      {assume c1:"n'=n2"
      ......(*detail proof commands for the case when n'=n2*)
      then have ?thesis by blast
      }
      ultimately show ?thesis by blast
qed
```

Here, we use Isar commands for calculation reasoning such as `moreover` and `ultimately` to do the proof of all case analysis. From the above extracts of the proof session of the lemma, we can see that Isar proof has a similar structure as the textual one for the case `Add_pos1` introduced in Section 4.5. Therefore, the transformation from a paper-and-pen proof shown in Section 4.5 to the mechanized version in this section is rather straightforward if the Isar proof language is mastered. This is the reason why Isar proof language is so popular and emerges as the main proof language of Isabelle. The assistance of Isabelle prover lies in that it can repeatedly execute proof commands issued by people and generate new proof states until all the goals are solved. Note that some proof states are quite complex, for instance, there are five goals after we apply the induction command. Except the goal for the `Nil` rule, the other goals are not trivial. Isabelle can help us create the new proof state clearly and automatically. These work is too tedious and error-prone for human. Thus people can be liberated to focus on the main proof points which need the most human intelligence: rule induction and case analysis.

The proof script in Isar for this lemma comprises 429 lines, which is longer than those in Isabelle tactic-based proof. Its execution needs 489 steps and finishes in 2 seconds. The script for the case `Nil` is an auto command, which is a standard automatic tactic. The scripts for the rule cases `Add_pos1` and `Add_neg1` are the main parts of all the scripts. The script for the cases `Add_pos2` is simpler, and the script for the case `Add_neg2` is almost the same as the case `Add_neg1`.

In the rest of the paper, for the aim of readability, we do not present the mechanical proofs in Isabelle for lemmas directly, but their proof structures are the same as in their mechanical counterparts.

4.7. Penetrators

First we declare a constant bad to denote the set of all the penetrators. Next we define the constant KP, the set of all the keys which is initially known to all the penetrators. KP contains the public keys of all the agents, the private keys of all the penetrators, and the symmetric keys each of which is initially shared among a penetrator and the server.

```
consts bad::agent set
constdefs KP::key set
       KP ≡ {k.∃A.(k=pubK A) ∨ (A∈bad ∧ (k=priK A ∨ k=shrK A))}
```

Then we need to define penetrator strands according to their trace specification. For example, we define a strand as a T$[g]$ strand or K$[K]$ as follows:

```
constdefs Is_Tee_strand::sigma⇒ bool
       Is_Tee_strand s ≡ ∃g.(SP s)=[(-,g),(+,g),(+,g)]
constdefs Is_K_strand::sigma⇒bool
       Is_K_strand s ≡ ∃K.K∈KP ∧ (SP s)=[(+,Key K)])
```

We use Is_Penetrator_strand s to define that a strand s is a penetrator's strand if and only if its trace is one of the forms defined in Definition 1. A strand is defined to be a regular strand if it is not a penetrator strand, formally Is_Regular_strand $s \equiv \neg$Is_Penetrator_strand s.

```
constdefs Is_Penetrator_strand::sigma⇒ bool
Is_Penetrator_strand s ≡
              Is_Tee_strand s ∨ Is_Flush_strand s ∨
              Is_T_strand s ∨ Is_K_strand s ∨
              Is_Cat_strand s ∨ Is_Sep_strand s ∨
              Is_E_strand s ∨ Is_D_strand s
```

Rather than following the way in [GJ01], we introduce our notion of a component. First we need to define a regular key in a bundle.

```
constdefs regularK::key⇒graph⇒bool
       regularK K b ≡ ∀n∈ nodes b.(term n=Key K) ⟶ Is_regular_strand (strand n)
```

The intuition is that K being regular in b means that K cannot be penetrated in a bundle under consideration. In most cases, we only consider security properties for a protocol in a given bundle, so it is natural for us to just consider whether a key can potentially be penetrated in this bundle. For the private keys or the symmetric keys of regular agents, they will never be compromised if they are not sent as a part of a message. That is, they are regular in any bundle. (In the term of [JHG98], they are *safe keys*.)

In the next definition, component t b means that t is a basic unit that cannot be analyzed in b by penetrators. Namely, neither t can be detached, nor t can be decrypted in b.

```
constdefs component::msg⇒graph⇒bool
       component t b ≡ (∀g h.t≠ ⦃g,h⦄)∧(∀K h.(t=⦃h⦄_K) ⟶ (regularK (invKey K) b))
```

Now we introduce the concept of a test suite.

```
constdefs suite::msg set⇒msg⇒graph⇒bool
       suite G g b ≡ (∀t∈G.(g⊏t⟶component t b)) ∧ (∀t.(g⊄t⟶t∈ G))
```

In the context of this paper, we usually assume that f is an unguessable atomic message such as a nonce or a session key when we mention suite G g b, which is uniquely originated from a regular strand and encrypted in a message. The first conjunctive requires that each message containing g cannot be analyzed by the penetrator. The second conjunctive requires that any message which does not contain g is in G. Let $G_0 = \{t | g \sqsubset t \wedge t \in G\}$. In later discussion, we usually assume that G_0 is the set of messages containing g which some regular agents can send. If G is a test suite for g in b, then the set synth G is a closure which penetrators can synthesize in the bundle b from G. Namely, if the messages received in a penetrator strand are in synth G, then the messages sent in the strand must still be in synth G. Before we prove the closure property, we need to present two useful lemmas.

Lemma 14. $[\![\mathsf{suite}\ G\ f\ b; \{\!|g,h|\!\} \in \mathsf{synth}\ G]\!] \Longrightarrow g \in \mathsf{synth}\ G \wedge h \in \mathsf{synth}\ G$

Lemma 14 shows that if a concatenated message $\{\!|g,h|\!\}$ is in synth G, where G is a test suite for some

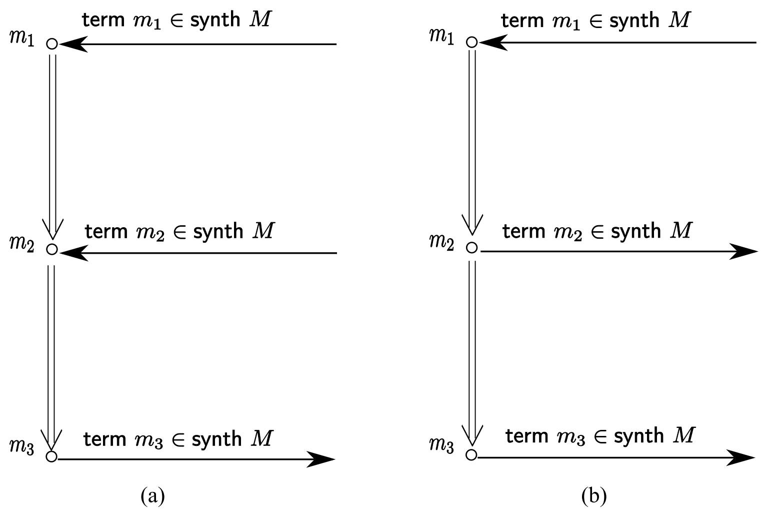

Fig. 6. Penetrator's knowledge closure property.

message f in b, then it must be the case where both $g \in \mathsf{synth}\ G$ and $h \in \mathsf{synth}\ G$. The antecedent $\mathsf{suite}\ G\ f\ b$ eliminates the possibility of the case when $g \notin \mathsf{synth}\ G$ and $h \notin \mathsf{synth}\ G$, but $\{\!|g, h|\!\} \in G$.

From $\{\!|h|\!\}_K \in \mathsf{synth}\ G$, we can conclude that either $\{\!|h|\!\}_K$ is synthesized by h and K, or $\{\!|h|\!\}_K$ is in G. In the former case, h should also be synthesized by G as well.

Lemma 15. $\{\!|h|\!\}_K \in \mathsf{synth}\ G \Longrightarrow h \in \mathsf{synth}\ G \vee \{\!|h|\!\}_K \in G$.

Let g be a message, m' be a positive penetrator node in a bundle b, and $g \sqsubset \mathsf{term}\ m'$, and m' is not the first node in the strand $\mathsf{strand}\ m'$. Namely, there is some node m such that $m \Rightarrow^+ m'$. Suppose G is a test suite for g in the bundle b, if any message that the penetrator can receive in the strand is in $\mathsf{synth}\ G$, then the penetrator can only send a term which is still in $\mathsf{synth}\ G$. Figure 6 illustrates such behaviors of penetrators on knowledge, where (a) shows the cases for $\mathsf{C}[g, h]$, $\mathsf{E}\,[h, K]$, and $\mathsf{D}\,[h, K]$; (b) shows the cases for $\mathsf{S}[g, h]$ and $\mathsf{T}[g]$.

Lemma 16.

$$\begin{array}{l}[\![b \in \mathsf{bundles}; m' \in \mathsf{nodes}\ b; \mathsf{Is\_Penetrator\_strand}\ (\mathsf{strand}\ m'); \mathsf{sign}\ m' = +; \mathsf{suite}\ G\ f\ b;\\ \exists m. m \Rightarrow^+ m'; \forall m_0. (m_0 \Rightarrow^+ m' \wedge \mathsf{sign}\ m_0 = - \longrightarrow \mathsf{term}\ m_0 \in \mathsf{synth}\ G)]\!]\\ \Longrightarrow \mathsf{term}\ m' \in \mathsf{synth}\ G\end{array}$$

Proof. For convenience, the assumption $\forall m_0. (m_0 \Rightarrow^+ m' \longrightarrow \mathsf{term}\ m_0 \in \mathsf{synth}\ M)$ is referred as (1).

By case analysis on the form of penetrator strand, and the assumptions $\mathsf{sign}\ m' = +$ and $\exists m. m \Rightarrow^+ m'$, m' can only be in a strand $\mathsf{C}[g, h]$, $\mathsf{S}[g, h]$, $\mathsf{E}[h, K]$, $\mathsf{D}[h, K]$, $\mathsf{T}[g]$. The proof for case $\mathsf{T}[g]$ is straightforward. Here, we analyze the other cases.

Case 1: $\mathsf{strand}\ m'$ is $\mathsf{C}[g, h]$, then $\mathsf{index}\ m' = 2$, $\mathsf{term}\ (\mathsf{strand}\ m', 0) = g$, $\mathsf{term}\ (\mathsf{strand}\ m', 1) = h$, and $\mathsf{term}\ m' = \{\!|g, h|\!\}$ for some g, h, and $\mathsf{sign}\ (\mathsf{strand}\ m', 0) = -$, and $\mathsf{sign}\ (\mathsf{strand}\ m', 1) = -$.

From the assumption (1), we have $\mathsf{term}\ (\mathsf{strand}\ m', 0) \in \mathsf{synth}\ M$ and $\mathsf{term}\ (\mathsf{strand}\ m', 1) \in \mathsf{synth}\ M$, then $g \in \mathsf{synth}\ M$ and $h \in \mathsf{synth}\ M$. By the definition of synth operator, $\{\!|g, h|\!\} \in \mathsf{synth}\ M$, then $\mathsf{term}\ m' \in \mathsf{synth}\ M$.

Case 2: $\mathsf{strand}\ m'$ is $\mathsf{S}[g, h]$, then $\mathsf{index}\ m' = 1$, or $\mathsf{index}\ m' = 2$, $\mathsf{term}\ (\mathsf{strand}\ m', 0) = \{\!|g, h|\!\}$, $\mathsf{term}\ (\mathsf{strand}\ m', 1) = g$, and $\mathsf{term}\ (\mathsf{strand}\ m', 2) = h$ for some g, h.

From the assumption (1), we have $\mathsf{term}\ (\mathsf{strand}\ m', 0) \in \mathsf{synth}\ M$, $\{\!|g, h|\!\} \in \mathsf{synth}\ M$. By Lemma 14, we have $g \in \mathsf{synth}\ M$ and $h \in \mathsf{synth}\ M$. Hence, $\mathsf{term}\ m' \in \mathsf{synth}\ M$.

Case 3: $\mathsf{strand}\ m'$ is $\mathsf{E}[h, K]$, then $\mathsf{index}\ m' = 2$, $\mathsf{term}\ (\mathsf{strand}\ m', 0) = \mathsf{Key}\ K$, $\mathsf{term}\ (\mathsf{strand}\ m', 1) = h$, and $\mathsf{term}\ m' = \{\!|h|\!\}_K$ for some K, h, and $\mathsf{sign}\ (\mathsf{strand}\ m', 0) = -$, and $\mathsf{sign}\ (\mathsf{strand}\ m', 1) = -$.

From the assumption (1), $\mathsf{term}\ (\mathsf{strand}\ m', 0) \in \mathsf{synth}\ M$ and $\mathsf{term}\ (\mathsf{strand}\ m', 1) \in \mathsf{synth}\ M$, then $\mathsf{Key}\ K \in \mathsf{synth}\ M$ and $h \in \mathsf{synth}\ M$. By the definition of synth, we have $\{\!|h|\!\}_K \in \mathsf{synth}\ M$, then $\mathsf{term}\ m' \in \mathsf{synth}\ M$.

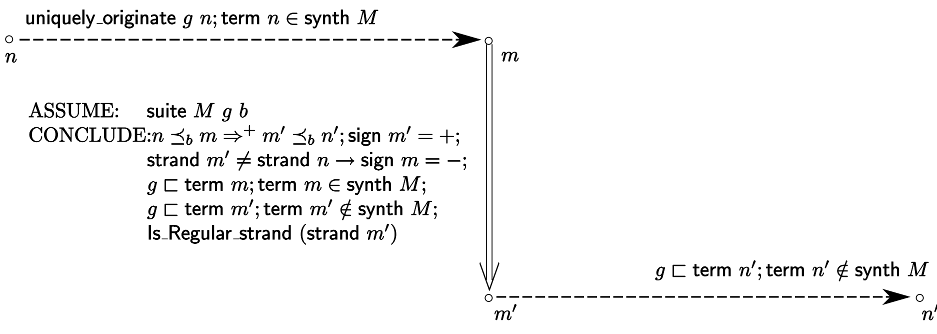
uniquely_originate g n; term $n \in$ synth M
n
m
ASSUME: suite M g b
CONCLUDE: $n \preceq_b m \Rightarrow^+ m' \preceq_b n'$; sign $m' = +$;
strand $m' \neq$ strand $n \rightarrow$ sign $m = -$;
$g \sqsubset$ term m; term $m \in$ synth M;
$g \sqsubset$ term m'; term $m' \notin$ synth M;
Is_Regular_strand (strand m')
$g \sqsubset$ term n'; term $n' \notin$ synth M
m'
n'

Fig. 7. Generalized authentication test.

Case 4: strand m' is $\mathsf{D}[h, K]$, then index $m' = 2$, term (strand $m', 0$) = Key K^{-1}, term (strand $m', 1$) = $\{\!|h|\!\}_K$, and term $m' = h$ for some K, h, and sign (strand m' , 0) = $-$, and sign (strand $m', 1$) = $-$.

From the assumption (1), we have term (strand $m', 0$) $\in$ synth M and term (strand $m', 1$) $\in$ synth M. Therefore, Key $K^{-1} \in$ synth M and $\{\!|h|\!\}_K \in$ synth M. By Lemma 15, we have either (i) term $m' = h \in$ synth M or (ii) $\{\!|h|\!\}_K \in M$. From (i), the lemma can be proved at once. For the case (ii), there are also two sub-cases, either (ii-1) $f \not\sqsubset \{\!|h|\!\}_K$ or (ii-2) $f \sqsubset \{\!|h|\!\}_K$. From (ii-1), we have $f \not\sqsubset \{\!|h|\!\}_K$. Since M is a test suite for f in b, $h \in M$, then $h \in$ synth M. Hence, term $m' \in$ synth$M$. From (ii-2), by M is a test suite for f in b, we have component $\{\!|h|\!\}_K$ b, then we have $\forall n \in$ nodes b.term n = Key $(K^{-1}) \longrightarrow$ Is_Regular_strand(strand n). By this and (strand $m', 0$) $\in$ nodes b and term (strand $m', 0$) = Key K^{-1}. Therefore, Is_Regular_strand (strand m'). But this contradicts with that m' is in a penetrator strand. $\square$

On the other hand, the receiving nodes of a strand get messages which are all in synth G, but a new message, which is not in synth G, is sent in the strand, then the strand must be regular because a penetrator strand cannot create such a term. The result can be simply inferred from Lemma 16.

Lemma 17.

$[\![b \in$ bundles; $m' \in$ nodes b; sign $m' = +$; suite G f b; $\exists m.m \Rightarrow^+ m'$; term $m' \notin$ synth G;
$\forall m_0.(m_0 \Rightarrow^+ m' \wedge$ sign $m_0 = - \longrightarrow$ term $m_0 \in$ synth $G)]\!]$
$\Longrightarrow$ Is_Regular_strand (strand m')

Lemma 16 characterizes the knowledge closure properties of a penetrator's operations on messages. It says that if a penetrator receives messages in synth G, where G is a test suite for some message g. Then the augmented knowledge of the penetrator is still in synth G after the receiving actions. Namely, synth G is the closure of knowledge which the penetrator can deduce from the test suite G. On the other hand, if a message, which is not in the closure synth G, can be deduced at a node of strand, then the strand must be regular. Therefore, Lemma 17 is a key technique for us to check whether a strand is regular. We find this result useful in proving that a strand is regular in the next section on generalized authentication tests. Note that the two lemmas relate the algebraic operator synth in trace theory [Pau98] with penetrator's strand ability to deduce knowledge, which differs our work from the classic strand space theory. Such closure properties are not available in the classic strand space theory as message algebra operators such as synth are not introduced. Here b is also a general bundle, not necessarily an efficient or a normal one. This result helps us to eliminate the need to introduce the tedious concepts of normal efficient bundles, and the equivalence between bundles, and avoid the corresponding proof of the existence of an equivalent normal and efficient bundle for the general bundle.

4.8. Generalized Authentication Tests

In this section we first present our notion of generalized authentication tests.The meaning of our assertion is illustrated in Figure 7. Suppose that b is a bundle, g is a message, G is a test suite for g in b, g uniquely originates on n, and term $n \in$ synth G. If $g \sqsubset$ term n', and term n' is not in synth G, then we can safely conclude the existence of regular nodes m and m' such that $n \preceq_b m$, $m' \preceq_b n'$, $m \Rightarrow^+ m'$, term $n \in$ synth G, term $n' \notin$ synth G. That is to say, the regular edges $m \Rightarrow^+ m'$ is responsible for the transformation from a message in $\mathsf{synth}\ G$ to another one which is not in $\mathsf{synth}\ G$. Nodes m and m' are regular because a penetrator strand cannot do such a transformation based on the aforementioned invariance of a penetrator strand on a test suite in Section 4.7. By Lemma 16, a penetrator can only emit a message which is still in $\mathsf{synth}\ G$ if the term of any receiving node in the strand is in $\mathsf{synth}\ G$, but $\mathsf{term}\ n' \notin \mathsf{synth}\ G$. Therefore, there must be a regular strand performing such a transformation for the first time. Note that g is a general message, not necessarily an atomic message.

Lemma 18 (Generalized authentication tests).

$$\begin{array}{l} [\![b \in \mathsf{bundles}; \mathsf{uniquely\_originate}\ g\ n; \mathsf{suite}\ G\ g\ b; \mathsf{term}\ n \in \mathsf{synth}\ G; \\ n' \in \mathsf{nodes}\ b; g \sqsubset \mathsf{term}\ n'; \mathsf{term}\ n' \notin \mathsf{synth}\ G)]\!] \\ \Longrightarrow \exists m\ m'. \left(\begin{array}{l} n \preceq_b m \wedge m' \preceq_b n' \wedge m \Rightarrow^+ m' \wedge \mathsf{sign}\ m' = + \wedge \\ (\mathsf{strand}\ m' \neq \mathsf{strand}\ n \to \mathsf{sign}\ m = -) \wedge g \sqsubset \mathsf{term}\ m \wedge \\ \mathsf{term}\ m \in \mathsf{synth}\ G \wedge g \sqsubset \mathsf{term}\ m' \wedge \mathsf{term}\ m' \notin \mathsf{synth}\ G \wedge \\ \mathsf{Is\_Regular\_strand}\ (\mathsf{strand}\ m') \end{array} \right) \end{array}$$

Proof. Let us define $M = \{x \mid x \preceq_b n' \wedge g \sqsubset \mathsf{term}\ x \wedge \mathsf{term}\ x \notin \mathsf{synth}\ G\}$.

Obviously $n' \in M$, then by Lemma 8, we have (1) there is a node m' such that $m' \in M$, $g \sqsubset \mathsf{term}\ m'$, $m' \preceq_b n'$, and for all $y \in \mathsf{nodes}\ b$, if $y \prec_b m'$, then $y \notin M$. From $n' \in \mathsf{nodes}\ b$ and $m' \preceq_b n'$, we have $m' \in \mathsf{nodes}\ b$.

First we show $\mathsf{sign}\ m' = +$. We prove it by contradiction. If $\mathsf{sign}\ m' = -$, then by Lemma 3, there is a node m'' such that $m'' \to m'$, $m'' \preceq_b m'$, $\mathsf{term}\ m' = \mathsf{term}\ m''$. Hence, $m'' \in M$. This contradicts the minimality of m'.

From $m' \in M$, then we have $\mathsf{term}\ m' \notin \mathsf{synth}\ G$, with $\mathsf{term}\ n \in \mathsf{synth}\ G$, we know (2) $m' \neq n$.

Furthermore, for any m'' such that $m'' \Rightarrow^+ m'$, either (3.1) $g \sqsubset \mathsf{term}\ m''$ or (3.2) $g \not\sqsubset \mathsf{term}\ m''$. For the case (3.1), from $m' \in \mathsf{nodes}\ b$ and $m'' \Rightarrow^+ m'$, we have $m'' \prec_b m'$. Then by (1) and $g \sqsubset \mathsf{term}\ m''$, we have $\mathsf{term}\ m'' \in \mathsf{synth}\ G$. For the case (3.2), it is obvious that $\mathsf{term}\ m'' \in \mathsf{synth}\ G$ by the definition of $\mathsf{suite}\ G\ g\ b$. So it can be concluded that (3) $\forall m_0.(m_0 \Rightarrow^+ m' \longrightarrow \mathsf{term}\ m_0 \in \mathsf{synth}\ G)$.

From (2) and $g \sqsubset \mathsf{term}\ m'$, by Lemma 9, either (4.1) $\mathsf{strand}\ m' = \mathsf{strand}\ n$ and $\mathsf{index}\ n < \mathsf{index}\ m'$ or (4.2) $\mathsf{strand}\ n \neq \mathsf{strand}\ m'$ and there exists an index i such that $\mathsf{sign}\ (\mathsf{strand}\ n', i) = -$ and $g \sqsubset \mathsf{term}\ (\mathsf{strand}\ n', i)$, and for all i_0, if $i_0 < i$, then $g \not\sqsubset \mathsf{term}\ (\mathsf{strand}\ n', i_0)$. For the case (4.1), let m be n. For the case (4.2), let m be $(\mathsf{strand}\ n', i)$, which is the first node containing g in $\mathsf{strand}\ m'$. In both cases, there exists a node m that satisfies $m \Rightarrow^+ m'$ and $g \sqsubset \mathsf{term}\ m$. By this and (3) together with $\mathsf{term}\ m' \notin \mathsf{synth}\ G$, by Lemma 17, we have $\mathsf{Is\_Regular\_strand}(\mathsf{strand}\ m')$.

From $m' \in \mathsf{nodes}\ b$ and $m \Rightarrow^+ m'$, we have $m \prec_b m'$ and $m \in \mathsf{nodes}\ b$. By this and $g \sqsubset \mathsf{term}\ m$, then by Lemma 13, there exists a path p from n to m through b. By Lemma 10, we have $n \preceq_b m$. Therefore, such m, m' satisfy the requirements in the conclusion part of the lemma. □

Here, we call node pair (n, n') a *generalized authentication test* for g because we can get an authentication guarantee that there must be some regular strand. Combining with trace analysis of the protocol under study, we can verify that the regular strand is of the intended agent, who has thereby been authenticated. Usually nodes n and n' are in the same regular strand, where n sends a term in $\mathsf{synth}\ G$ and later n' receives back another term that is not in $\mathsf{synth}\ G$.

An outgoing authentication tests is a special case of our generalized authentication tests. For the outgoing test shown in Figure 2, we can let $G = \{\{\!|h|\!\}_K\} \cup \{t | a \not\sqsubset t\}$. Note that a is a session key or nonce, and $a \sqsubset h$. We have $\mathsf{term}\ n \in \mathsf{synth}\ G$, and $\mathsf{term}\ n' \notin \mathsf{synth}\ G$ if K^{-1} cannot be penetrated.

4.9. Unsolicited Authentication Tests

The notion of unsolicited tests is introduced in [GJ01], which is frequently used to prove that a key server authenticates its clients. An *unsolicited authentication test* is a kind of regularity about an encrypted term $\{\!|h|\!\}_K$, where K is a long-term regular key. Once $\{\!|h|\!\}_K$ occurs as a subterm of a node n in a bundle b, it can be ensured that there is a positive regular node m originating $\{\!|h|\!\}_K$ as a subterm (i.e. m has $\{\!|h|\!\}_K$ as a subterm, and it also holds that $\{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ m'$ for any node $m' \prec_b m$). Intuitively, the reason why m must be regular lies in that K cannot be penetrated in the bundle b. So the penetrator cannot create $\{\!|h|\!\}_K$ by encrypting h with K.

Lemma 19 (Unsolicited authentication tests).

$$\begin{array}{l}[\![b \in \mathsf{bundles}; n \in \mathsf{nodes}\ b; \{\!|h|\!\}_K \sqsubset \mathsf{term}\ n; \mathsf{regularK}\ K\ b]\!] \\ \Longrightarrow \exists m. \left(\begin{array}{l} m \preceq_b n \wedge \mathsf{sign}\ m = + \wedge \{\!|h|\!\}_K \sqsubset \mathsf{term}\ m\ \wedge \\ \mathsf{Is\_Regular\_strand}\ (\mathsf{strand}\ m)\ \wedge \\ (\forall y. y \prec_b m \longrightarrow \{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ y) \end{array} \right)\end{array}$$

Proof. Let $M = \{x \mid x \preceq_b n \wedge \{\!|h|\!\}_K \sqsubset \mathsf{term}\ x\}$.

Obviously, $n \in M$. By the well-foundedness of a bundle, there exists a node m such that m is minimal in M, which means $\{\!|h|\!\}_K \sqsubset \mathsf{term}\ m$, and for all node $m' \in \mathsf{nodes}\ b$, if $m' \prec_b m$ then $m' \notin M$ and $\{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ m'$.

First, we prove that the sign of m is positive. If $\mathsf{sign}\ m = -$, then by upward-closed property of a bundle there must be another node $m'' \in \mathsf{nodes}\ b$ such that $\mathsf{sign}\ m'' = +$ and $m'' \rightarrow m'$. This contradicts with the minimality of m.

Second, we prove that m is regular by deriving contradictions if m is in a penetrator strand. Here we only analyze the cases when m is in either $\mathsf{C}[g, g']$ (concatenation strand) or $\mathsf{E}[g, K]$ (encryption strand). Other cases are either straightforward or can be analyzed in a similar way.

- m is in $i \in \mathsf{C}\,[g, g']$. By the form of the strand $\mathsf{C}\,[g, g']$ and the fact that m is a positive node, we have $m = (i, 2)$, $\mathsf{term}\ m' = \{\!|g, g'|\!\}$, $\mathsf{term}\ (i, 0) = g$, and $\mathsf{term}\ (i, 1) = g'$ for some g, g'. By the upwards-closed property of a bundle, we have that nodes $(i, 0)$ and $(i, 1)$ must be in b. By $\{\!|h|\!\}_K \sqsubset \{\!|g, g'|\!\}$, we have either $\{\!|h|\!\}_K \sqsubset g$ or $\{\!|h|\!\}_K \sqsubset g'$. So either node $(i, 0) \in M$, or node $(i, 1) \in M$. Both contradict with the minimality of m.
- m is in $i \in E_{g,K'}$. By the form of the strand $\mathsf{E}[g, K]$ and the fact that m is a positive node, we have $m = (i, 2)$, $\mathsf{term}\ m = \{\!|g|\!\}_{K'}$, $\mathsf{term}\ (i, 0) = K'$, and $\mathsf{term}\ (i, 1) = g$ for some g, K'. So $\{\!|h|\!\}_K \sqsubset \{\!|g|\!\}_{K'}$. Hence, it is straightforward that either (1) $\{\!|h|\!\}_K \sqsubset g$ or (2) $h = g$ and $K = K'$. For (1), we have $\{\!|h|\!\}_K \sqsubset \mathsf{term}\ (i, 1)$. It is easy to derive a contradiction by the same argument as in the first case. For (2), by the assumption that K must be regular in b, i must be regular, and this contradicts with the fact that i is a penetrator strand.

□

The above proof totally depends on the well-founded induction principle on bundles, and we have formalized the proof of this lemma in [Li08]. In fact, Lemma 19 also provides a useful proof method to reason about authentication properties based on secrecy properties. Note that the premise that n is an unsolicited test for $\{\!|h|\!\}_K$ requires that K is regular w.r.t. $n \in \mathsf{nodes}\ b$, which is an assumption on the secrecy of K. And the conclusion is an authentication guarantee of the existence of a regular node m. Besides, compared with the original version of unsolicited test [GJ01], our result has two extensions that $m \preceq_b n$ and m is minimal (i.e., $\{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ m'$ for any node m' such that $m' \preceq_b m$). We find that the extended version of unsolicited authentication test is quite useful in many cases. We discuss this in the case study on Otway-Rees protocol.

Interestingly enough, we can derive a generalized version of incoming authentication tests if we combine unsolicited tests with the assumptions that some term g uniquely originates. In details, if a term g uniquely originates at some node n, $g \sqsubset \{\!|h|\!\}_K$, and $\{\!|h|\!\}_K$ does not occur in n, but $\{\!|h|\!\}_K$ occurs in another node n', then there exists a transforming edge $m \Rightarrow^+ m'$ in a regular strand such that $\{\!|h|\!\}_K \sqsubset \mathsf{term}\ m'$ and $\{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ m$.

Lemma 20 (Incoming authentication tests).

$$\begin{array}{l}[\![b \in \mathsf{bundles}; n' \in \mathsf{nodes}\ b; \mathsf{uniquely\_originate}\ g\ n; \{\!|h|\!\}_K \sqsubset \mathsf{term}\ n'; g \sqsubset h; \\ \{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ n; \mathsf{regularK}\ K\ b]\!] \Longrightarrow \\ \exists m\ m'. \left(\begin{array}{l} n \preceq_b m \wedge m' \preceq_b n' \wedge m \Rightarrow^+ m' \wedge \mathsf{sign}\ m' = +\ \wedge \\ (\mathsf{strand}\ m' \neq \mathsf{strand}\ n \rightarrow \mathsf{sign}\ m = -)\ \wedge \\ \{\!|h|\!\}_K \sqsubset \mathsf{term}\ m' \wedge \{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ m\ \wedge \\ \mathsf{Is\_Regular\_strand}\ (\mathsf{strand}\ m) \end{array} \right)\end{array}$$

The proof is rather straightforward. Firstly, by Lemma 19, we can infer that there exists a positive regular node m' which satisfies $m' \preceq_b n'$ and $\{\!|h|\!\}_K \sqsubset \mathsf{term}\ m'$. Furthermore, for all y, if $y \prec_b m$, then $\{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ y$. Secondly, there are two cases: either $\mathsf{strand}\ m' = \mathsf{strand}\ n$ or $\mathsf{strand}\ m' \neq \mathsf{strand}\ n$. In the first case, let $m = n$, we can easily derive the conclusion. In the second case, by Lemma 9, there exists a negative node m which is in the same strand as m' and m is the first node containing g as a subterm.

If n and n' are in the same regular strand, then we can formalize the version of incoming authentication tests as defined in [GJ01].

All the definitions, lemmas, and proofs in this section are implemented in a formal theory `strand.thy`, which provides a mechanized library for protocol analysis. This theory comprises 4636 lines. Its execution needs only 40 seconds.

5. Applications

5.1. General Modeling and Proof Strategy for Protocol Analysis

To analyze a protocol with our mechanical strand space method, we need to import the `strand.thy` theory. Besides, we need to specify the form of all regular strands of the protocol, and to define the strand space to be the union of regular strands of the protocol or penetrators.

In this paper we mainly focus on authentication and secrecy properties. As mentioned in Section 2.5, a value g is secret for a protocol if in every bundle b of the protocol the penetrator cannot receive g in cleartext; that is, there is no node n in b such that $\mathsf{term}\ n = g$. Usually g is a long-term key of a regular agent, a nonce, or a session key issued by a server. To prove the secrecy property of a long term key K, which is not a part of any message, e.g., the private key $\mathsf{priK}\ A$ of any regular agent A in NSL protocol, we use the well-foundedness of property of a bundle, and we prove it by contradiction. Normally, we construct a set

$$M = \{n | n \in \mathsf{nodes}\ b \wedge \mathsf{Key}\ K \sqsubset \mathsf{term}\ n\},$$

and need to show $M = \emptyset$. If M is non-empty, then by the existence of a $\prec_b$ minimal element of any non-empty set of bundle nodes (Lemma 8), there is a $\prec_b$ minimal element in S. By case analysis on the penetrator strands and regular strands, we can derive a contradiction. From this, we can derive that there is no node n such that $K \sqsubset \mathsf{term}\ n$. On the other side, if we want to prove the secrecy of an atom a such as a nonce or a session key, which is a part of a message, e.g., the nonce Na originated from an initiator strand in the NSL protocol, we can directly use the authentication tests. Because such value a is usually uniquely originated and encrypted by an unpenetrated key K in a context such as $\{\!|v|\!\}_K$ by a regular strand, where $a \sqsubset v$. We can prove this by contradiction. Let

$$G = \{t | \exists t\ v\ K. t = \{\!|v|\!\}_K \wedge a \sqsubset v \wedge \mathsf{regularK}\ K\ b\} \cup \{t | a \not\sqsubset t\},$$

if there is a node n such that $\mathsf{term}\ n = a$, then obviously $\mathsf{term}\ n \notin \mathsf{synth}\ G$. By authentication test, there is a regular strand in which there are two nodes m and m' such that $a \sqsubset \mathsf{term}\ m$, $\mathsf{term}\ m \in \mathsf{synth}\ G$, $a \sqsubset \mathsf{term}\ m'$ and $\mathsf{term}\ m' \notin \mathsf{synth}\ G$. However by the specifications on regular strands, we can prove that such a strand does not exist. Therefore a contradiction is obtained.

Authentication test and unsolicited test are two main techniques to prove authentication guarantees. The details of applying the two techniques are discussed in case studies on the NSL and Otway-Rees protocols. Here we only emphasize key steps in applying this technique. First we usually need to prove the secrecy of some key K when we apply the above two proof techniques. Recall that in an authentication test we require any encrypted term containing g in a test suite G should be encrypted by some key whose inverse key is not penetrated in the bundle b, and in an unsolicited test we also require that the key K cannot be leaked in the bundle b which is used to encrypt some subterm $\{\!|h|\!\}_K$. Second we need to figure out a proper message set G which is a test suite for a relevant atom a. Then we check the antecedents of generalized authentication tests are satisfied in a routine way. At last, combining the analysis of traces of regular strands and some side assumptions, we can ensure that only the intended regular strand exists.

5.2. Example 1: the Needham-Schroeder-Lowe Protocol

For the NSL protocol, we define its initiator and responder strands. For instance, we define an initiator and a responder strands of NSL protocol as follows:

$$
\begin{array}{l}
\texttt{constdefs NSLInit::sigma}\Rightarrow\texttt{agent}\Rightarrow\texttt{agent}\Rightarrow\texttt{nat}\Rightarrow\texttt{nat}\Rightarrow\texttt{bool}\\
\texttt{NSLInit s A B Na Nb} \equiv\\
\texttt{(SP s)=}\left[\begin{array}{l}
(+,\texttt{Crypt pubK B } \{\!|\texttt{Nonce Na, Agent A}|\!\}),\\
(-,\texttt{Crypt pubK A } \{\!|\texttt{Nonce Na, Nonce Nb, Agent B}|\!\}),\\
(+,\texttt{Crypt pubK B } \{\!|\texttt{Nonce Nb}|\!\})
\end{array}\right]\\
\\
\texttt{constdefs NSLResp::sigma}\Rightarrow\texttt{agent}\Rightarrow\texttt{ agent }\Rightarrow\texttt{nat}\Rightarrow\texttt{nat}\Rightarrow\texttt{bool}\\
\texttt{NSLResp s A B Na Nb} \equiv\\
\texttt{(SP s)=}\left[\begin{array}{l}
(-,\texttt{Crypt (pubK B) } \{\!|\texttt{Nonce Na,Agent A}|\!\}),\\
(+,\texttt{Crypt (pubK A) } \{\!|\texttt{Nonce Na,Nonce Nb,Agent B}|\!\}),\\
(-,\texttt{Crypt (pubK B) (Nonce Nb)})
\end{array}\right]
\end{array}
$$

Next we define the strand space of NSL protocol to be the union of NSL initiator or NSL responder or penetrator strands.

$$
\texttt{defs NSL\_def:}\Sigma \equiv \left\{\texttt{s.}\left(\begin{array}{l}
\exists\texttt{A B Na Nb. NSLInit s A B Na Nb} \lor\\
\exists\texttt{A B Na Nb. NSLResp s A B Na Nb} \lor\\
\texttt{Is\_Penetrator\_strand s}
\end{array}\right)\right\}
$$

In the following discussion, we show how to prove some security properties of the NSL protocol. First we need to prove the secrecy property on any regular key.

Given a bundle in the NSL strand spaces, for any node in the bundle, the term of this node cannot be a key which is not in KP because neither a penetrator can emit a key K such that $K \notin \mathsf{KP}$ nor any regular agent can send a message containing such a key.

Lemma 21. $[\![b \in \mathsf{bundles}; n \in \mathsf{nodes}\ b; K \notin \mathsf{KP}]\!] \Longrightarrow \mathsf{Key}\ K \not\sqsubset \mathsf{term}\ n$

Proof. Let $M = \{x \mid x \in \mathsf{nodes}\ b \land \mathsf{Key}\ K \sqsubset \mathsf{term}\ x\}$.

We show that M is empty by contradiction. If there is a node $n \in M$, then by the well-foundedness of a bundle, there exists a node m such that m is minimal in M. Namely, $m \in \mathsf{nodes}\ b$, $\mathsf{Key}\ K \sqsubset \mathsf{term}\ m$, and for all $m' \in \mathsf{nodes}\ b$, if $m' \prec_b m$ then $m' \notin M$.

We prove that the sign of m is positive. If $\mathsf{sign}\ m = -$, then by upward-closed property of a bundle there must be another node m'' in the bundle b such that $\mathsf{sign}\ m'' = +$ and $m'' \to m$. This contradicts with the minimality of m. Then m is either in a regular strand or in a penetrator strand.

- m is in a regular strand. Then by the definition of NSL strands, there are two cases. Here we only analyze the case when m is in a responder strand s such that $\mathsf{NSLresp}[s, A, B, Na, Nb]$ for some A, B, Na, Nb. The other case can be analyzed in a similar way. By inspection on the trace form of a responder strand, we have $m' = (s, 1)$, but $\mathsf{term}\ (s,1) = \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A}$. Obviously $\mathsf{Key}\ K \not\sqsubset \mathsf{term}\ (s,1)$. We have a contradiction.
- m is in a penetrator strand s'. Here we only analyze the cases when s' is either $\mathsf{K}[K']$ (key strand) or $\mathsf{C}[g,h]$ (concatenation). Other cases are either straightforward or can be analyzed in a similar way.

- s' is $\mathsf{K}[K']$. We have $m = (s', 0)$ and $K = K' \in \mathsf{KP}$. This contradicts with $K \notin \mathsf{KP}$.
- s' is $\mathsf{C}[g,h]$. We have $m = (s', 2)$ and $k \sqsubset \{\!|g,h|\!\}$. By the definition of $\sqsubset$, we have $\mathsf{Key}\ K \sqsubset g$, or $\mathsf{Key}\ K \sqsubset h$. If $\mathsf{Key}\ K \sqsubset g$, then $\mathsf{Key}\ K \sqsubset \mathsf{term}\ (s', 0)$. This contradicts with the minimality of m. The case when $\mathsf{Key}\ K \sqsubset h$ can be analyzed similarly.

□

From the above lemma and the definition of KP, we can easily derive that each private key of a regular agent cannot be penetrated by any nodes in the bundle. This lemma is used later when we apply authentication results to prove the initiator's and responder's authentication guarantees.

Lemma 22. $[\![b \in \mathsf{bundles}; A \notin \mathsf{bad}]\!] \Longrightarrow \mathsf{regularK}\ (\mathsf{priK}\ A)\ b$

If some nonce Na uniquely originates on an initiator strand s, the nonce can uniquely identify the strand, so if another initiator strand s' satisfies $\mathsf{is\_initiator}\ s'\ A\ B\ Na\ Nb$, then s must be s'.

Lemma 23. $[\![\mathsf{uniquely\_originate}\ (\mathsf{Nonce}\ Na)\ (s,0); \mathsf{NSLInit}\ s\ A_0\ B_0\ Na\ Nb_0; \mathsf{NSLInit}\ s'\ A\ B\ Na\ Nb]\!] \Longrightarrow s = s' \wedge A_0 = A \wedge Nb_0 = Nb \wedge B_0 = B$

Now we prove the initiator's and responder's authentication guarantee. Here we only give the proof of the former, which is more difficult than that of the latter. The difficulty lies in that we need not only to prove a responder strand $\mathsf{NSLResp}[A, B, Na, N]$ exists for some nonce N, but also to ensure that $N = Nb$. Our main technique is the result of authentication test, and the key step of applying authentication test is to figure out a proper G (in Lemma 18) to serve our aim.

Lemma 24 (Initiator's guarantee).

$$\begin{array}{l}[\![b \in \mathsf{bundles}; \mathsf{NSLInit}\ s\ A\ B\ Na\ Nb; \mathsf{uniquely\_originate}\ (\mathsf{Nonce}\ Na)\ (s,0);\\ (s,2) \in \mathsf{nodes}\ b; A \notin \mathsf{bad}; B \notin \mathsf{bad}; Na \neq Nb]\!] \Longrightarrow\\ \exists r. \mathsf{NSLResp}\ r\ A\ B\ Na\ Nb\ \wedge (r,1) \in \mathsf{nodes}\ b\end{array}$$

Proof. From $(s,2) \in \mathsf{nodes}\ b$, we have $(s,1) \in \mathsf{nodes}\ b$. Let $a = \mathsf{Nonce}\ Na$, $n = (s,0)$, $n' = (s,1)$, then $\mathsf{term}\ n' = \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A}$, and we define

$$G_0 = \{t. \exists r\ N. ((r,1) \in \mathsf{nodes}\ b \wedge \mathsf{NSLResp}\ r\ A\ B\ Na\ N \wedge t = \mathsf{term}\ (r,1))\},$$

$$G = G_0 \cup \{\{\!|\mathsf{Nonce}\ Na, \mathsf{Agent}\ A|\!\}_{pubK\ B}\} \cup \{t \mid a \not\sqsubset t\}.$$

Obviously if $t \in G_0$, then $t = \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ N, \mathsf{Agent}\ B|\!\}_{pubK\ A}$ for some N.

From $A \notin \mathsf{bad}$ and $B \notin \mathsf{bad}$, by Lemma 21, we have that neither $\mathsf{priK}\ A$ nor $\mathsf{priK}\ B$ can be penetrated in b. So G is a test suite for $\mathsf{Nonce}\ Na$ in b. Then by Lemma 14 and Lemma 15, we have two auxiliary results.

(1) $[\![t = \{\!|\mathsf{Nonce}\ Na', \mathsf{Nonce}\ Nb', \mathsf{Agent}\ B'|\!\}_{pubK\ A'}; t \in \mathsf{synth}\ G; \mathsf{Nonce}\ Na \sqsubset t]\!] \Longrightarrow t \in G_0$

(2) $[\![t = \{\!|\mathsf{Nonce}\ Na', \mathsf{Agent}\ A'|\!\}_{pubK\ B'}; t \in \mathsf{synth}\ G; \mathsf{Nonce}\ Na \sqsubset t]\!] \Longrightarrow B' = B \wedge Na' = Na \wedge A' = A$

We show that $\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A} \in \mathsf{synth}\ G$. We prove it by contradiction. Suppose $\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A} \notin \mathsf{synth}\ G$. It is rather routine to show that all assumptions of Lemma 18 can be satisfied, so (a) there are two regular nodes m_1' and m_1 such that $n \preceq_b m_1$, $m_1' \preceq_b n'$, $m_1 \Rightarrow^+ m_1'$, $a \sqsubset \mathsf{term}\ m_1$, $\mathsf{term}\ m_1 \in \mathsf{synth}\ G$, $a \sqsubset \mathsf{term}\ m_1'$, $\mathsf{term}\ m_1' \notin \mathsf{synth}\ G$, $\mathsf{Is\_Regular\_strand}\ \mathsf{strand}\ m_1'$, $\mathsf{sign}\ m_1 = -$ and $\mathsf{sign}\ m_1' = +$.

The proof of $\mathsf{sign}\ m_1 = -$ is simple, we only need to prove $\mathsf{strand}\ m_1 \neq s$. We prove it by contradiction. Suppose $\mathsf{strand}\ m_1 = s$, from $\mathsf{sign}\ m_1' = +$, we have m_1' is either $(s,0)$ or $(s,2)$. From $\mathsf{NSLInit}\ s\ A\ B\ Na\ Nb$, we have $\mathsf{term}\ (s,0) = \{\!|\mathsf{Nonce}\ Na, \mathsf{Agent}\ A|\!\}_{pubK\ B}$, and $\mathsf{term}\ (s,2) = \{\!|\mathsf{Nonce}\ Nb|\!\}_{pubK\ B}$. From $Na \neq Nb$ and the definition of G, we have that $\{\!|\mathsf{Nonce}\ Na, \mathsf{Agent}\ A|\!\}_{pubK\ B} \in \mathsf{synth}\ G$ and $\{\!|\mathsf{Nonce}\ Nb|\!\}_{pubK\ B} \in \mathsf{synth}\ G$. Then $\mathsf{term}\ m_1' \in \mathsf{synth}\ G$. But this contradicts with $\mathsf{term}\ m_1' \notin \mathsf{synth}\ G$. Therefore, $s \neq \mathsf{strand}\ m_1'$. Then by Lemma 18, we have $\mathsf{sign}\ m_1 = -$.

By (a) and the NSL specification, m_1' is in either an initiator or a responder strand. If the former case holds, then by the definition of a NSL initiator, m_1 is the second node in the strand $\mathsf{strand}\ m_1'$, because the only negative node in the strand is the second one. So $\mathsf{term}\ m_1 = \{\!|\mathsf{Nonce}\ Na', \mathsf{Nonce}\ Nb', \mathsf{Agent}\ B'|\!\}_{pubK\ A'}$ for some A', B', Na', and Nb'. From the facts $\mathsf{term}\ m_1 \in \mathsf{synth}\ G$ and $a \sqsubset \mathsf{term}\ m_1$, by (1) we have $\mathsf{term}\ m_1 \in G_0$, $A' = A$, $B' = B$, $Na' = Na$, so $\mathsf{NSLInit}\ (\mathsf{strand}\ m')\ A\ B\ Na\ Nb'$, with the assumption $\mathsf{NSLInit}\ s\ A\ B\ Na\ Nb$. By Lemma 23, we have $Nb' = Nb$, $(\mathsf{strand}\ m') = s$. Then by $m_1 \Rightarrow^+ m_1'$, we can derive that m_1' is the last node of the strand s, so it must be $\mathsf{term}\ m_1' = \{\!|\mathsf{Nonce}\ Nb|\!\}_{pubK\ B}$. By $\mathsf{Nonce}\ Na \sqsubset \mathsf{term}\ m_1'$, $Na = Nb$. This contradicts the assumption $Na \neq Nb$.

If the latter case holds, then $\mathsf{strand}\ m_1$ is a responder strand, by NSL specification about a responder, m_1' is the second node in the strand $\mathsf{strand}\ m_1'$ because the only positive node in an NSL responder strand is the second one. By $m_1 \Rightarrow^+ m_1'$, m_1 is the first node in the responder strand. Hence, $\mathsf{term}\ m_1 = \{\!|\mathsf{Nonce}\ Na', \mathsf{Agent}\ A'|\!\}_{pubK\ B'}$ for some A', B', and Na'. From the facts $\mathsf{term}\ m_1 \in \mathsf{synth}\ G$ and $a \sqsubset \mathsf{term}\ m_1$, and by (2), $B' = B$, $Na' = Na$, $A' = A$. So $\mathsf{term}\ m_1' = \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ N, \mathsf{Agent}\ B|\!\}_{pubK\ A'}$ for some N. Obviously, $m_1' \in \mathsf{node}\ b$, by the definition of G_0, we can conclude that $\mathsf{term}\ m_1' \in G_0$. Hence, $\mathsf{term}\ m_1' \in \mathsf{synth}\ G$. This contradicts the fact $\mathsf{term}\ m_1' \notin \mathsf{synth}\ G$.

Therefore, we have $\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A} \in \mathsf{synth}\ G$. From this, by (1), we have $\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A} \in G_0$, and by the definition of G_0, it is easy to show the conclusion of the lemma holds. $\square$

Next lemma is on the secrecy of the nonce Nb which is originated by a responder.

Lemma 25 (Nb's secrecy).

$[\![b \in \mathsf{bundles}; \mathsf{NSLResp}\ r\ A\ B\ Na\ b; \mathsf{uniquely\_originate}\ (\mathsf{Nonce}\ Nb)\ (r, 1)$
$(r, 2) \in \mathsf{nodes}\ b; A \notin \mathsf{bad}; B \notin \mathsf{bad}; Na \neq Nb]\!] \Longrightarrow$
$\forall n \in \mathsf{nodes}\ b.\mathsf{term}\ n \neq Nb$

Proof. We define $G = \{\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A}, \{\!|\mathsf{Nonce}\ Nb|\!\}_{pubK\ B}\} \cup \{t \mid \mathsf{Nonce}\ Nb \not\sqsubset t\}$ and $n = (r, 1)$.

By the specification of $\mathsf{NSLResp}$, we know that $\mathsf{term}\ (r, 1) = \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A}$ and $\mathsf{term}\ (r, 1) \in \mathsf{synth}\ G$. Similar to the proof in Lemma 24, by Lemma 14 and Lemma 15, we have two auxiliary results.

(1) $[\![t = \{\!|\mathsf{Nonce}\ Na', \mathsf{Nonce}\ Nb', \mathsf{Agent}\ B'|\!\}_{pubK\ A'}; t \in \mathsf{synth}\ G; \mathsf{Nonce}\ Nb \sqsubset t]\!] \Longrightarrow Na' = Na \wedge Nb' = Nb \wedge B' = B \wedge A' = A$

(2) $[\![t \in \mathsf{synth}\ G; \mathsf{Nonce}\ Nb \sqsubset t]\!] \Longrightarrow \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A} \sqsubset t\ \vee\ \{\!|\mathsf{Nonce}\ Nb|\!\}_{pubK\ B} \sqsubset t$

We show that (a) for any node $n \in \mathsf{nodes}\ b$, then $\mathsf{term}\ n \in \mathsf{synth}\ G$. We prove it by contradiction. Suppose there is a node n' such that $\mathsf{term}\ n' \notin \mathsf{synth}\ G$. It is rather routine to show that all assumptions of Lemma 18 can be satisfied, so (b) there are two regular nodes m_1' and m_1 such that $n \preceq_b m_1$, $m_1' \preceq_b n'$, $m_1 \Rightarrow^+ m_1'$, $a \sqsubset \mathsf{term}\ m_1$, $\mathsf{term}\ m_1 \in \mathsf{synth}\ G$, $a \sqsubset \mathsf{term}\ m_1'$, $\mathsf{term}\ m_1' \notin \mathsf{synth}\ G$, $\mathsf{Is\_Regular\_strand}\ (\mathsf{strand}\ m_1')$ and $\mathsf{sign}\ m_1' = +$.

Here we prove that m_1' is not in the strand r. We prove it by contradiction. Suppose m_1' is in r, then m_1' can only be the second node $(r, 1)$ because the only positive node is $(r, 1)$. But by the specification of $\mathsf{NSLResp}$, we have $\mathsf{term}\ (r, 1) = \{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A}$. Then $\mathsf{term}\ (r, 1) \in \mathsf{synth}\ G$. This contradicts with $\mathsf{term}\ m_1' \notin \mathsf{synth}\ G$. Hence, $r \neq \mathsf{strand}\ m_1'$. By this and Lemma 18, we have $\mathsf{sign}\ m_1 = -$.

By (b) and NSL specification, m_1' is either an initiator strand or a responder strand. If the former case holds, then by the definition of a NSL initiator, the only one negative node in a NSL initiator strand is the second one, so $\mathsf{term}\ m_1 = \{\!|\mathsf{Nonce}\ Na', \mathsf{Nonce}\ Nb', \mathsf{Agent}\ B'|\!\}_{pubK\ A'}$ and $\mathsf{term}\ m_1' = \{\!|\mathsf{Nonce}\ Nb'|\!\}_{pubK\ B'}$ for some A', B', Na', and Nb'. From $\mathsf{term}\ m_1 \in \mathsf{synth}\ G$ and $\mathsf{Nonce}\ Nb \sqsubset \mathsf{term}\ m_1$, and by (1), we have $Na' = Na$, $Nb' = Nb$, $B' = B$ and $A' = A$. By the fact $\mathsf{Nonce}\ Nb \sqsubset \mathsf{term}\ m_1'$, we have $Nb' = Nb$. So $\mathsf{term}\ m_1' = \{\!|\mathsf{Nonce}\ Nb|\!\}_{pubK\ B} \in \mathsf{synth}\ G$. This contradicts with $\mathsf{term}\ m_1' \notin \mathsf{synth}\ G$.

If the latter case holds, then $\mathsf{strand}\ m_1$ is a responder strand, by NSL specification of a responder, m_1' is the second node in the strand $\mathsf{strand}\ m_1'$ because the only positive node in a NSL responder strand is the second one. By $m_1 \Rightarrow^+ m_1'$, m_1 is the first node in the responder strand. Then $\mathsf{term}\ m_1 = \{\!|\mathsf{Nonce}\ Na', \mathsf{Agent}\ A'|\!\}_{pubK\ B'}$ for some A', B', and Na'. From the fact $\mathsf{Nonce}\ Nb \sqsubset \mathsf{term}\ m_1$, we have $Na' = Nb$. From $\mathsf{term}\ m_1 \in \mathsf{synth}\ G$, and by (2), we have that (c) $\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ B|\!\}_{pubK\ A} \sqsubset \mathsf{term}\ m_1$ or $\{\!|\mathsf{Nonce}\ Nb|\!\}_{pubK_B} \sqsubset \mathsf{term}\ m_1$. But from $\mathsf{term}\ m_1 = \{\!|\mathsf{Nonce}\ Na', \mathsf{Agent}\ A'|\!\}_{pubK\ B'}$ and (c), we can derive a contradiction.

From (a) and the definition of G and synth, we can prove that for any node $n \in \mathsf{nodes}\ b$, $\mathsf{term}\ n \neq \mathsf{Nonce}\ Nb$. $\square$

Analyzing NSL protocol took us only 3 days. The proof scripts comprises 1954 lines, and executes in 20 seconds.

5.3. Example 2: the Otway-Rees Protocol

The Otway_Rees protocol [OR87] (see Figure 8) assumes a shared-key environment. In Figure 8, we abbreviate $\mathsf{Agent}\ A$ as A, $\mathsf{Nonce}\ Na$ as Na, and $\mathsf{shrK}\ A$ as K_A. The aim of the protocol is to use long-term symmetric keys shared with the server and its clients to distribute a new session key for a conversation between two clients. There are three roles in the protocol: initiator, responder and server. The strands of the agents acting as responder and server are defined as follows:

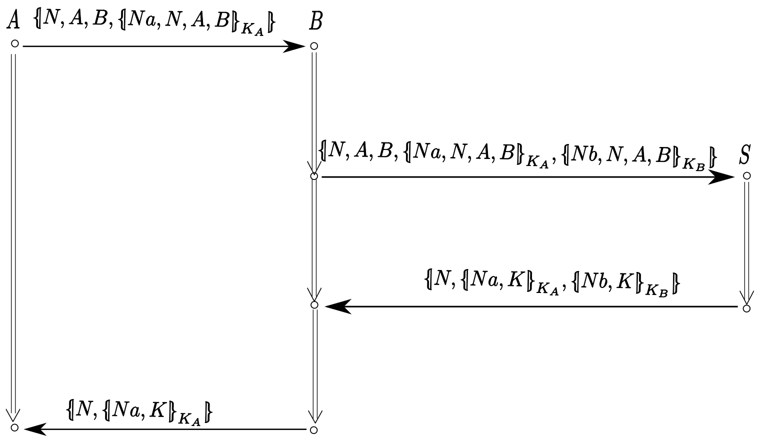
A
B
S
$\{\!|N, A, B, \{\!|Na, N, A, B|\!\}_{K_A}|\!\}$
$\{\!|N, A, B, \{\!|Na, N, A, B|\!\}_{K_A}, \{\!|Nb, N, A, B|\!\}_{K_B}|\!\}$
$\{\!|N, \{\!|Na, K|\!\}_{K_A}, \{\!|Nb, K|\!\}_{K_B}|\!\}$
$\{\!|N, \{\!|Na, K|\!\}_{K_A}|\!\}$

Fig. 8. Message exchanging in the Otway-Rees Protocol.

```
constdefs ORResp::sigma⇒agent⇒ agent⇒nat⇒msg⇒key⇒ msg⇒msg⇒bool
ORResp A B Nb N K H H' ≡ N∈T ∧ (N≠Nonce Nb) ∧(¬Nonce Nb⊏H) ∧
          ⎡ (−,{|N, Agent A,Agent B,H|}),                                          ⎤
(SP s)=   ⎢ (+,{|N, Agent A,Agent B,H,Crypt (shrK B) {|Nonce Nb,N,Agent A,Agent B|}|}), ⎥
          ⎢ (−,{|N,H′,Crypt (shrK B){|Nonce Nb,Key K|}|}),                            ⎥
          ⎣ (+,{|N,H′|})                                                           ⎦

constdefs ORServ::sigma⇒agent⇒ agent⇒nat⇒nat⇒msg⇒ key⇒bool
ORServ s A B Na Nb N K ≡ (∀A.K≠shrK A) ∧ N∈T ∧
          ⎡ (−,{|N, Agent A, Agent B,                                                  ⎤
(SP s)=   ⎢        Crypt (shrK A) {|Nonce Na, N, Agent A, Agent B|},                     ⎥
          ⎢        Crypt (shrK B) {|Nonce Nb, N, Agent A, Agent B|} |}),                  ⎥
          ⎣ (+, {|N, Crypt (shrK A) {|Nonce Na, Key K|},Crypt (shrK B) {|Nonce Nb,Key K|}|}) ⎦
```

Note that N is an atomic message such as a nonce to identify a session of the protocol. Namely, N is of type `msg`.

Next we define the strand space of Otway-Rees protocol to be the union of Otway-Rees initiator or Otway-Rees responder or Otway-Rees server or penetrator strands.

```
                  ⎧    ⎛ ∃A B Na N K. ORInit s A B Na N K ∨              ⎞⎫
defs OR_def:Σ ≡   ⎨ s. ⎜ ∃A B Nb N K H H′. ORResp s A B Nb N K H H′ ∨    ⎟⎬
                  ⎪    ⎜ ∃A B Na Nb N K′. ORServ s A B Na Nb N K ∨       ⎟⎪
                  ⎩    ⎝ Is_Penetrator_strand s                          ⎠⎭
```

Unsolicited authentication test was applied to prove a server's guarantee in the Otway-Rees protocol in [OR87]. But their proofs in [GJ02] for an initiator's and a responder's authentication guarantee depend on the result of outgoing authentication tests and a side assumption that no proper encrypted subterms are contained in the forwarding component H, which is corresponding to $\{\!|N, \mathsf{Nonce}\ Na, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A}$ in the first and second messages of a responder (see Figure 8). However, this side assumption is not realistic since a responder cannot enforce such a constraint. In the intended case, H is a term encrypted by the initiator's long-term key, which is unintelligible to the responder. To remedy this deficiency, the authors devoted one section (Section 5.1.3, [GJ02]) in their paper to show that this constraint does not hide any attacks. In particular, if the penetrator can succeed without this restriction, then they can also succeed if this constrain is enforced. Their proof is rather complicated. First, they need to introduce another notion of *nearly equivalence* between a constrained Otway-Rees bundle and an unconstrained Otway-Rees bundle.

Second, they need an intermediate result showing that a *nearly equivalent* constrained Otway-Rees bundle can be constructed from a unconstrained Otway-Rees bundle.

Now we only make use of results of unsolicited authentication tests and give simpler proofs for authentication goals of the Otway-Rees protocol. The proofs differ from the proofs in [GJ02] in several ways.

- We use unsolicited authentication tests to prove regularity of nodes in Otway-Rees protocol, namely that once $\{\!|h|\!\}_K$ occurs as a subterm of a node n in a bundle b, and if the key K cannot be penetrated in this bundle, then there is a regular node m originating $\{\!|h|\!\}_K$. Moreover, we frequently use the additionally strengthened assertion that m is the node originating $\{\!|h|\!\}_K$ as a subterm, and $\{\!|h|\!\}_K \not\sqsubset \mathsf{term}\ m'$ for any node $m' \preceq_b m$. This strengthened property turns out to be very useful in our analysis.
- Combining freshness property of a nonce in Otway-Rees protocol with unsolicited authentication test, we give an alternative proof for the initiator's and responder's authentication guarantees. In particular, we do not need the aforementioned side assumption and outgoing authentication tests in [GJ02]. Our main result lies in that we extend the result of a server's guarantees, in turn which can be used to prove the guarantees of an initiator and a responder.

In order to prove the main results of authentication guarantees, we need some auxiliary results first. For any node $n \in \mathsf{nodes}\ b$, $\mathsf{term}\ n$ cannot be a long-term symmetric key of a regular agent, because such a key is not sent as a part of a message in the Otway-Rees protocol. The first requirement of the definition $\mathsf{ORServ}[s, A, B, Na, Nb, N, K]$, which eliminates the possibility that a penetrable key occurs in the second node of s.

Lemma 26. $[\![b \in \mathsf{bundles}; n \in \mathsf{nodes}\ b; A \notin \mathsf{bad}]\!] \Longrightarrow \mathsf{Key}\ (\mathsf{shrK}\ A) \not\sqsubset \mathsf{term}\ n$

Following this lemma, it is easy to prove that a long-term symmetric key of a regular agent cannot be penetrated in the bundle.

Lemma 27. $[\![b \in \mathsf{bundles}; A \notin \mathsf{bad}]\!] \Longrightarrow \mathsf{regularK}\ (\mathsf{shrK}\ A)\ b$

As in [GJ01], we assume a nonce originates uniquely in some strand space. If a nonce Na originates uniquely, and $\mathsf{ORInit}\ i\ A\ B\ Na\ N\ K$, then the nonce can uniquely identify this strand i, which means if another initiator strand i' satisfies $\mathsf{ORInit}\ i'\ A'\ B'\ Na\ N'\ K'$, then $i = i'$. This is captured by the following lemma.

Lemma 28.

$$[\![\mathsf{uniquely\_originate}\ (\mathsf{Nonce}\ Na)\ (i, 0); \mathsf{ORInit}\ i\ A\ B\ Na\ N\ K; \mathsf{ORInit}\ i'\ A'\ B'\ Na\ N'\ K']\!]$$
$$\Longrightarrow i = i' \wedge A = A' \wedge B = B' \wedge N = N' \wedge K = K'$$

Now we come to prove the server's authentication guarantees. Our main technique is Lemma 19 in Section 4.9. The main differences between our proofs and the original proof of Guttman *et al.* lie in the guarantees of the existence of a server strand s such that $\mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K$. They only analyzed the case when $A \neq B$, while we consider more general cases without the restriction $A \neq B$. We show that if there is a server s such that $\mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K$ in b and A is regular, then there is a regular initiator i such that $\mathsf{ORInit}\ i\ A\ B\ Na\ N\ K'$ for some K', or a regular responder r such that $\mathsf{ORResp}\ r\ A\ B\ Na\ N\ H\ H'$ for some H, H' with $A = B$.

Lemma 29 (Server's guarantee 1).

$$[\![b \in \mathsf{bundles}; A \notin \mathsf{bad}; \mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K; (s, 0) \in \mathsf{nodes}\ b]\!]$$
$$\Longrightarrow \begin{pmatrix} (\exists i\ K'.\mathsf{ORInit}\ i\ A\ B\ Na\ N\ K' \wedge (i, 0) \in \mathsf{nodes}\ b) \vee \\ (\exists r\ K'\ H\ H'.\mathsf{ORResp}\ r\ A\ B\ Na\ N\ K'\ H\ H' \wedge A = B\ \wedge (r, 1) \in \mathsf{nodes}\ b) \end{pmatrix}$$

Proof. Suppose we have $n = (s, 0)$, and term of $(s, 0)$ is of the following form :$\{\!|N, \mathsf{Agent}\ A, \mathsf{Agent}\ B, \{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A}, \{\!|\mathsf{Nonce}\ Nb, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ B}|\!\}$.

By Lemma 27 and that $A \notin \mathsf{bad}$, $\mathsf{shrK}\ A$ is regular. So n is a unsolicited test for $\{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A}$. Therefore, by Lemma 19 there is a positive regular node m such that $\{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{\mathsf{shrK}\ A} \sqsubset \mathsf{term}\ m$, and $\{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A} \not\sqsubset \mathsf{term}\ m'$ for all m' such that $m' \prec_b m$.

By the trace form of regular strands, we have either (1) m is in an initiator strand i such that (for some

A', B', Na', N', K') $\mathsf{ORInit}\ i\ A'\ B'\ Na'\ N'\ K'$, or (2) m is in a responder strand r such that (for some $A', B', Nb', N', K', H, H'$) $\mathsf{ORResp}\ r\ A'\ B'\ Nb'\ N'\ K'\ H\ H'$.

If (1) holds, then by inspection on the trace form of an initiator strand, we have $m = (i, 0)$, and $\{\!|\mathsf{Nonce}\ Na', N', \mathsf{Agent}\ A', \mathsf{Agent}\ B'|\!\}_{shrK\ A'} = \{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A}$, then $Na' = Na$, $N' = N$, $A' = A$ and $B' = B$.

If (2) holds, then by inspection on the trace form of a responder strand, we have either $m = (r, 1)$ or $m = (r, 3)$. $m = (r, 3)$ is not possible. Otherwise, $\{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A} \sqsubset H'$. However, H' also occurs in $(r, 2)$. We have $m = (r, 1)$, then either (i) $\{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A} \sqsubset H$ or (ii) $\{\!|\mathsf{Nonce}\ Nb', N', \mathsf{Agent}\ A', \mathsf{Agent}\ B'|\!\}_{shrK\ B'} = \{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A}$. (i) is not possible, since H also occurs in $(r, 0)$. So (ii) must hold, then we have $Nb' = Na$, $N' = N$, $A' = A$, $B' = B$ and $shrK\ B' = shrK\ A$. By the injectivity of $shrK$, we have $B' = A$. Then $A = B$. □

Note that if we strengthen the assumptions of Lemma 29 with $A \neq B$, then the second case of the conclusion of Lemma 30 can be excluded.

Lemma 30 (Server's guarantee to an initiator).

$$\begin{array}{l}[\![b \in \mathsf{bundles};\ A \notin \mathsf{bad}; \mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K; (s, 0) \in \mathsf{nodes}\ b; A \neq B]\!]\\ \Longrightarrow (\exists i\ K'. \mathsf{ORInit}\ i\ A\ B\ Na\ N\ K' \wedge (i, 0) \in \mathsf{nodes}\ b)\end{array}$$

Similar to Lemma 29, we also prove another server's guarantee using $\{\!|\mathsf{Nonce}\ Nb, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ B}$ as an unsolicited test. By the assumption that a server s such that $\mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K$ exists in a bundle b and the fact that $\mathsf{shrK}\ B$ is regular, there is a regular responder r such that $\mathsf{ORResp}\ r\ A\ B\ Nb\ N\ K\ H\ H'$ for K, H, H', or a regular initiator i such that $\mathsf{ORInit}\ i\ A\ B\ Nb\ N\ K'$ for some K' with $A = B$.

Lemma 31 (Server's guarantee 2).

$$\begin{array}{l}[\![b \in \mathsf{bundles}; B \notin \mathsf{bad}; \mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K; (s, 0) \in \mathsf{nodes}\ b]\!]\\ \Longrightarrow \left(\begin{array}{l}(\exists r\ K\ H\ H'. \mathsf{OResp}\ r\ A\ B\ Nb\ N\ K\ H\ H' \wedge (r\ 1) \in \mathsf{nodes}\ b) \vee\\ (\exists i\ K.\ \mathsf{ORInit}\ i\ A\ B\ Nb\ N\ K \wedge A = B \wedge (i, 0) \in \mathsf{nodes}\ b)\end{array}\right)\end{array}$$

If we require $A \neq B$, we can also exclude the second part of the conclusion in Lemma 31.

Lemma 32 (Server's guarantee to a responder).

$$\begin{array}{l}[\![b \in \mathsf{bundles};\ B \notin \mathsf{bad}; \mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K; (s, 0) \in \mathsf{nodes}\ b; A \neq B]\!]\\ \Longrightarrow (\exists r\ K\ H\ H'. \mathsf{OResp}\ r\ A\ B\ Nb\ N\ K\ H\ H' \wedge (r\ 1) \in \mathsf{nodes}\ b)\end{array}$$

In order to prove the authentication guarantee of an initiator i such that $\mathsf{ORInit}\ i\ A\ B\ Na\ N\ K$ with $A \neq B$, we can use $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A}$ as an unsolicited test to prove the existence of a server s such that $\mathsf{ORServ}\ s\ A'\ B'\ Na'\ Nb\ N'\ K'$ for some A', B', Na', Nb, N', K'. Then with the above results of the guarantee of s, and the unique-origination of Na, we can ensure that $Na' = Na$, $K' = K$, $A' = A$, $N' = N$ and $B' = B$.

Lemma 33 (Initiator's guarantee).

$$\begin{array}{l}[\![b \in \mathsf{bundles}; A \notin \mathsf{bad}; A \neq B; \mathsf{ORInit}\ i\ A\ B\ Na\ N\ K; (i, 1) \in \mathsf{nodes}\ b; \mathsf{uniquely\_originate}\ (\mathsf{Nonce}\ Na)\ (i, 0)]\!]\\ \Longrightarrow \exists s\ Nb. \mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K \wedge (s, 1) \in \mathsf{nodes}\ b\end{array}$$

Proof. Suppose $n = (i, 1)$, term of $(i, 1)$ is $\{\!|N, \{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A}|\!\}$. $A \notin \mathsf{bad}$, by Lemma 27 $shrK\ A$ is regular. Hence, $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A}$ is an unsolicited test. By Lemma 19, there is a positive regular node m such that $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A} \sqsubset \mathsf{term}\ m$, and $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A} \not\sqsubset \mathsf{term}\ m'$ for all m' such that $m' \prec_b m$.

By the trace form of regular strands, m cannot be in an initiator's strand because no positive node has a subterm of the form $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A}$ in an initiator strand. If m is in a responder's strand, since a subterm of the form $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A}$ can only occur in the second or the forth nodes, we have $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A} \sqsubset H$ or $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A} \sqsubset H'$. However, neither H nor H' occurs as new in the strand. (H appears as a subterm of node $(r, 0)$, and H' appears as a subterm of node $(r, 2)$). So m is only in a server strand s such that $\mathsf{ORServ}\ s\ A'\ B'\ Na'\ Nb'\ N'\ K'$ for some $A\ , B', Na', Nb', N', K'$. By inspection on the trace form of a server strand, m can only be the second node in this strand, so either (1) $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A} \sqsubset \{\!|\mathsf{Nonce}\ Na', \mathsf{Key}\ K'|\!\}_{shrK\ A}$ or (2) $\{\!|\mathsf{Nonce}\ Na, \mathsf{Key}\ K|\!\}_{shrK\ A} \sqsubset \{\!|\mathsf{Nonce}\ Nb', \mathsf{Key}\ K'|\!\}_{shrK\ B'}$.

If (1) holds, then $Na' = Na$, $K' = K$, $A' = A$. We have $\mathsf{ORServ}\ s\ A\ B'\ Na\ Nb'\ N'\ K$. By Lemma 29, there exists either an initiator strand i' such that $\mathsf{ORInit}\ i'\ A\ B'\ Na\ N'\ K''$ for some K'' and $(i', 0) \in b$, or a responder strand r such that $\mathsf{ORResp}\ r\ A\ B'\ Na\ N'\ K''\ H\ H'$ for some K'', H and H' with $A = B'$, and $(r, 1) \in b$. We first prove the second case cannot hold. Suppose that there exists a responder strand r such that $\mathsf{ORResp}\ r\ A\ B'\ Na\ N'\ K''\ H\ H'$, then by the trace forms of a responder strand and an initiator strand, both $(i, 0)$ and $(r, 2)$ will be nodes originating Na, and this leads to a contradiction. So it can only be the case when there exists an initiator strand i' such that $\mathsf{ORInit}\ i'\ A\ B'\ Na\ N'\ K''$. Then by the facts $\mathsf{ORInit}\ i\ A\ B\ Na\ N\ K$ and $\mathsf{ORInit}\ i'\ A\ B'\ Na\ N'\ K''$, and by Lemma 28, we have $B' = B$, $N' = N$. Hence, $\mathsf{ORserv}\ s\ A\ B\ Na\ Nb'\ N\ K$ and $(s, 1) \in b$.

If (2) holds, then $Nb' = Na$, $K' = K$, $B' = A$. We have $\mathsf{ORserv}\ s\ A'\ A\ Na'\ Na\ N'\ K$. By Lemma 31, there exists either a responder strand r such that $\mathsf{ORResp}\ r\ A'\ A\ Na\ N'\ K''\ H\ H'$ for some K'', H and H' and $(r, 1) \in b$, or $i' \in \mathsf{ORInit}\ i'\ A'\ A\ Na\ N'$ with $A' = A$, and $(i', 0) \in b$. If the first case holds, then by the definition of a responder's trace and an initiator's trace, both $(i, 0)$ and $(r, 1)$ can be the node originating Na. This leads to a contradiction. If the second case holds, then by the facts $\mathsf{ORInit}\ i\ A\ B\ Na\ N\ K$ and $\mathsf{ORInit}\ i'\ A\ A\ Na\ N'\ K''$, then by Lemma 28, we have $B = A$. This contradicts with the assumption $A \neq B$. $\square$

Similarly, we can prove a responder's authentication guarantee.

Lemma 34 (Responder's guarantee).

$$\begin{array}{l} [\![b \in \mathsf{bundles};\ B \notin \mathsf{bad};\ A \neq B; \mathsf{ORResp}\ r\ A\ B\ Nb\ N\ K\ H\ H'; \\ (r, 2) \in \mathsf{nodes}\ b;\ \mathsf{uniquely\_originate}\ (\mathsf{Nonce}\ Nb)\ (r, 1)]\!] \\ \Longrightarrow \exists s\ Na. \mathsf{ORServ}\ s\ A\ B\ Na\ Nb\ N\ K \wedge (s, 1) \in \mathsf{nodes}\ b \end{array}$$

To sum up, we mainly use unsolicited tests and the freshness property of nonces to derive the above proofs of authentication guarantees. Here, we emphasize that we use Lemma 19 by asserting the existence of a regular node m which originates $\{\!|h|\!\}_K$. So for any n such that $n \prec_b m$, $\{\!|h|\!\}_K$ is not a subterm of m. We frequently use this in the above proofs to ensure that a node can only be in an intended regular node. For example, we use this result to prove that the node which originates $\{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A}$ can only be the second node if it is in a responder strand (Lemma 29). Besides, Lemmas 30, 32, 33, and 34 prove that the Otway_Rees protocol actually achieves the authentication goals when we require that an initiator A and a responder B cannot be the same agent in one session. We observe that the protocol does not establish that the same key is delivered to both A and B, only that if either A or B reaches the end of its strand, then the other has submitted the expected matching original request $\{\!|\mathsf{Nonce}\ Nb, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ B}$ or $\{\!|\mathsf{Nonce}\ Na, N, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}_{shrK\ A}$. These are security properties as explored in [JHG99, GJ01].

Analyzing the Otway_Rees protocol took us about one week's effort. This time is longer because for this protocol we need more time to figure out a new proof strategy during our proof procedure. The proof script comprises 3421 lines and executes in 30 seconds.

6. Related Work

Besides the inherence and extension from the classical work on strand space [JHG98, JHG99], our work is also related to the work by Perrig and Song [Son99, PS00]. They explore automatic verification techniques based on the strand space model. Athena [Son99], which is an automatic tool for security protocol analysis, has incorporated the authentication tests [GJ01]. Several efficient methods have been developed in order to increase the performance of Athena. In work [PS00, Gut02] authentication tests are employed for automatic security protocol design. In our work, instead of supporting automatic generation of security protocols, we emphasize the soundness of the strand space theory itself, and have developed an extension of authentication tests and formalized the theory in Isabelle/HOL [NPW02], which can be seen as another mechanical framework for proving the correctness of authentication protocols.

Jacobs and Hasuo [JH09] use the theorem prover PVS [ORS92] for reasoning about security protocols, where they use strand spaces as semantics for a BAN-like logic. This method does not employ the key techniques of the strand space theory, such as well-foundedness of bundles and authentication tests.

More recent work has been devoted to extensions of the classical strand space theory and applications of the theory to some real-world security protocols. For instance, the strand space theory is extended with a new sub-term relation to specify syntax of messages with MAC (Message Authentication Code) payload for security protocols [WZL05], and the bounds on the penetrator's behaviors are expanded accordingly. Sharp and Hansen combine strand space theory with interval logics to develop a timed strand space theory for analyzing real-time properties of security protocols in [SH07]. Kamil and Lowe adopt the strand space theory to specify and model the secure channel in layered protocols, where an application layer protocol is layered on top of a secure transport protocol in [KL10]. An IEEE-standard protocol used in local area wireless network, 802.11i, is modeled and verified using the strand space framework in [FMG06].

Our work is also closely related to Paulson's work [Pau98]. We have directly imported their theory on messages, and used their inductive method. Two inductively defined operators on messages parts and synth are borrowed to formally introduce the subterm relation and *test suite*. Besides, Isabelle/HOL's built-in support for inductive set and rule induction makes it convenient to apply induction proof method. However, our work differs from theirs in the following aspects. (a) We model the semantics of a protocol by strands, and mainly use the well-founded principle of a bundle and authentication tests to prove properties; while they model the semantics of a protocol by an inductively defined set of traces and mainly use the induction principle to directly prove properties. (b) The inductive approach is borrowed to give a more tractable definition for a general bundle, which is independent of a protocol case; while the inductive approach is used to define a special protocol case in [Pau98], and different protocols have different inductive definitions. (c) Authentication tests, which are our main proof techniques for security protocols, are general results and can be applied in the verification of many protocol cases. However, no such general principles are available because each protocol has its own inductive definition in [Pau98], and therefore induction proofs for the similar security properties of two protocols differ from each other because the two protocols have different induction definitions

This paper is also related to our previously published results [Li05, LP06]. In this paper, we have developed a new result on characterization of a penetrator's ability to deduce knowledge from a message set, i.e., the so-called *test suite*. This result is crucial to prove our version of generalized authentication tests which is not included in [Li05, LP06] The formalization of generalized authentication tests is revised to a much simpler form.[2] We consider the simplification of the proof of the authentication tests an improvement upon the classical strand space theory. Moreover, the advantages of proof techniques of Isablelle/Isar are illustrated in the current paper, which are adopted to have mechanical support for the strand space theory. In particular, we show how the induction method and calculation reasoning are applied in Isar.

7. Conclusion

The key contribution of our work is to introduce an inductive approach to the strand space theory. Essentially, the inductive approach allows us to define a protocol session set by induction rules and reason about this set by the corresponding induction principles. Usually, the induction rules can specify the semantics in protocol execution steps. Each element of the set is a finite object, but the set might have unbounded size. Although a bundle has one finite node set and one finite edge set, the size of the set of all the bundles is unbounded. The inductive approach is the most effective to reason about properties of each element in such a set.

- Firstly, the inductive approach is an effective technology to improve the strand space theory. The inductively defined operators parts and synth are borrowed from Paulson's work [Pau98] to characterize the semantics of actions on messages in strand spaces. In particular, the synth operator is used to characterize penetrator's ability to deduce knowledge from a message set. The inductive definition of bundles formalizes the operational semantics of protocol steps, and the induction principles on bundles helps us a lot to formally prove that there is a path in a bundle from the node at which a message originates to a node where the message occurs.
  Combining the aforementioned two results of characterization of penetrator's ability and the existence of a path from a messages's originating node to a node where the message occurs, we give a generalized version of authentication test and a simpler proof.
- Secondly, the inductive approach is a strong tool to mechanize all the theory in a theorem prover which makes strand space theory more practical. Our proposal is easy to be mechanized in any theorem prover provided that it has support for inductively defined sets and the corresponding automatic reasoning

[2] Hence, its proof in the current paper is a lot different from [LP06]. Previously, our proof relies on the well-foundedness of a bundle and is rather complex.

tools. Mechanizing all the theory in a theorem prover not only models protocols rigorously and specifies protocol goals without any ambiguity, but also guarantees a formal proof. Therefore, it helps us to achieve the highest possible assurance for formal reasoning of security protocols. We believe that our work is the first that thoroughly formalizes the strand space theory.

In the future, we want to extend our research in two directions. (1) We want to add more features into our approach to tackle modern security protocols. Currently, we have focused on timestamp, Diffie-Hellman operations, and protocol composition. (2) We want to automate (or semi-automate) our approach. By inspection on our case studies, we find that proof procedures of applying authentication tests are rather routine and modular. We need to figure out a proper test nonce, a test edge, and a proper term set G (in Lemma 18) to ensure that an agent uses this test to send a term in $\mathsf{synth}\ G$, but receives it in a transformed form which is not contained in $\mathsf{synth}\ G$. After that we need to prove that a regular agent is due to this transformation. Moreover, combining the analysis of traces of regular strands and other side assumptions, it can be ensured that only the intended one can do so. The real creative part of such a proof lies in how to figure out G, which depends on the human insights for the protocol under consideration, and the other parts are routine. It is interesting to implement such a general proof strategy as a tactic in Isabelle/HOL.

Acknowledgements. We would like to thank the three anonymous referees who read carefully the first submission version of this paper and gave a lot of valuable comments, which help us to greatly improve the quality of our paper both in content and presentation. Christian Urban also tells us a lot how to present Isabelle formalization elegantly in a paper.

References

[BAN90] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[DY83] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(12):198–208, 1983.

[FMG06] Z. Furqan, S. Muhammad, and R. K. Guha. Formal verification of 802.11i using strand space formalism. In *Proc. Conference on Networking and Conference on Mobile Communications and Learning Technologies*, page 140. IEEE Computer Society, 2006.

[GJ01] J. D. Guttman and F. Javier Thayer. Authentication tests. In *Proc. 12th IEEE Symposium on Security and Privacy*, pages 96–109. IEEE Computer Society, 2001.

[GJ02] J. D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[Gut02] J. D. Guttman. Security protocol design via authentication tests. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 92–103. IEEE Computer Society, 2002.

[JH09] B. Jacobs and I. Hasuo. Semantics and logic for security protocols. *Journal of Computer Security*, 17(6):909–940, 2009.

[JHG98] F. Javier Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 19th IEEE Symposium on Security and Privacy*, pages 96–109. IEEE Computer Society, 1998.

[JHG99] F. Javier Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1):191–230, 1999.

[KL10] A. Kamil and G. Lowe. Specifying and modelling secure channels in strand spaces. In *Proc. 6th Workshop on Formal Aspects in Security and Trust*, volume 5983 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2010.

[Li05] Y. Li. The inductive approach to strand space. In *Proc. 25th IFIP Conference on Formal Techniques for Networked and Distributed Systems*, volume 3731 of *Lecture Notes in Computer Science*, pages 547–552. Springer, 2005.

[Li08] Y. Li. Strand spaces and security protocols, 2008. `http://lcs.ios.ac.cn/~lyj238/strand.html`.

[Low96a] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[Low96b] G. Lowe. Some new attacks upon security protocols. In *Proc. 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society, 1996.

[LP06] Y. Li and J. Pang. Generalized unsolicited tests for authentication protocol analysis. In *Proc. 7th Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 509–514. IEEE Computer Society, 2006.

[LP07] Y. Li and J. Pang. Extending the strand space method to verify Kerberos v. In *Proc. 8th Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 437–444. IEEE Computer Society, 2007.

[Mea99] C. Meadows. Analysis of the internet key exchange protocol using the NRL protocol analyzer. In *Proc. 12th IEEE Computer Security Foundations Workshop*, pages 216–231. IEEE Computer Society, 1999.

[Mil95] J. K. Millen. The interrogator model. In *Proc. 16th IEEE Symposium on Security and Privacy*, pages 251–260. IEEE Computer Society, 1995.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *Proc. 18th Symposium on Security and Privacy*, pages 141–153. IEEE Computer Society, 1997.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic.* LNCS 2283. Springer, 2002.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Reviews*, 27(2):10–14, 1987.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proc. 11th Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer, 1992.

[Pau96] L. C. Paulson. *ML for the Working Programmer.* University of Cambridge Press, 1996.

[Pau97] L. C. Paulson. Proving properties of security protocols by induction. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society, 1997.

[Pau98] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[PS00] A. Perrig and D. X. Song. Looking for diamonds in the desert: Extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proc. 13th IEEE Computer Security Foundations Workshop*, pages 64–76. IEEE Computer Society, 2000.

[Sch97] S. Schneider. Verifying authentication protocols with CSP. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 3–17. IEEE Computer Society, 1997.

[SH07] R. Sharp and M. R. Hansen. Timed traces and strand spaces. In *Proc. 2nd Symposium on Computer Science in Russia*, volume 4649 of *Lecture Notes in Computer Science*, pages 373–386. Springer, 2007.

[Son99] D. X. Song. Athena: A new efficient automated checker for security protocol analysis. In *Proc. 12th IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE Computer Society, 1999.

[Wen99] M. Wenzel. Isar - A generic interpretative approach to readable formal proof documents. In *Proc. 12th Conference on Theorem Proving in Higher Order Logics*, volume 1690 of *Lecture Notes in Computer Science*, pages 167–184. Springer, 1999.

[WZL05] H. Wang, Y. Zhang, and Y. Li. Modeling for security verification of a cryptographic protocol with MAC payload. In *Proc. International Conference on Intelligent Computing*, volume 3645 of *Lecture Notes in Computer Science*, pages 538–547. Springer, 2005.

Appendix

In the appendix, we briefly present some Isabelle concepts, notations and commands, and our notation conventions for variables in our work. Readers will find these useful if they want to check our Isabelle proof script with our paper.

Isabelle's meta-logic is the intuitionistic fragment of Church's theory of simple types, which can be used to formalize an object-logic which we need [NPW02]. Normally, we use rich infrastructure of the object-logics such as HOL to formalize some theory, which has been provided by Isabelle system. Important connectives of the meta-logic are as follows: implication ($\Longrightarrow$) is for separating premises and conclusion of theorems; equality ($\equiv$) definitions; universal quantifier ($\bigwedge$) parameters in goals. In our work, we use the object-logic HOL to formalize the strand space theory. Therefore, we briefly show how to use HOL to formalize a theory.

Theories. Working with Isabelle means creating theories. A theory is a file with a named collection of types, functions, and theorems, proofs. The general format of a theory T is as follows:

theory $T = B_1 + B_2 + \ldots + B_n$;
declarations for types, definitions, lemmas, and proofs
end

where $B_1, B_2, \ldots, B_n$ are are the names of existing theories that T is based on. For our case, we only need import HOL library `Main` to write the theory `strand.thy`.

Types. There are basic types such as bool, the type of truth values; nat, the type of natural numbers. Function types are denoted by $\Rightarrow$, and product types by $\times$. Types can also be constructed by type constructors such as list and set. For instance, nat list declares the type of lists whose members are natural numbers.

Terms. Forms of terms used in this paper are rather simple. It is simply a constant or variable identifier, or a function application such as $f\ t$, where f is a function of type $\tau_1 \Rightarrow \tau_2$, and t is a term of type τ_1. Formulae are terms of type bool. bool has two basic constants True and False and the usual logical connectives (in decreasing order of priority): $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\forall$, and $\exists$, all of which (except the unary $\neg$) associate to the right. Note that the logical connectives introduced here are used in the object-logic HOL.

Introducing new types. There are three kinds of commands for introducing new types. typedecl *name* introduces new "opaque" type name without definition; types $name = \tau$ introduces an abbreviation name for type τ. datatype command can introduce a recursive data type. A general datatype definition is of the form

$$\mathsf{datatype}\ (\alpha_1, \ldots, \alpha_n) = C_1\ \tau_{11}\ \ldots\ \tau_{1k_1}\ \mid \ldots \mid\ C_m\ \tau_{m1} \ldots \tau_{mk_m}$$

where α_i are distinct type variables (the parameters), C_i are distinct constructor names and τ_{ij} are types. Note that n can be 0, i.e., there is no type parameters in datatype declaration.

Definition commands. consts command declares a function's name and type. defs gives the definition of a declared function. constdefs combines the effect of consts and defs. Combining a consts and inductive commands, we can give an inductive definition for a set. An inductively defined set S is typically of the following form:

consts $S :: \tau$set inductive S intros
$rule_1 : [\![a_{11} \in S; \ldots; a_{1k_1} \in S; A_{11}, \ldots, A_{1i_1}]\!] \Longrightarrow a_1 \in S$
...
$rule_n : [\![a_{n1} \in S; \ldots; a_{nk_n} \in S; A_{n1}, \ldots, A_{ni_n}]\!] \Longrightarrow a_n \in S$

Lemmas. In Isabelle's traditional style, we use the notation lemma $name : [\![A_1; A_2; \ldots; A_n]\!] \Longrightarrow B$ to donote that with assumptions $A_1, \ldots, A_n$, we can derive a conclusion B. In Isar's style, a lemma is written as lemma $name$: assumes a_1 : "A_1" and ... and a_n : "A_n" shows B.

Proof scripts and proof states. In Isabelle's traditional tactics-based style, a proof script comprises a sequences of application of tactics: apply $tac_1, \ldots,$ apply tac_n done. The script will be executed by Isabelle system until all subgoals are solved. A typical proof in Isar style has a more human-readable structure as follows:

```
proof
assume $asm_0$ and ... and $asm_m$
have $formula_1$
    proof ....(*proof script for $formula_1$*) qed
...
have $formula_n$
    proof ....(*proof script for $formula_n$*) qed
have $formula_{n+1}$
    proof ....(*proof script for $formula_{n+1}$*) qed
qed
```

where $asm_0, \ldots, asm_m$ are assumptions, $formula_1, \ldots, formula_n$ are intermediate results. From assumptions and intermediate results, we can show the final goal $formula_{n+1}$.

Techniques such as case distinction, induction, calculational reasoning in Isar language can make our proof more structured, which are immensely more readable and maintainable than apply-scripts. But the price we pay is that the length of proof in Isar is usually much longer than that of the counterpart in Isabelle tactical style. Therefore, we adopt a mixed style in our formalized proofs: we use commands in Isar style to decompose a large goal into subgoals to keep our proof with a clear structure; when a subgoal is simple enough, we directly use apply-scripts to prove the subgoal, thus we can keep the length of our proof script relatively short. After processing a proof command, Isabelle will display a proof state:

$$\bigwedge x_1, \ldots, x_p.\ [\![A_1; A_2; \ldots; A_n]\!] \Longrightarrow B$$

where $x_1, \ldots, x_p$ are local constants, $A_1, \ldots, A_n$ are local assumptions, and B is the actual (sub)goal in this proof state. Note that $\bigwedge$ is the universal quantifier in the meta-logic, not the conjunction operator ($\wedge$) in the object logic HOL.

Notation conventions. Throughout this paper, we use the conventions for meta-variables as follows:

b, c	range over bundles
m, n, x, y	range over nodes
i, j	range over natural numbers for indexes of nodes
f, g, h, t	range over messages
a	ranges over atomic messages of nonces and session keys
p	ranges over paths
A, B	range over agent names
K, Ka, Kb	range over keys
N, Na, Nb	range over nonces
G, H	range over essage sets
s, s'	range over strands
M	ranges over node sets

Form of lemma for induction. We have shown the formal notation of `bundles.induct` in Section 4.3. This rule is applied as an elimination rule. Isabelle unifies the first premise $xa \in bundles$ in `bundles.induct` with an assumption of a lemma, then eliminates the assumption, and new five subgoals (`Nil−Add_neg2`) will appear. Therefore, in order to allow the induction to go through, we must conform the lemma to a form which can be matched for this induction principle. That is to say, we usually write the form of a lemma as

$$[\![b \in \mathsf{bundles}]\!] \Longrightarrow P\ b$$

if we want to show a property P holds for each bundle. After proving a lemma by induction, sometimes we need to turn the lemma to a new form, which is presented as a new lemma for further application. For instance, for Lemma 3, we firstly need to prove a lemma $[\![b \in \mathsf{bundles}]\!] \Longrightarrow \mathsf{sign}\ n = - \longrightarrow n \in \mathsf{nodes}\ b \longrightarrow n' \to n \longrightarrow n' \in \mathsf{nodes}\ b \wedge (n', n) \in \mathsf{edges}\ b$ by induction. Then we use this lemma to prove Lemma 3, which is suitable for further application. Similarly, after proving Lemma 13, we can use it to prove a new lemma: $[\![b \in \mathsf{bundles};\mathsf{uniquely\_originate}\ g\ n; n' \in \mathsf{nodes}\ b; g \sqsubset \mathsf{term}\ n']\!] \Longrightarrow (\exists p.p \in (\mathsf{complete\_Path}\ g\ b) \wedge (\mathsf{last}\ p) = n')$, which is used in the proof of Lemma 4.8.