Semantics and Logic for Security Protocols

Bart Jacobs Ichiro Hasuo
Institute for Computing and Information Sciences, Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
Email: {B.Jacobs, I.Hasuo}@cs.ru.nl
URL: http://www.cs.ru.nl/{B.Jacobs, I.Hasuo}

October 20, 2006

Abstract

This paper presents a sound BAN-like logic for reasoning about security protocols with theorem prover support. The logic has formulas for sending and receiving messages (with nonces, public and private encryptions *etc.*), and has both temporal and epistemic operators (describing the knowledge of participants). The logic's semantics is based on strand spaces. Several (secrecy or authentication) formulas are proven in general and are applied to the Needham-Schroeder(-Lowe), bilateral key exchange and the Otway-Rees protocols, as illustrations.

1 Introduction

Security protocols are difficult to get right, and so a formal understanding of their meaning with associated reasoning techniques is an important topic since many years. Roughly two approaches have emerged, one based on algorithmic techniques using model checkers (such as [17, 3], see also [24]) and one on logical reasoning. This paper fits in the latter tradition. The algorithmic techniques are very good at detecting errors in relatively simple protocols in a "push button" style, but usually run into problems with the size of state spaces for more complicated protocols. In contrast, the techniques based on logical reasoning can in principle handle arbitrarily complex protocols, but may involve considerable user interaction. Hence, again in broad terms, algorithmic techniques are most useful early on in design, and logical techniques later on in certification.

This paper describes a formalisation of security protocols, involving a mathematical model in the style of strand spaces [27], on top of which a (sound) logic is defined that resembles BAN logic [4]. The whole formalisation is represented in the higher order logic of the theorem prover PVS [22], and allows verifications of actual protocols with tool support, like in [23]. Hence, in a sense, it combines the best of these three approaches [27, 4, 23].

The following aspects distinguish our formalisation.

- The logic involves both (linear) temporal operators (like henceforth) and epistemic ones (describing knowledge of participants).
- There is a clear distinction between possession (of messages) and knowledge (of logical assertions).

- There is a syntactic distinction between "new" nonces (or secret keys[1]) and "used" nonces, with an associated requirement that "new" nonces (or keys) are globally fresh. Hence freshness does not refer to a particular run of a protocol, simply because a run is a "meta" notion that is not explicitly present in our model.

This paper makes essential use of earlier related work [27, 4, 23, 25] but differs in several ways.

- The strand spaces of [27] form a mathematical model consisting of "bundles" of "strands" of incoming and outgoing messages for each participant in a security protocol. Here we formalise essential parts of these strand spaces in the language of a theorem prover and provide it with a logic.

- The virtue of BAN-style logical rules does not confine itself to their use in verification via theorem proving: they also tell a designer of a protocol how a protocol can get right, functioning as "rules of thumb". However, one of the most problematic aspects of the so-called BAN logic [4] is its lack of semantics. Here we approach the matter from a different angle: we do not start with a logic, but with a semantics and formulate logical rules as valid consequences in the formalised model.

  Hence, formally speaking we don't have a logic as a collection of (syntactical) formulas with a derivation relation. Our logic is "shallow" and exists only as provable implications or lemmas about (interpreted) formulas as predicates on the model. Such a "logic" is most convenient for the verification of concrete protocols in a theorem prover.

- Within the inductive approach [23] a new model is constructed for each protocol that is verified, namely as inductively defined set of lists of events. Here instead we have a semantical infrastructure of strands that is the same for each specific protocol. This allows us to prove more general logical rules, for instance about the sending or receiving of public or secret encryptions.

- The aim of the present paper is very similar to [25] (see also [26, Section 6.2]), namely to formulate a BAN-like logic on top of a strand space semantics. However, the reference [25] only contains a number of definitions for such a logic, without any rules, applications, or formalisations. Thus, one could say, the present paper achieves what is sketched in [25].

The use of a theorem prover in the formalisation of our model is for two reasons.

1. In general, a theorem prover is like a skeptical colleague who patiently checks all details. In the present setting this is useful because many proofs (esp. of the logical rules in Section 5) are complex, highly combinatorial, and involve many different case distinctions. In such situations humans easily make mistakes.

2. The present formalisation arose via several iterations, in which some subtle properties of the set of messages or of the semantical infrastructure were changed. The ability to re-run the proofs of existing results after such basic changes is very helpful to quickly see the consequences, and to maintain overall consistency. It will also be of use for possible future extensions and adaptations of the model.

[1] In this paper a key for symmetric encryption is called a *secret* key. A key pair for asymmetric encryption consists of a *public* key and a *private* key.

The fact that we used the theorem prover PVS is not especially relevant for the topic. We could also have used the higher order logic of the theorem provers Isabelle or COQ. The presentation in this paper abstracts from the concrete syntax of PVS and uses a more mathematical/logical style. However, readers who wish to see the details of the formalisation [15] will have to read PVS code.

The idea of building a BAN-like logic on top of a strand space model is not really new (see for instance [25]). The contribution of this paper is however, that this idea has been elaborated in full detail, been formalised, and put to use successfully in concrete examples. Doing so required a subtle balancing of the various possible requirements and formulations—and a non-trivial amount of PVS work!

We analyse three protocols using our toolkit: the Needham-Schroeder(-Lowe), the bilateral key exchange and the Otway-Rees protocols. Security properties of Otway-Rees is substantially harder to prove compared to for the first two, due to the presence of *tickets*. In this paper we first introduce some simpler secrecy/authentication rules and analyse the first two protocols. Those rules turns out to be of no use for Otway-Rees since the premises of those rules can hardly be verified: therefore we strengthen these rules in a later section.

Below we start in Section 2 with some fundamental results about the well-known Needham-Schroeder protocol. Their sole role at this stage is to illustrate the style of properties that can be proved. The explanation of the underlying theories starts in Section 3 with the theory of protocol messages, and proceeds with strands and bundles in Section 4, and with logical rules in Section 5. The bilateral key exchange protocol (from [5, Section 6.6.6]) is analysed in Section 6 as the second example. In Section 7 the verification challenge in the Otway-Rees protocol is identified and the logical rules are strengthened accordingly.

2 Results about the Needham-Schroeder public key protocol

Here we consider the well-known Needham-Schroeder public key protocol from [20], which was shown to be flawed in [16]. It can be described via the following three message exchanges between Alice (written as A, with public key K_A) and Bob (B with public key K_B).

$$\begin{array}{lcl} A \longrightarrow B & : & \{\, n_A, A \,\}_{K_B} \\ B \longrightarrow A & : & \{\, n_B, n_A \,\}_{K_A} \\ A \longrightarrow B & : & \{\, n_B \,\}_{K_B} \end{array}$$

Here we write n_A for a fresh (or new) nonce introduced by A, and similarly n_B for a fresh nonce of B.

What we can prove about this protocol is formulated as follows.

$$\begin{array}{l} A \ \mathsf{Sends}\ (\{\, \mathsf{newnonce}(n_A), \mathsf{name}(A) \,\}_{K_B})\ @\, i \\ \quad \wedge \\ A \ \mathsf{Sees}\ (\{\, \mathsf{nonce}(n_B), \mathsf{nonce}(n_A) \,\}_{K_A})\ @\, i+j \\ \quad \Longrightarrow \\ \mathsf{HenceForth}\ (\mathsf{Secret}(A,B)(\mathsf{nonce}(n_A)))\ @\, i+j \end{array}$$

In ordinary words: if Alice sends the first protocol message at stage i and then sees the second message at stage $i+j$, then her nonce n_A is a shared secret between her and Bob from stage $i+j$ onwards. Notice that Alice's nonce n_A is labeled with 'new' in its first use, and as a 'plain' nonce in subsequent use. Bob's nonce n_B is new when Bob sends it, but not anymore when Alice sees it.

An analogous result about Bob's nonce n_B is not provable because of Lowe's attack [16]. It can however be proved for the repaired "Needham-Schroeder-Lowe" protocol:

$$\begin{array}{lcl} A \longrightarrow B & : & \{\, n_A, A \,\}_{K_B} \\ B \longrightarrow A & : & \{\, n_B, n_A, B \,\}_{K_A} \\ A \longrightarrow B & : & \{\, n_B \,\}_{K_B} \end{array}$$

Then we can prove for instance the following non-trivial result.

$$\begin{array}{l} A \ \mathsf{Sends}\ (\{\, \mathsf{newnonce}(n_A), \mathsf{name}(A) \,\}_{K_B})\ @\, i \\ \quad \wedge \\ A \ \mathsf{Sees}\ (\{\, \mathsf{nonce}(n_B), \mathsf{nonce}(n_A), \mathsf{name}(B) \,\}_{K_A})\ @\, i+j \\ \quad \wedge \\ A \ \mathsf{Knows}\ (B \ \mathsf{Sees}\ (\{\, \mathsf{nonce}(n_B) \,\}_{K_B}))\ @\, i+j+k \\ \quad \Longrightarrow \\ A \ \mathsf{Knows}\ (B \ \mathsf{Knows}\ (\mathsf{HenceForth}\ (\mathsf{Secret}(A, B)(\mathsf{nonce}(n_B)))))\ @\, i+j+k \end{array}$$

It tells that if Alice first sends and sees the appropriate messages and then knows that Bob sees the final message, then Alice knows that Bob knows that his nonce n_B is henceforth a shared secret. The knowledge operator satisfies $(X\ \mathsf{Knows}\ (\varphi)) \Rightarrow \varphi$ so that the conclusion is pretty strong, and implies for instance the secrecy of the nonce. Note that Alice needs to know the fact that the final message was seen by Bob—a fact that she cannot directly observe herself—in order to draw conclusions about Bob's knowledge. This is quite natural.

In the remainder of this paper we shall explain what formulas like $A\ \mathsf{Sends}\ (m)\ @\, i$ or $B\ \mathsf{Knows}\ (\varphi)$ mean, and how they can be proved.

3 The theory of messages

The parties involved in the security protocols will be called agents. For the time being we shall assume a (parameter) type AG of agents, without any further structure. Only later on we shall assume a spy among the agents.

Messages are the identities that are exchanged between agents in a protocol. They may be nonces, keys, names, sequence numbers *etc.* The type MSG of messages is defined inductively, using the following BNF notation.

$$\begin{array}{rcl} m & ::= & \mathsf{name}(a) \mid \mathsf{number}(i) \mid \mathsf{newnonce}(n) \mid \mathsf{nonce}(n) \mid \mathsf{newseckey}(k) \mid \\ & & \quad \mathsf{seckey}(k) \mid \mathsf{pubkey}(k) \mid \mathsf{privkey}(k) \mid k\{\, m \,\} \mid \{\, m \,\}_k \mid \mathsf{hash}(m) \mid \langle m, m \rangle \end{array}$$

The identifier a in $\mathsf{name}(a)$ refers to an agent, where $\mathsf{name}(-)$ acts like a tag. The n in $\mathsf{nonce}(n)$ and $\mathsf{newnonce}(n)$ belong to an unspecified domain of nonces. Similarly for the different types of keys k, with a tag *e.g.* $\mathsf{pubkey}(-)$ making a key k into a message. The role of the 'new' will be explained later. We use the notation $k\{\, m \,\}$ for secret (symmetric) encryption and $\{\, m \,\}_k$ for public (asymmetric) encryption. There is an implicit function $\overline{\cdot}$ that sends a public key k to its associated private one $\overline{k}$. The notation $\langle -, - \rangle$ for tuples is often used implicitly, for instance in $k\{\, m_1, m_2 \,\}$. Since messages are freely generated tupling is not associative: $\langle \langle m_1, m_2 \rangle, m_3 \rangle \neq \langle m_1, \langle m_2, m_3 \rangle \rangle$. Signed messages are at this stage not supported.

The use of an inductively defined set of messages implicitly involves certain idealisations. For instance, hashes are assumed to be perfect, since by construction $\mathsf{hash}(m_1) =$

$\mathsf{hash}(m_2)$ implies $m_1 = m_2$. Also, the explicit use of the datatype's constructors excludes type flaw attacks (see *e.g.* [18, 14]).

A first basic function is $\mathsf{n2p} \colon \mathsf{MSG} \to \mathsf{MSG}$ for "new-to-plain". It erases 'new' recursively, and is defined in the obvious way:

$$\begin{array}{rclrcl}
\mathsf{n2p}(\mathsf{name}(a)) & = & \mathsf{name}(a) & \mathsf{n2p}(\mathsf{pubkey}(k)) & = & \mathsf{pubkey}(k) \\
\mathsf{n2p}(\mathsf{number}(i)) & = & \mathsf{number}(i) & \mathsf{n2p}(\mathsf{privkey}(k)) & = & \mathsf{privkey}(k) \\
\mathsf{n2p}(\mathsf{nonce}(n)) & = & \mathsf{nonce}(n) & \mathsf{n2p}(k\{\, m \,\}) & = & k\{\, \mathsf{n2p}(m) \,\} \\
\mathsf{n2p}(\mathsf{newnonce}(n)) & = & \mathsf{nonce}(n) & \mathsf{n2p}(\{\, m \,\}_k) & = & \{\, \mathsf{n2p}(m) \,\}_k \\
\mathsf{n2p}(\mathsf{seckey}(k)) & = & \mathsf{seckey}(k) & \mathsf{n2p}(\mathsf{hash}(m)) & = & \mathsf{hash}(\mathsf{n2p}(m)) \\
\mathsf{n2p}(\mathsf{newseckey}(k)) & = & \mathsf{seckey}(k) & \mathsf{n2p}(\langle m_1, m_2 \rangle) & = & \langle \mathsf{n2p}(m_1), \mathsf{n2p}(m_2) \rangle.
\end{array}$$

It is clear that $\mathsf{n2p}$ is idempotent, *i.e.* satisfies $\mathsf{n2p}(\mathsf{n2p}(m)) = \mathsf{n2p}(m)$. A message m is called "plain" if it contains no 'new', *i.e.* if $\mathsf{n2p}(m) = m$.

3.1 Subterms and paths

We use the symbol $\preceq$ for the syntactic subterm relation, that can also be defined inductively. It ignores encryptions or hashes, so that for instance $\mathsf{nonce}(n) \preceq \mathsf{hash}(\mathsf{nonce}(n))$, but not $\mathsf{seckey}(k) \preceq k\{\, m \,\}$. It is not hard to see that $\preceq$ is a partial order. Further results are: $m_1 \preceq m_2$ implies $\mathsf{n2p}(m_1) \preceq \mathsf{n2p}(m_2)$, and: $m_1 \preceq m_2$ with m_2 plain implies that m_1 is also plain; also: if $m_1 \preceq \mathsf{n2p}(m_2)$ then $m_1' \preceq m_2$ for some term m_1' with $\mathsf{n2p}(m_1') = m_1$.

It is useful to extend the subterm relation to subsets $U \subseteq \mathsf{MSG}$ of messages: $m \preceq U$ means that $m \preceq m'$ for some $m' \in U$.

We shall often need a more informative subterm relationship involving "paths". A path ℓ in $\ell \colon m_1 \rightsquigarrow m_2$ or $m_1 \overset{\ell}{\rightsquigarrow} m_2$ indicates how a term m_1 occurs as a subterm in m_2. Such a path is a list of labels from the set

$$\{\mathsf{se}(k), \mathsf{pe}(k), \mathsf{ha}, \pi_1, \pi_2\}$$

indicating how the subterm can be reached—where $\mathsf{se}(k)$ stands for secret encryption with k, *etc*. The path relation is defined inductively by the following two clauses.

$$m \overset{\langle\rangle}{\rightsquigarrow} m \qquad \text{and} \qquad m_1 \overset{\ell}{\rightsquigarrow} m_2 \Longrightarrow \begin{cases} m_1 \overset{\mathsf{se}(k)\cdot\ell}{\rightsquigarrow} k\{\, m_2 \,\} \\ m_1 \overset{\mathsf{pe}(k)\cdot\ell}{\rightsquigarrow} \{\, m_2 \,\}_k \\ m_1 \overset{\mathsf{ha}\cdot\ell}{\rightsquigarrow} \mathsf{hash}(m_2) \\ m_1 \overset{\pi_1\cdot\ell}{\rightsquigarrow} \langle m_2, m_3 \rangle \\ m_1 \overset{\pi_2\cdot\ell}{\rightsquigarrow} \langle m_3, m_2 \rangle \end{cases}$$

where $\langle\rangle$ is the empty list, and prefixing an element a to a list ℓ is described by the dot notation $a \cdot \ell$.

Terms with such paths between them form a category[2], with the empty list as identity map, and list-append ; as composition: if $m_1 \overset{\ell_1}{\rightsquigarrow} m_2$ and $m_2 \overset{\ell_2}{\rightsquigarrow} m_3$ then $m_1 \overset{\ell_2;\ell_1}{\rightsquigarrow} m_3$. Composition works backwards because of the (arbitrary) direction that we have used for paths.

The relation between paths and subterms is easy:

$$m_1 \preceq m_2 \iff \exists \ell.\, m_1 \overset{\ell}{\rightsquigarrow} m_2$$

Further, with paths we can define the following useful notions.

[2]It happens more frequently that a partial order can be described in a more refined way leading to morphisms in a category, for instance in the propositions-as-types view where implications become proofterms.

- *Single-occurrence*: $m_1 \preceq_1 m_2$ is described as: $m_1 \overset{\ell_1}{\rightsquigarrow} m_2$ and $m_1 \overset{\ell_2}{\rightsquigarrow} m_2$ implies $\ell_1 = \ell_2$. To be precise, what we define is really *at most* single occurrence.
- *Occurrence under public encryption with key k*: $m_1 \preceq_{\mathsf{pe}(k)} m_2$ means that each path ℓ with $m_1 \overset{\ell}{\rightsquigarrow} m_2$ must contain $\mathsf{pe}(k)$.

  Similarly one defines occurrence $m_1 \preceq_{\mathsf{se}(k)} m_2$ under secret encryption.

  These relations are extended in the obvious way to subsets. We have $m \preceq_{\mathsf{pe}(k)} U$ and $m \preceq_{\mathsf{se}(k)} U$, where the relation holds for any $m' \in U$.

  Additionally, for a set $\mathcal{K}$ of keys, $m_1 \preceq_{\mathcal{K}} m_2$ holds if each path ℓ with $m_1 \overset{\ell}{\rightsquigarrow} m_2$ contains $\mathsf{pe}(k)$ or $\mathsf{se}(k)$ with $k \in \mathcal{K}$.

3.2 Possessions

We shall refer to "possessions" as the cryptographically accessible parts of messages. For instance, m is in the possessions of a set $U \subseteq \mathsf{MSG}$ if $k\{\, m \,\} \in U$ and $\mathsf{seckey}(k) \in U$, and also if $\{\, m \,\}_k \in U$ and $\mathsf{privkey}(\overline{k}) \in U$—where the function $\overline{\cdot}$ maps a public key to the corresponding private one.

Formally we define possessions as a closure operation $\mathbf{P}\colon \mathcal{P}(\mathsf{MSG}) \to \mathcal{P}(\mathsf{MSG})$ via a least fixed point (see *e.g.* [6]). It is the analogue of Paulson's `analz` [23]. For $U \subseteq \mathsf{MSG}$ we can describe $\mathbf{P}(U) \subseteq \mathsf{MSG}$ as the smallest subset with:

$$U \subseteq \mathbf{P}(U) \qquad \text{and} \qquad \left.\begin{array}{r} k\{\, m \,\}, \mathsf{seckey}(k) \in \mathbf{P}(U) \vee \\ \{\, m \,\}_k, \mathsf{privkey}(\overline{k}) \in \mathbf{P}(U) \vee \\ \langle m, m' \rangle \in \mathbf{P}(U) \vee \\ \langle m', m \rangle \in \mathbf{P}(U) \end{array}\right\} \Longrightarrow m \in \mathbf{P}(U)$$

A basic observation is that $m \preceq \mathbf{P}(U)$ if and only if $m \preceq U$.

3.1. Example. The set of terms

$$\mathbf{P}\Big(\Big\{\, \langle\, \{\, \mathsf{seckey}(ks) \,\}_{kp}, ks\{\, \mathsf{name}(a), \mathsf{hash}(\mathsf{nonce}(n)) \,\}\, \rangle,\ \mathsf{privkey}(\overline{kp}) \Big\}\Big)$$

contains $\mathsf{name}(a)$ but not $\mathsf{nonce}(n)$.

The PVS formalisation [15] contains appropriate rewrite rules that can prove such inhabitation statement via automatic rewriting. The rewrite rules make several passes through a list of terms, each time recording the keys that are available and that are still needed. The rewriting stops at the end of such a pass when there is no overlap between these sets of available and needed keys.

Possessions can be described in terms of paths:

$$\begin{array}{rcl} m \in \mathbf{P}(U) & \iff & \exists m' \in U.\, \exists \ell.\, m \overset{\ell}{\rightsquigarrow} m' \wedge \mathsf{ha} \notin \ell \wedge \\ & & \quad \forall k.\, \mathsf{se}(k) \in \ell \Rightarrow \mathsf{seckey}(k) \in \mathbf{P}(U) \wedge \\ & & \quad \forall k.\, \mathsf{pe}(k) \in \ell \Rightarrow \mathsf{privkey}(\overline{k}) \in \mathbf{P}(U) \end{array}$$

This allows us to prove the following two important properties about occurrences under public encryptions.

1. If $m \preceq_{\mathsf{pe}(k)} U$ and $m \in \mathbf{P}(U)$ but not $m \in U$, then $\mathsf{privkey}(\overline{k}) \in \mathbf{P}(U)$.
2. If $m \preceq_{\mathsf{pe}(k)} U$ and $\mathsf{privkey}(\overline{k}) \notin \mathbf{P}(U)$, then $m \preceq_{\mathsf{pe}(k)} \mathbf{P}(U)$.

They form a crucial ingredient of the proof of the public encryption rule in Subsection 5.2.

3.3 Communications

Agents use the possessions operator from the previous subsection to decompose incoming messages. They use a "communications" operator to build up new messages that they can send out. This operator—comparable to the `synth` of [23]—is again a closure operator $\mathbf{C}\colon \mathcal{P}(\mathsf{MSG}) \to \mathcal{P}(\mathsf{MSG})$. For $U \subseteq \mathsf{MSG}$ we have $\mathbf{C}(U) \subseteq \mathsf{MSG}$ as the smallest set with:

$$\begin{array}{c} U \subseteq \mathbf{C}(U) \\ \mathsf{name}(a), \mathsf{number}(i), \mathsf{newnonce}(n), \mathsf{newseckey}(k) \in \mathbf{C}(U) \\ m, \mathsf{seckey}(k) \in \mathbf{C}(U) \implies k\{\, m \,\} \in \mathbf{C}(U) \\ m, \mathsf{pubkey}(k) \in \mathbf{C}(U) \implies \{\, m \,\}_k \in \mathbf{C}(U) \\ m \in \mathbf{C}(U) \implies \mathsf{hash}(m) \in \mathbf{C}(U) \\ m, m' \in \mathbf{C}(U) \implies \langle m, m' \rangle \in \mathbf{C}(U). \end{array}$$

The most important point is that 'new' nonces and secret keys always belong to the communications. In contrast, 'plain' nonces and secret keys only belong to $\mathbf{C}(U)$ if they already belong to U. The restriction that 'new' nonces and secret keys must be really new is imposed on a global level, that is, on bundles. This will be explained later in Section 4.2.

Obvious properties are $m \preceq U \Rightarrow m \preceq \mathbf{C}(U)$, and similarly for $\preceq_{\mathsf{pe}(k)}$ and $\preceq_{\mathsf{se}(k)}$ instead of $\preceq$. A less trivial one is:

$$\begin{array}{l} \mathsf{nonce}(n) \overset{\ell}{\rightsquigarrow} m \in \mathbf{C}(U) \implies \\ \quad \exists m' \in U.\, \exists \ell_1, \ell_2.\, \mathsf{nonce}(n) \overset{\ell_2}{\rightsquigarrow} m' \wedge m' \overset{\ell_1}{\rightsquigarrow} m \wedge \ell = \ell_1; \ell_2. \end{array}$$

It says: if $\mathsf{nonce}(n)$ appears in a message $m \in \mathbf{C}(U)$ then we can find a message $m' \in U$ which contains $\mathsf{nonce}(n)$ and is contained in m. The same result holds for $\mathsf{seckey}(k)$ in place of $\mathsf{nonce}(n)$.

4 Strands and bundles

4.1 Strands

In our formalisation we shall use "strands" as infinite sequences of possible messages. A strand describes what happens to an individual agent, in terms of the incoming and outgoing messages (like in [27]). Such a message m at an arbitrary stage i, if any, is either incoming, written as $-m$, or outgoing, written as $+m$. The type of "message" strands is thus defined as function space:

$$\mathsf{MStr} \stackrel{\text{def}}{=} \mathbb{N} \longrightarrow \Big(1 + (\mathsf{Sgn} \times \mathsf{MSG})\Big)$$

where $1 = \{\bot\}$ and $\mathsf{Sgn} = \{-, +\}$. An example of a strand is thus $\langle +m, \bot, \bot, -m', \ldots \rangle$ where $\bot$ indicates that nothing is sent or received at that stage.

With such a strand of messages we associate a strand of possessions. It contains at each stage the cryptographically accessible parts of the strand's messages so far. The type of "possession" strands is:

$$\mathsf{PStr} \stackrel{\text{def}}{=} \mathbb{N} \longrightarrow \mathcal{P}(\mathsf{MSG})$$

Such possession strands are obtained via a function $\mathsf{M2P}\colon \mathsf{MStr} \to \mathsf{PStr}$ defined by induc-

tion:

$$\begin{array}{rcl} \mathsf{M2P}(s)(0) & = & U, \quad \text{a given set of initial possessions (terms)} \\ \mathsf{M2P}(s)(i+1) & = & \begin{cases} \mathsf{M2P}(s)(i) & \text{if } s(i) = \bot \\ & \text{if } s(i) = -m \\ \mathsf{M2P}(s)(i) \cup & \\ \quad \{\mathsf{nonce}(n) \mid \mathsf{newnonce}(n) \preceq m\} \cup & \text{if } s(i) = +m \\ \quad \{\mathsf{seckey}(n) \mid \mathsf{newseckey}(k) \preceq m\} & \end{cases} \end{array}$$

This requires some explanation: the possessions at stage $i+1$ are the same as at i if nothing happens in the message strand s, *i.e.* if $s(i) = \bot$. But if there is an incoming message $-m$ at i, we extract everything we can from what we already have and from m together. For instance, if the incoming message is a secret encryption $m = k\{\, m' \,\}$ and we already possess $\mathsf{seckey}(k)$, then we possess m' at stage $i+1$. Finally, if there is an outgoing message at stage i, the plain versions of any new nonces or secret keys in the outgoing messages are added. The reason for this should become clearer in the next subsection.

We note the following two properties.

- The generated possession strand is increasing, or monotone:

$$i \leq j \implies \mathsf{M2P}(s)(i) \subseteq \mathsf{M2P}(s)(j).$$

  A disadvantage of this property is that session keys will always be around, and cannot be "forgotten".

- Call a subset X of messages **P**-closed if $\mathbf{P}(X) = X$. Then:

$$\mathsf{M2P}(s)(0) \text{ is } \mathbf{P}\text{-closed} \implies \mathsf{M2P}(s)(i) \text{ is } \mathbf{P}\text{-closed}.$$

We need to cover two more properties for (message) strands. A strand is called *well-formed* if it satisfies the following two conditions:

1. The initial possessions in $\mathsf{M2P}(s)(0)$ are "primitive", *i.e.* of the form $\mathsf{name}(a)$, $\mathsf{nonce}(n)$, $\mathsf{number}(i)$, $\mathsf{seckey}(k)$, $\mathsf{pubkey}(k)$ or $\mathsf{privkey}(k)$. This means in particular that $\mathsf{M2P}(s)(0)$ is **P**-closed—and hence all other $\mathsf{M2P}(s)(i)$ as well. This condition excludes tickets or certificates from initial possessions; for the examples in this paper it is not a problem.

2. Each outgoing message $s(i) = +m$ satisfies $m \in \mathbf{C}(\mathsf{M2P}(s)(i))$. This means that it can be build from the terms that are possessed at that stage.

Next, a strand may be determined by a protocol rule. Here we shall only consider elementary rules about message initiation and response. We allow for parametrisation by stages and sets of messages (possessions) in rules, so that the type of rules is:

$$\mathsf{Rule} \stackrel{\text{def}}{=} \mathbb{N} \times \mathcal{P}(\mathsf{MSG}) \longrightarrow \mathcal{P}\big(\mathsf{MSG} + (\mathsf{MSG} \times \mathsf{MSG})\big)$$

Here $+$ denotes a disjoint union. We then say that a strand $s \in \mathsf{MStr}$ is *rule-based* wrt. rule $r \in \mathsf{Rule}$ if for each $s(i) = +m$ one has either $m \in r(i, \mathsf{M2P}(s)(i))$ or $i > 0$ and $s(i-1) = -m'$ with $(m', m) \in r(i, \mathsf{M2P}(s)(i))$. In the first case the outgoing message $+m$ is an initiative according to rule r, and in the second case it is a reaction determined by r.

4.2 Bundles

Bundles are collections of strands, parametrised by agents, including a spy. Recall we used the type AG for agents, in which we now assume a special element $\mathsf{spy} \in \mathsf{AG}$. We shall define two new types: bundles and of bundle constraints, and define what it means that a bundle satisfies such a constraint. First, the type of bundles:

$$\mathsf{Bun} \stackrel{\text{def}}{=} \mathsf{AG} \longrightarrow \mathsf{MStr}$$

describes for each agent a message strand. A particular bundle can thus be represented in a table:

	spy	A_1	$A_2 \quad \cdots \quad \in \mathsf{AG}$
stage 0	$-\langle m_1, m_2 \rangle$	$+\langle m_1, m_2 \rangle$	$\bot$
1	$+m_1$	$-m_1$	$-m_1$
2	$\vdots$	$\vdots$	$\vdots$
$\vdots$			

This describe a scenario where the tuple message $\langle m_1, m_2 \rangle$ sent by agent A_1 at stage 0 ends up with the spy. At the next stage the spy replays the first part m_1 of this tuple, and both A_1 and A_2 receive it.

Bundles involve arbitrary sequences. We want them to satisfy certain restrictions, partly determined by protocol rules. This is realised via the notion of bundle constraint:

$$\mathsf{BunCst} \stackrel{\text{def}}{=} \mathsf{AG} \longrightarrow \Big(\mathcal{P}(\mathsf{MSG}) \times \mathsf{Rule}\Big).$$

Such a bundle constraint tells for each participant what the initial possessions and the rules are. It thus specifies a protocol.

The next seven points describe what it means that an arbitrary bundle $b \in \mathsf{Bun}$ satisfies a bundle constraint $bc \in \mathsf{BunCst}$. Intuitively it says that the bundle b is a combined run for all participants in the protocol specified by bc.

1. For each agent $a \in \mathsf{AG}$, $b(a)$ is a well-formed and rule-based strand with initial possessions given by $\pi_1(bc(a))$ and rule by $\pi_2(bc(a))$.

2. At each stage i, if there is activity, there must be a sender: if some $b(a)(i) \neq \bot$, then there must be an $a' \in \mathsf{AG}$ with $b(a')(i) = +m$.

3. There is at most one sender at each stage: if $b(a)(i) = +m$ and $b(a')(i) = +m'$, then $a = a'$ (and hence also $m = m'$).

   (This excludes so-called parallel attacks, see [19].)

4. Received messages are plain versions of what is sent: if $b(a)(i) = +m$ and $b(a')(i) = -m'$, then $m' = \mathsf{n2p}(m)$.

5. The spy receives all messages: if $b(a)(i) = +m$ and $a \neq \mathsf{spy}$, then $b(\mathsf{spy})(i) = -\mathsf{n2p}(m)$.

6. New nonces are really fresh: if $b(a)(i) = +m$ and $\mathsf{newnonce}(n) \preceq m$, then for each $a' \in \mathsf{AG}$, not: $\mathsf{nonce}(n) \preceq \mathsf{M2P}(b(a'))(i)$.

7. New secret keys are also fresh: if $b(a)(i) = +m$ and $\mathsf{newseckey}(k) \preceq m$, then for each $a' \in \mathsf{AG}$, not: $\mathsf{seckey}(k) \preceq \mathsf{M2P}(b(a'))(i)$ and also not: $k\{\, m' \,\} \preceq \mathsf{M2P}(b(a'))(i)$, for some term m'.

The set of all bundles satisfying a particular constraint/protocol form our model of the protocol. In the remainder of this section we shall investigate some further properties of such models.

First of all, the standard Dolev-Yao attack model [7] is incorporated, like in strand spaces. The rules for the spy may further refine these capabilities. But in the basic set up it is guaranteed only that the spy receives any outgoing message. If no-one else does, one may understand that the spy has deleted the message. But the spy may also replay the message, or adapt it—subject to cryptographic constraints as described by the possessions operators from Subsection 3.2.

Next, we are finally in a position to explain the role of the 'new' versions $\mathsf{newnonce}(n)$ and $\mathsf{newseckey}(k)$ of nonces and secret keys. They can always be included in outgoing messages $m \in \mathbf{C}(\mathsf{M2P}(b(a))(i))$, according to the definition of the communication operator $\mathbf{C}$ from Subsection 3.3. But the last two of the above constraints require that such 'new' subterms in outgoing messages should be globally fresh. This is how we realise a global freshness idealisation. Further, by invoking the $\mathsf{n2p}$ functions on the incoming messages the 'new' subterms, if any, are turned into 'plain' ones, so that 'new' really only occurs in the first use. It is indeed not hard to see that all sets $\mathsf{M2P}(b(a))(i)$ of possessions only contain plain terms.

An important consequence of the distinction between new and plain nonces (and secret keys) is that if we have a plain nonce $\mathsf{nonce}(n) \preceq m$ as subterm of an outgoing message $s(a)(i) = +m$, then $\mathsf{nonce}(n)$ is already in a's possession at i—this follows from the last statement in Subsection 3.3—so that it must be a re-use of n, *i.e.* so that n must already have been sent before.

To conclude, we list a number of technical properties that hold for an arbitrary bundle $b \in \mathsf{Bun}$ satisfying a constraint $bc \in \mathsf{BunCst}$.

- All sets of possessions $\mathsf{M2P}(b(a))(i)$ are $\mathbf{P}$-closed and only contain plain terms.
- New nonces can only be sent once: if there are two outgoing messages $b(a_1)(i_1) = +m_1$ and $b(a_2)(i_2) = +m_2$ which both have the same new nonce $\mathsf{newnonce}(n) \preceq m_1$ and $\mathsf{newnonce}(n) \preceq m_2$ as subterm, then $a_1 = a_2$ and $i_1 = i_2$, and hence also $m_1 = m_2$.

  Similarly for new secret keys.
- If we have a subterm $m \preceq m_1$ of an incoming message $b(a_1)(i_1) = -m_1$, then there is an agent a_2 and earlier stage $i_2 \leq i_1$ with outgoing message $b(a_2)(i_2) = +m_2$ containing a "source" subterm $m' \preceq m_2$ where $m' \in \mathbf{C}(\mathsf{M2P}(b(a_2))(i_2))$ satisfies $\mathsf{n2p}(m') = m$.
- More specifically, if we have a secret encryption $k\{\, m \,\} \preceq m_1$ of an incoming message $b(a_1)(i_1) = -m_1$, then there is an agent a_2 and earlier stage $i_2 \leq i_1$ where the secret key $\mathsf{seckey}(k) \in \mathsf{M2P}(b(a_2))(i_2)$ is possessed and where there is an outgoing message $b(a_2)(i_2) = +m_2$ with a secret encryption $k\{\, m' \,\} \preceq m_2$ as subterm, for which $m' \in \mathbf{C}(\mathsf{M2P}(b(a_2))(i_2))$ satisfies $\mathsf{n2p}(m') = m$.

  There is an analogous result for public encryptions.

Via such results we can reason backwards about what happens in bundles, much like in the well-foundedness arguments used in [27].

4.3 Comparison with strand spaces

The notions of strand and bundle as defined above resemble the notions with the same names in [27]. The basic idea is: behaviour of each agent is described as a *strand*; a collection of strands communicating with each other form a *bundle*. A bundle is a whole system where communication takes place. However there are also some differences between the original notions and ours. We shall explain these differences here.

In general in a system where communication takes place according to a cryptographic protocol, there are two different kinds of constraints on the participants' behaviour, namely a *message rule* and a *cryptographic ability*. In our case every agent that wants to send a message needs to be able to construct the message. This is a crucial part of what we call well-formedness. This restriction only applies explicitly to the spy in the original strand space model. There, strands belonging to the spy must be instantiations of the several kinds of templates—each corresponding to a cryptographic operation which the spy can conduct, such as decryption, encryption, providing a possessed key, *etc.* These templates are independent of specific protocols. The strands of legitimate (non-spy) agents must be instantiations of message rules. These rules must incorporate our well-formedness restrictions. The constraints are thus basically the same in both models, but organised in a slightly different manner. This is summarised in the following table.

	in original strand space semantics	in our semantics
message rule restricts	only legitimate agents	everybody
cryptographic ability restricts	only spy	everybody

Our formalism is a bit more complicated, because we have to keep track of each agents possessions at each stage. However, the formalism is more uniform and is thus more suitable for expressing various rules, and for proving their soundness. Additionally, we can use our formalism as basis for other purposes than analysis of authentication protocols. For example, in the analysis of anonymity protocols (see *e.g.* [9]) we often need to prove that a certain agent does not possess some given message. It is not possible in the original strand space formalism, in which we cannot express legitimate participant's possessions.

As illustration Lowe's attack [16] on the Needham-Schroeder protocol (see Section 2) is presented in the original strand space formalism in Figure 1, and in our formalism in Figure 2. In the latter we have written N_i for $\mathsf{nonce}(N_i)$ in order to save space. Also we leave the possession strands implicit. Observe that in Figure 1, strands for legitimate agents embody message rules while those for the spy embody cryptographic operations. These constraints are implicitly imposed on every agent in Figure 2 as the well-formedness requirement.

Asynchronous nature of the network—messages can arrive delayed or in a different order—is well modelled in the original strand space formalism which does not have a global timing (or stages). In our formalism it is modelled using the spy: for example a delayed message is first received only by the spy and later sent by the spy to the intended recipient.

5 Logical formulas and rules for protocols

In this section we put a layer of abstraction on the model that was introduced in the previous section by defining appropriate logical formulas that capture essential aspects of the model. As we emphasised in the beginning our "logic" is interpreted and consists of a number of definitions and operations for formulas, together with a suitable collection of implications (rules) between them. We refrain at this stage from formulating a proper syntactic logic

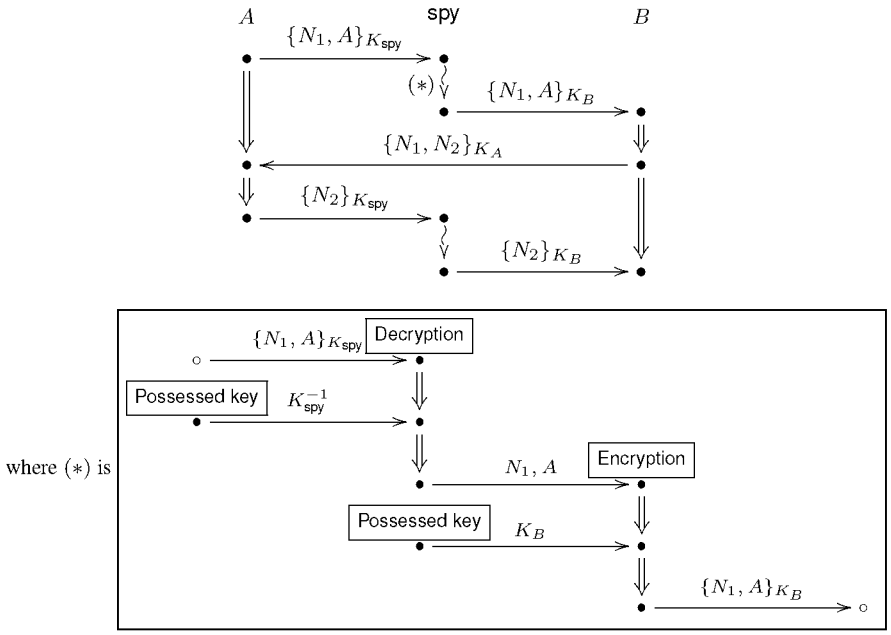

Figure 1: Lowe's attack in the formalism of [27]

Time	A's MStr	B's MStr	spy's MStr
0	$+\{$ newnonce$(N_1), A\}_{K_{spy}}$	$\perp$	$-\{N_1, A\}_{K_{spy}}$
1	$\perp$	$-\{N_1, A\}_{K_B}$	$+\{N_1, A\}_{K_B}$
2	$-\{N_1, N_2\}_{K_A}$	$+\{N_1,$ newnonce$(N_2)\}_{K_A}$	$-\{N_1, N_2\}_{K_A}$
3	$+\{N_2\}_{K_{spy}}$	$\perp$	$-\{N_2\}_{K_{spy}}$
4	$\perp$	$-\{N_2\}_{K_B}$	$+\{N_2\}_{K_B}$

Figure 2: Lowe's attack in our formalism

with a derivation relation because: (1) we think that the set of rules is not sufficiently stable yet, and that further applications of this framework may lead to refinements and/or additions; and (2) for the verification work in a theorem prover it is unnecessary—even inconvenient—to have a purely syntactic logic.

The formulas that we shall consider below are atomic formulas of the form:

$$A\ \mathsf{Possess}\ (m) \quad A\ \mathsf{Sends}\ (m) \quad A\ \mathsf{Says}\ (m) \quad A\ \mathsf{Receives}\ (m)$$
$$\mathsf{Secret}(A)(m) \quad \mathsf{Secret}(A, B)(m)$$

and compound formulas of the form:

$$\neg\varphi \quad \varphi_1 \wedge \varphi_2 \quad \forall x \in X.\, \varphi \quad \mathsf{HenceForth}\ (\varphi) \quad A\ \mathsf{Knows}\ (\varphi)$$

where φ, φ_i are formulas.

These formulas, in interpreted form, are special predicates on our model. They take a bundle constraint $bc \in \mathsf{BunCst}$, a bundle $b \in \mathsf{Bun}$ satisfying bc, and a number (or stage) $i \in \mathbb{N}$ to true or false. In order to describe the type of formulas we introduce the notation $[bc]$ for the set of bundles that satisfy the bundle constraint bc. Then:

$$\mathsf{Form} \stackrel{\text{def}}{=} \prod_{bc \in \mathsf{BunCst}} \Big([bc] \times \mathbb{N} \to \mathsf{bool} \Big)$$

Here $\prod$ denotes a dependent product.
We shall write Greek letters $\varphi, \psi, \ldots$ for formulas. The usual logical operations are extended to such formulas via pointwise definitions, as in:

$$\begin{array}{rcl} \big(\varphi_1 \wedge \varphi_2\big)(bc, b, i) & \stackrel{\text{def}}{=} & \varphi_1(bc, b, i) \wedge \varphi_2(bc, b, i) \\ \big(\forall x \in X.\, \varphi(x)\big)(bc, b, i) & \stackrel{\text{def}}{=} & \forall x \in X.\, \varphi(x)(bc, b, i). \end{array}$$

The (linear) temporal operator $\mathsf{HenceForth}\ (-)$—sometimes written as $\Box$—is defined in the obvious way, namely as:

$$\big(\mathsf{HenceForth}\ (\varphi)\big)(bc, b, i) \stackrel{\text{def}}{=} \forall j \geq i.\, \varphi(bc, b, j).$$

The epistemic "knowledge" operator will be described separately in Subsection 5.4. Very often the bundle (constraint) arguments bc and b remain the same and there is only variation in the stage argument i. Therefore it makes sense to leave bc and b implicit, and write $\varphi @ i$ for $\varphi(bc, b, i)$. The temporal operator then says that $\mathsf{HenceForth}\ (\varphi)\ @\, i$ is $\varphi @ j$ for all $j \geq i$. This $@$-notation was already used informally in Section 2.

5.1 Basic formulas and rules

Next we introduce some of the basic formulas about possession, sending, seeing *etc.* Throughout we shall use variables $A \in \mathsf{AG}$ and $m \in \mathsf{MSG}$ for agents and messages.

$$A\ \mathsf{Possess}\ (m)\ (bc, b, i) \stackrel{\text{def}}{=} m \in \mathsf{M2P}(b(A))(i)$$

This says that agent A possesses message m if m can be extracted from A's messages up-to i. Notice that we are careful to talk about "possession" and not "knowledge" of messages, because "knowledge" is reserved for use with the modal operator from Subsection 5.4.

As we noted in Section 4, M2P is monotone, so the implication A Possess $(m) \Rightarrow$ HenceForth $(A$ Possess $(m))$ holds. Hence the following rule is sound.

$$\frac{A \text{ Possess } (m)}{\text{HenceForth } (A \text{ Possess } (m))}$$

Since M2P yields **P**-closed subsets of terms we also have the following four closure rules.

$$\frac{A \text{ Possess } (k\{\, m \,\}) \qquad A \text{ Possess } (\text{seckey}(k))}{A \text{ Possess } (m)}$$

$$\frac{A \text{ Possess } (\{\, m \,\}_k) \qquad A \text{ Possess } (\text{privkey}(\overline{k}))}{A \text{ Possess } (m)}$$

$$\frac{A \text{ Possess } (\langle m_1, m_2 \rangle)}{A \text{ Possess } (m_1)} \qquad \frac{A \text{ Possess } (\langle m_1, m_2 \rangle)}{A \text{ Possess } (m_2)}$$

Our next formulas are about sending.

$$\begin{aligned} A \text{ Sends } (m)\ (bc, b, i) &\stackrel{\text{def}}{=} b(A)(i) = +m \\ A \text{ Says } (m)\ (bc, b, i) &\stackrel{\text{def}}{=} \exists m' \in \text{MSG}.\, m \preceq m' \wedge A \text{ Sends } (m')(bc, b, i) \\ &\qquad \wedge\, m \in \mathbf{C}(\text{M2P}(b(A))(i)) \end{aligned}$$

Agent A thus "says" a message m if m is a subterm of an outgoing message and m itself can be constructed. Here are some obvious rules.

$$\frac{A \text{ Sends } (m)}{A \text{ Says } (m)} \qquad \frac{A \text{ Says } (\text{newnonce}(n)) \,@\, i}{A \text{ Possess } (\text{nonce}(n)) \,@\, i+1}$$

$$\frac{A \text{ Says } (\text{newnonce}(n)) \,@\, i \quad B \text{ Says } (\text{newnonce}(n)) \,@\, j}{A = B \ \wedge\ i = j}$$

There are analogous rules for newseckey(k) in place of newnonce(n).

Now we turn to receiving messages.

$$\begin{aligned} A \text{ Receives } (m)\ (bc, b, i) &\stackrel{\text{def}}{=} b(A)(i) = -m \\ A \text{ Sees } (m)\ (bc, b, i) &\stackrel{\text{def}}{=} i > 0 \wedge A \text{ Possess } (m)(bc, b, i) \wedge \\ &\qquad \neg(A \text{ Possess } (m)(bc, b, i-1)) \wedge \\ &\qquad \exists j, m'.\, j < i \wedge A \text{ Receives } (m')(bc, b, j) \wedge m \preceq m' \end{aligned}$$

The notion is "seeing" is a bit complicated: A sees a message m at stage i if m is a subterm of an earlier incoming message and is only now (*i.e.* at stage i and not earlier) accessible. Then:

$$\frac{A \text{ Sees } (m)}{A \text{ Possess } (m)} \qquad \frac{A \text{ Sees } (k\{\, m \,\}) \qquad A \text{ Possess } (\text{seckey}(k))}{A \text{ Possess } (m)}$$

$$\frac{A \text{ Sees } (k\{\, m \,\}) \,@\, i}{\exists B, m', j \leq i.\, \text{n2p}(m') = m \wedge B \text{ Says } (m') \,@\, j \wedge B \text{ Possess } (\text{seckey}(k)) \,@\, j}$$

There are similar rules for public encryptions.

Finally we have secrecy formulas, in twofold, namely in shared form (for two agents) and un-shared form (for a single agent). We use overloading and give them the same name.

$$\begin{array}{l}\mathsf{Secret}(A)(m)\ (bc, b, i)\\ \quad \stackrel{\text{def}}{=}\ A\ \mathsf{Possess}\ (m)(bc, b, i) \wedge\\ \qquad\qquad \forall X, j \leq i.\ X\ \mathsf{Possess}\ (m)(bc, b, j) \Rightarrow X = A\\ \mathsf{Secret}(A, B)(m)\ (bc, b, i)\\ \quad \stackrel{\text{def}}{=}\ A\ \mathsf{Possess}\ (m)(bc, b, i) \wedge B\ \mathsf{Possess}\ (m)(bc, b, i) \wedge\\ \qquad\qquad \forall X, j \leq i.\ X\ \mathsf{Possess}\ (m)(bc, b, j) \Rightarrow X = A \vee X = B\end{array}$$

The definition can easily be extended to multiple (more than two) agents. Associated rules appear in Subsection 5.3.

5.2 Rules for authentication

We present some logical rules by which we can draw authentication conclusions: from what an agent A observes, A can be sure that a certain action has been actually done by a specific agent B. Even in the presence of the Dolev-Yao adversary that controls all the traffic in the network, we can establish authentication with the help of cryptography.

The rules in this section—together with the strengthened ones later in Section 7—are to our semantics what the *authentication tests* [12] are to the strand space formalism [27]. Authentication properties often involve the most complex proofs among various security goals. Since the strand space formalism is independent of specific protocols, it is possible to establish generic "lemmas" about strand spaces which, via suitable instantiations, help us to prove authentication properties of various protocols. The proofs of these generic lemmas are quite complicated, as one observes in [12]. However, the task of proving these lemmas is like giving complicated authentication proofs once for (almost) all protocols. That is the virtue of having "authentication tests" for the strand space formalism. The logical rules we shall introduce are useful in the very same sense.

The typical case we wish to consider is when an agent A first sends out a new nonce under encryption with another agent B's public key, and then sees its own nonce back in unencrypted form. This forms a so-called outgoing authentication test in [12]. The desired conclusion is then that B must have seen the nonce, together with whatever was also included in the encryption, because only B could have decrypted the relevant message. Making all this precise turns out to be quite subtle.

The rule that we have is too large to fit on one line, so that we describe it with labels as follows.

$$\frac{\text{"}A \text{ sends at } j\text{"} \quad \text{"}A \text{ sees at } i \geq j\text{"} \quad \text{"}A \text{ doesn't say after } j\text{"} \quad \text{"}B\text{'s privkey secret until } i\text{"}}{\exists j_1.\, j < j_1 \leq i \wedge \text{"}B \text{ sees at } j_1\text{"} \wedge \exists j_2, C.\, j_1 \leq j_2 \leq i \wedge \text{"}C \neq A \text{ sends}\text{"}}$$

The meaning of the assumptions in this rule will be explained first.

- "A sends at j" means that A sends a public-encrypted new nonce as a subterm. Formally: $A\ \mathsf{Sends}\ (m)\ @\ j$ with $\{\, \mathsf{newnonce}(n), m_1 \,\}_k \preceq m$, together with the requirement that the nonce occurs only once in the outgoing message: $\mathsf{nonce}(n) \preceq_1 \mathsf{n2p}(m)$.

- "A sees at $i \geq j$" is used as abbreviation for: $A\ \mathsf{Sees}\ (m_2)\ @\ i$ where $i \geq j$ and not: $\mathsf{nonce}(n) \preceq_{\mathsf{pe}(k)} m_2$. The latter says that the nonce n has in m_2 an occurrence which is outside an encryption by k; it requires a subtle addition to exclude trivial cases, namely that m_2 is not $\mathsf{nonce}(n)$ itself or the tuple $\langle \mathsf{nonce}(n), \mathsf{n2p}(m_1) \rangle$.

- "A doesn't say after j" means not: A $\mathsf{Says}\,(\mathsf{nonce}(n))\,@\,j'$, for any j' with $j < j' \leq i$. Indeed, the nonce n at stake should not be used again by A, so that when it re-appears it must indeed have been decrypted.

- "B's privkey secret until i" expresses the crucial assumption that B is the only one that possesses the private key associated with k, *i.e.* $\mathsf{Secret}(B)(\mathsf{privkey}(\overline{k}))\,@\,i'$ for all $i' \leq i$.

The rule's conclusions then have the following meaning.

- "B sees at j_1" means B $\mathsf{Sees}\,(\langle \mathsf{nonce}(n), \mathsf{n2p}(m_1)\rangle)\,@\,j_1$ and expresses that B must have seen the encrypted tuple—because it is the only agent that could have decrypted the tuple.

- "$C \neq A$ sends" finally means $C \neq A \wedge C$ $\mathsf{Sends}\,(m_3)\,@\,j_2$ for some message m_3 with $\mathsf{n2p}(m_3) = m_2$. One might have expected that B must also have been the one that sent the incoming message m_2 of A, but that is not guaranteed: B could have passed the nonce n on to some other agent C who uses it for the message seen by A.

Proving the soundness of this rule is a *tour de force*, and involves many case distinctions. The main steps are as follows.

1. The incoming term m_2 must come from some agent, say C. We know that C is not A, because A does not "say" n.

2. The nonce n must occur unencrypted in C's possessions, because of the last property mentioned in Subsection 3.3.

3. Now we use well-foundedness to find the first stage, say j_1, where an agent, say A', different from A possesses n in unencrypted form.

4. By a non-trivial induction proof we establish that up-to j_1 the nonce n can only occur encrypted under k.

5. Because B is the only agent with the decryption key, we must have $A' = B$, using property 1. mentioned at the end of Subsection 3.2.

There is a similar rule in which $\mathsf{nonce}(n)$ and $\mathsf{newnonce}(n)$ are replaced by $\mathsf{seckey}(k)$ and $\mathsf{newseckey}(k)$: a freshly generated secret key can play the role of a nonce as a random value. But there also is a more useful variation, in which the secret key k does not re-appear as subterm of the incoming message but as (secret) encryption key in a cipher text $k\{\, m \,\}$. We shall see such an example in Section 6.

5.3 Rules for secrecy

Among the many possible rules for secrecy we shall consider the case where two agents A_1, A_2 only exchange a nonce under each other's public keys k_1, k_2 respectively. In labeled form this rule looks as follows.

$$\frac{\text{"}A_1 \text{ sends new } n \text{ at } j \leq i\text{"} \quad \text{"}A_1, A_2 \text{ send } n \text{ only encrypted"} \quad \text{"}A_1, A_2 \text{ key secrecy"}}{B\ \mathsf{Possess}\,(\mathsf{nonce}(n))\,@\,i \Longrightarrow B = A_1 \vee B = A_2} \tag{1}$$

The conclusion speaks for itself, so we only explain the three assumptions.

- "A_1 sends new n at $j \leq i$" means A_1 $\mathsf{Says}\,(\mathsf{newnonce}(n))\,@\,j$ for $j \leq i$.

- "A_1, A_2 send n only encrypted" expresses that for j' with $j \leq j' \leq i$ the nonce n is sent by A_1 at j' only under A_2's public key k_2, and vice-versa. Formally, if A_1 $\mathsf{Sends}\,(m)\,@\,j'$ then $\mathsf{nonce}(n) \preceq_{\mathsf{pe}(k_2)} m$, and similarly for A_2.
- "A_1, A_2 key secrecy" expresses that agents A_1 and A_2 both keep their own private keys secret: $\mathsf{Secret}(A_p)(\mathsf{privkey}(\overline{k_p}))\,@\,i'$ for $p \in \{1, 2\}$ and $i' \leq i$.

There are similar secrecy rules possible where a nonce or secret key is only sent under secret encryptions, or even under both public and secret encryptions.

5.4 Knowledge of formulas

In order to define knowledge we keep our bundle constraint $bc \in \mathsf{BunCst}$ fixed, but consider different bundles that satisfy the constraint. Recall that we write $[bc]$ for the set of such bundles. We define a collection of equivalence relations $\overset{A,i}{\sim} \subseteq [bc] \times [bc]$, for $A \in \mathsf{AG}$ and $i \in \mathbb{N}$ as follows.

$$b \overset{A,i}{\sim} b' \quad \overset{\text{def}}{=} \quad \forall j \leq i.\, b(A)(j) = b'(A)(j).$$

This means that up-to stage i the strands of A are the same in b and b'. We then define, much like in [25],

$$A\ \mathsf{Knows}\,(\varphi)\ (bc, b, i) \quad \overset{\text{def}}{=} \quad \forall b' \in [bc].\, b \overset{A,i}{\sim} b' \Rightarrow \varphi(bc, b', i).$$

The intuition behind this definition is the following. At stage i in bundle b an agent A only has information about its own incoming and outgoing message so far, *i.e.* about $b(A)(j)$ for $j \leq i$. If φ holds in all possible scenarios b' that agree with b on these incoming and outgoing messages of A, then φ can in fact only depend on these messages (because everything else may differ in the various scenarios), and so A has all the information to know φ.

Typically in verifications, if we can prove an implication $\varphi_1 \wedge \cdots \wedge \varphi_n \Rightarrow \psi$ and the assumptions φ_i are of the the form $A\ \mathsf{Sends}\,(-)$, $A\ \mathsf{Sees}\,(-)$ or $A\ \mathsf{Knows}\,(-)$, only involving agent A, then one can prove $\varphi_1 \wedge \cdots \wedge \varphi_n \Rightarrow A\ \mathsf{Knows}\,(\psi)$, since the result ψ only depends on A's perspective. The next section contains several such examples.

Because the relations $\sim$ are equivalence relations we have the familiar associated "S5" modal rules (see *e.g.* [10, 2]), including for instance:

$$\frac{A\ \mathsf{Knows}\,(\varphi)}{\varphi} \qquad \frac{A\ \mathsf{Knows}\,(\varphi)}{A\ \mathsf{Knows}\,(A\ \mathsf{Knows}\,(\varphi))} \qquad \frac{A\ \mathsf{Knows}\,(\varphi) \quad A\ \mathsf{Knows}\,(\psi)}{A\ \mathsf{Knows}\,(\varphi \wedge \psi)}$$

where the double line means that the rule may be used in both directions.

6 The bilateral key exchange example

In this final section we apply the theories from the previous section to a standard protocol, the so-called bilateral key exchange (BKE), described in [5, §§6.6.6]. Our aim is to establish an authentication result in the form of a shared secret. The protocol involves the following steps.

$$\begin{array}{lcl} A \longrightarrow B & : & A, \{\, n_A, A \,\}_{K_B} \\ B \longrightarrow A & : & \{\, K, \mathsf{hash}(n_A), n_B, B \,\}_{K_A} \\ A \longrightarrow B & : & K\{\, \mathsf{hash}(n_B) \,\} \end{array} \tag{2}$$

The protocol is an interesting verification challenge because of 1) the combination of public and secret encryption (with a fresh session key K), and 2) the hashed versions of nonces are used as proof of possession of the original nonces.

We shall first consider the representation of this protocol in our model, and then discuss some interesting statements about the protocol (and a sketch of their proofs).

6.1 Representation of the BKE protocol

As agents we take the set/type $\mathsf{AG} = \{A, B, S\}$ for Alice, Bob and the Spy, with corresponding public keys K_A, K_B, K_S, that are (pairwise) different. We assume three nonce functions N_A, N_B, N_S, so that at each stage i we have a fresh nonce $\mathsf{newnonce}(N_A(i))$ for Alice (and similarly for Bob and the Spy). We assume that:

$$i \neq j \Rightarrow N_A(i) \neq N_A(j) \qquad N_A(i) \neq N_B(j) \qquad \textit{etc.}$$

Further we need a function $\mathsf{SK}(-)$ so that Bob can at each stage take a different session key $\mathsf{SK}(i)$.

The initial possessions of our three agents are given as the following sets of messages.

$$\begin{array}{lcl} P_A & = & \{\, \mathsf{pubkey}(K_A), \mathsf{privkey}(\overline{K_A}), \mathsf{pubkey}(K_B), \mathsf{pubkey}(K_S) \,\} \\ P_B & = & \{\, \mathsf{pubkey}(K_A), \mathsf{pubkey}(K_B), \mathsf{privkey}(\overline{K_B}), \mathsf{pubkey}(K_S) \,\} \\ P_S & = & \{\, \mathsf{pubkey}(K_A), \mathsf{pubkey}(K_B), \mathsf{pubkey}(K_S), \mathsf{privkey}(\overline{K_S}) \,\}. \end{array}$$

Hence at stage 0 they possess their own public and private key, and each others public keys.

Recall that a rule is a function $\mathbb{N} \times \mathcal{P}(\mathsf{MSG}) \longrightarrow \mathcal{P}\big(\mathsf{MSG} + (\mathsf{MSG} \times \mathsf{MSG})\big)$ that restricts the sending of messages. For the spy we impose no restrictions via (message) rules, so that its rule is:

$$r_S(i, U) \quad = \quad \{z \in \mathsf{MSG} + (\mathsf{MSG} \times \mathsf{MSG}) \mid \mathsf{true}\}.$$

Hence rule-basedness does not restrict the spy's strand. Note that, however, well-formedness does restrict: the spy can send any message, as long as the spy can (cryptographically) construct the message.

The rules for Alice and Bob correspond to the protocol rules (2) from the beginning of this section. Alice has both an initiator rule (for spontaneously sending $A, \{\, n_A, A \,\}_{K_B}$) and a responder rule (for the final message $K\{\, \mathsf{hash}(n_B) \,\}$). We formalise this as follows.

$$\begin{array}{l} r_A(i, U) \\ \quad = \quad \{\langle \mathsf{name}(A), \{\, \mathsf{newnonce}(N_A(i)), \mathsf{name}(A) \,\}_{K_X} \rangle \mid X \in \mathsf{AG}, X \neq A\} \\ \qquad \cup \\ \qquad \{\, (\, \{\, \mathsf{seckey}(K), \mathsf{hash}(\mathsf{nonce}(N_A(j))), \mathsf{nonce}(n), \mathsf{name}(X) \,\}_{K_A}, \\ \qquad\quad K\{\, \mathsf{hash}(\mathsf{nonce}(n)) \,\}\,) \\ \qquad\qquad\qquad \mid j < i \wedge X \neq A \wedge \mathsf{nonce}(n) \notin U \wedge \mathsf{seckey}(K) \notin U \,\}. \end{array}$$

This rule thus says that A may at any stage i send the BKE protocol's first message $\langle \mathsf{name}(A), \{\, \mathsf{newnonce}(N_A(i)), \mathsf{name}(A) \,\}_{K_X} \rangle$, in which the newly generated nonce $N_A(i)$ is encrypted with another agent X's public key. Notice that the protocol description (2) prescribes that the first message should be sent to B, but this is misleading since A does not know for sure that B can be trusted, *i.e.* is not the spy. Hence we should leave this open by using a variable X. This enables the spy to participate as player in the protocol. In A's strand multiple runs of the protocol can be initiated; in each of the runs both B and the spy can be a responder.

The second part of the rule (after the union $\cup$) describes the reaction: if A gets a message of the form $\{\, \mathsf{seckey}(K), \mathsf{hash}(\mathsf{nonce}(N_A(j))), \mathsf{nonce}(n), \mathsf{name}(X) \,\}_{K_A}$ where $j < i$, $X \neq A$ and both $\mathsf{nonce}(n)$ and $\mathsf{seckey}(K)$ are not already in A's possession, then A replies by sending out the message $K\{\, \mathsf{hash}(\mathsf{nonce}(n)) \,\}$ that is built from the incoming nonce and key. The condition that neither $\mathsf{nonce}(n)$ nor $\mathsf{seckey}(K)$ is possessed yet is included to prevent replays, and is actually used in the verification. It is needed because our formalisation does not have an explicit notion of run. A consequence is that a rule can be applied repeatedly resulting in different continuations. In the BKE protocol for instance, having sent the first message with nonce n_A Alice may reply to different pairs K, n_B for this same nonce.

We turn to Bob's rule. It only involves a reaction:

$$
\begin{array}{l}
r_B(i,U) \\
\quad = \quad \{\, (\langle \mathsf{name}(X), \{\, \mathsf{nonce}(n), \mathsf{name}(X) \,\}_{K_B} \rangle, \\
\qquad\quad \{\, \mathsf{newseckey}(\mathsf{SK}(i)), \mathsf{hash}(\mathsf{nonce}(n)), \mathsf{newnonce}(N_B(i)), \mathsf{name}(B) \,\}_{K_X}\,) \\
\qquad\qquad \mid X \neq B \wedge \mathsf{nonce}(n) \notin U \,\}.
\end{array}
$$

Hence if B receives a message of the form $\langle \mathsf{name}(X), \{\, \mathsf{nonce}(n), \mathsf{name}(X) \,\}_{K_B} \rangle$ where $\mathsf{nonce}(n)$ is not already possessed and $X \neq B$, then B responds by picking both a fresh nonce $N_B(i)$ and a session key $\mathsf{SK}(i)$, and including them in the encrypted response message $\{\, \mathsf{newseckey}(\mathsf{SK}(i)), \mathsf{hash}(\mathsf{nonce}(n)), \mathsf{newnonce}(N_B(i)), \mathsf{name}(B) \,\}_{K_X}$, using the public key of the agent X that occurs in the incoming message.

We see that the representation of the BKE protocol basically follows the informal description (2), but requires that certain implicit assumptions (about the targets of messages or the freshness of incoming nonces and keys) are made explicit.

Formally, the above initial possessions and rules form a bundle constraint $\mathsf{BKE} \in \mathsf{BunCst}$, like in Subsection 4.2. The properties that we shall establish below hold for an arbitrary bundle $b \in \mathsf{Bun}$ satisfying the constraint BKE.

6.2 BKE properties

Once the protocol is represented appropriately, the verification can start. Below we shall sketch some of the secrecy properties that have been proven.

First of all we establish that A and B keep their private keys secret: for each i,

$$\mathsf{Secret}(A)(\mathsf{privkey}(\overline{K_A})) \mathbin{@} i \qquad \text{and} \qquad \mathsf{Secret}(B)(\mathsf{privkey}(\overline{K_B})) \mathbin{@} i$$

Here we use a secrecy rule that says: if a private key is never sent out then it remains secret. The rule is found in the PVS specification. Next we have

$$
\begin{array}{l}
A\ \mathsf{Sends}\ (\langle \mathsf{name}(A), \{\, \mathsf{newnonce}(N_A(i)), \mathsf{name}(A) \,\}_{K_B} \rangle) \mathbin{@} i \\
\quad \wedge \\
X\ \mathsf{Possess}\ (\mathsf{nonce}(N_A(i))) \mathbin{@} j \vee X\ \mathsf{Possess}\ (\mathsf{hash}(\mathsf{nonce}(N_A(i)))) \mathbin{@} j \\
\quad \Longrightarrow \\
X = A \vee X = B
\end{array}
$$

This follows from the secrecy rules in Subsection 5.3. There is a similar implication for B

with his session key:

$$\begin{array}{l} B\ \mathsf{Sends}\ (\{\,\mathsf{newseckey}(\mathsf{SK}(i)), \mathsf{hash}(\mathsf{nonce}(n)), \mathsf{newnonce}(N_B(i)), \\ \qquad\qquad \mathsf{name}(B)\,\}_{K_A})\ @\, i \\ \quad \wedge \\ X\ \mathsf{Possess}\ (\mathsf{seckey}(\mathsf{SK}(i)))\ @\, j \\ \quad \Longrightarrow \\ X = A \vee X = B \end{array}$$

We are not so interested in B's newnonce $N_B(i)$ because it plays a minor role in the protocol. In fact, it could be replaced by a constant.

After these preparatory results, we first concentrate on A's perspective. Assume for a moment both:

$$\begin{array}{l} A\ \mathsf{Sends}\ (\langle \mathsf{name}(A), \{\,\mathsf{newnonce}(n_A), \mathsf{name}(A)\,\}_{K_B}\rangle)\ @\, i \\ \quad \wedge \\ A\ \mathsf{Sees}\ (\{\,\mathsf{seckey}(K), \mathsf{hash}(\mathsf{nonce}(n_A)), \mathsf{nonce}(n_B), \mathsf{name}(B)\,\}_{K_A})\ @\, i+j \end{array}$$

Then we can prove the following results.

1. $n_A = N_A(i)$ and $j > 0$.
2. $\mathsf{HenceForth}\ (\mathsf{Secret}(A, B)(\mathsf{nonce}(n_A)))\ @\, i+j$.
3. $A\ \mathsf{Knows}\ (-)$ of the previous result, *i.e.*
   $A\ \mathsf{Knows}\ (\mathsf{HenceForth}\ (\mathsf{Secret}(A, B)(\mathsf{nonce}(n_A))))\ @\, i+j$.
4. $\exists j'.\, j' \leq j \wedge B\ \mathsf{Sends}\ (\{\,\mathsf{newseckey}(K), \mathsf{hash}(\mathsf{nonce}(n_A)), \mathsf{newnonce}(n_B), \mathsf{name}(B)\,\}_{K_A})\ @\, i+j'$.
5. $A\ \mathsf{Knows}\ (-)$ of the previous result.
6. $\mathsf{HenceForth}\ (\mathsf{Secret}(A, B)(\mathsf{seckey}(K)))\ @\, i+j$.
7. $A\ \mathsf{Knows}\ (-)$ of the previous result.

Next we turn to Bob's perspective, and assume the following two formulas.

$$\begin{array}{l} B\ \mathsf{Sends}\ (\{\,\mathsf{newseckey}(K), \mathsf{hash}(\mathsf{nonce}(n)), \mathsf{newnonce}(n_B), \mathsf{name}(B)\,\}_{K_A})\ @\, i \\ \quad \wedge \\ B\ \mathsf{Sees}\ (K\{\,\mathsf{hash}(n_B)\,\})\ @\, i+j \end{array}$$

Now we can prove:

1. $n_B = N_B(i)$ and $k = \mathsf{SK}(i)$ and $j > 0$.
2. $\mathsf{HenceForth}\ (\mathsf{Secret}(A, B)(\mathsf{seckey}(K)))\ @\, i+j$.
3. $B\ \mathsf{Knows}\ (-)$ of the previous result.
4. $\exists j'.\, j' \leq j \wedge A\ \mathsf{Sends}\ (K\{\,\mathsf{hash}(n_B)\,\})\ @\, i+j'$.
5. $B\ \mathsf{Knows}\ (-)$ of the previous result.
6. $\exists j'.\, j' \leq j \wedge A\ \mathsf{Sees}\ (\{\,\mathsf{seckey}(K), \mathsf{hash}(\mathsf{nonce}(n)), \mathsf{nonce}(n_B), \mathsf{name}(B)\,\}_{K_A})\ @\, i+j'$.

7. B Knows $(-)$ of the previous result.

The two strongest results we have arise by combining assumptions for A and B, in terms of knowledge about what the other agent sends or sees.

$$
\begin{array}{l}
A\ \mathsf{Sends}\ (\langle \mathsf{name}(A), \{\, \mathsf{newnonce}(n_A), \mathsf{name}(A)\,\}_{K_B}\rangle)\ @\, i \\
\quad \wedge \\
A\ \mathsf{Sees}\ (\{\, \mathsf{seckey}(K), \mathsf{hash}(\mathsf{nonce}(n_A)), \mathsf{nonce}(n_B), \mathsf{name}(B)\,\}_{K_A})\ @\, i+j \\
\quad \wedge \\
A\ \mathsf{Knows}\ (B\ \mathsf{Sees}\ (K\{\, \mathsf{hash}(n_B)\,\}))\ @\, i+j+k \\
\quad \Longrightarrow \\
A\ \mathsf{Knows}\ (B\ \mathsf{Knows}\ (\mathsf{HenceForth}\ (\mathsf{Secret}(A,B)(\mathsf{seckey}(K)))))\ @\, i+j+k
\end{array}
$$

In this case it is thus assumed that A knows that B sees the final message. In a dual sense, if B knows that A sent the initial message, we can also obtain a similarly strong secrecy result.

$$
\begin{array}{l}
B\ \mathsf{Knows}\ (A\ \mathsf{Sends}\ (\langle \mathsf{name}(A), \{\, \mathsf{newnonce}(n_A), \mathsf{name}(A)\,\}_{K_B}\rangle))\ @\, i \\
\quad \wedge \\
B\ \mathsf{Sends}\ (\{\, \mathsf{newseckey}(K), \mathsf{hash}(\mathsf{nonce}(n)), \mathsf{newnonce}(n_B), \mathsf{name}(B)\,\}_{K_A})\ @\, i+j \\
\quad \wedge \\
B\ \mathsf{Sees}\ (K\{\, \mathsf{hash}(n_B)\,\})\ @\, i+j+k \\
\quad \Longrightarrow \\
B\ \mathsf{Knows}\ (A\ \mathsf{Knows}\ (\mathsf{HenceForth}\ (\mathsf{Secret}(A,B)(\mathsf{seckey}(K)))))\ @\, i+j+k
\end{array}
$$

These two conclusions are beginning to look like common knowledge (see *e.g.* [8]). They provide a basis for authentication.

7 Logic strengthened for the Otway-Rees

The Otway-Rees key distribution protocol [21] is another well-known target in verification work. There is a substantial gap between Otway-Rees and, say, Needham-Schroeder or BKE, with respect to verification: this is due to the presence of *tickets*. We cope with this new challenge by strengthening the logic that we have already introduced. We will see that this improvement is comparable to the introduction of *honest ideals* [11] in the strand space formalism.

We apply the stronger logical rules to the Otway-Rees protocol to verify its key establishment properties. Additionally, we will observe that an attack presented in [27]—which can happen in the strand space formalism—is impossible in our formalism. This is due to a difference in nonce management between the two formalisms, exploiting that we keep track of agents' possessions.

7.1 Verification challenge in the Otway-Rees protocol

The goal of the Otway-Rees protocol is key-distribution: an initiator A and a responder B aim to establish a session key K_s with the help of a trusted server S. It is assumed that each of A and B has already established a shared long-term secret key K_{AS} and K_{BS} with S. The protocol is informally described as follows.

$$
\begin{array}{lcl}
A \longrightarrow B & : & n, A, B, K_{AS}\{\, n_A, n, A, B\,\} \\
B \longrightarrow S & : & n, A, B, K_{AS}\{\, n_A, n, A, B\,\}, K_{BS}\{\, n_B, n, A, B\,\} \\
S \longrightarrow B & : & n, K_{AS}\{\, n_A, K_s\,\}, K_{BS}\{\, n_B, K_s\,\} \\
B \longrightarrow A & : & n, K_{AS}\{\, n_A, K_s\,\}
\end{array}
\tag{3}
$$

Here n and n_A are nonces freshly generated by A, n_B is a fresh nonce by B, and K_s is a fresh session key generated by S. Recall that the notation $K\{\, m \,\}$ is for symmetric encryption.

A major difference from the Needham-Schroeder or BKE protocol is the existence of *tickets*: a ticket is a subterm of a received message which an agent does not look inside and just passes on to another agent. In the above run of Otway-Rees there are two tickets, $K_{AS}\{\, n_A, n, A, B \,\}$ and $K_{AS}\{\, n_A, K_s \,\}$. Roughly speaking, the responder B can send a message containing any ticket t after B receives the message n, A, B, t. It is not supposed (or possible either) for B to check whether a ticket t is of the expected form $K_{AS}\{\, n_A, n, A, B \,\}$.

Because of tickets, validity of assumptions of most rules in Section 5 are now non-trivial. Hence those rules—although they are still sound—are of little use. For example consider the following secrecy rule (1) in Section 5.3, adapted to the current situation.

$$\frac{\text{“}B \text{ sends new } n_B \text{ at } j \le i\text{, encrypted by } K_{BS}\text{”} \quad \text{“}B, S \text{ send } n_B \text{ only encrypted by } K_{BS}\text{”} \quad \text{“}K_{BS} \text{ is kept secret by } B, S\text{”}}{C \ \mathsf{Possess}\,(\mathsf{nonce}(n_B))\,@\,i \implies C = B \vee C = S} \tag{4}$$

We cannot use this rule to derive secrecy of nonce n_B in the above run (3). Since B is supposed to send anything he receives as a ticket, validness of the second assumption is now questioned.

In this section we shall introduce stronger rules which have weaker premises. These weakened premises are trivially valid even for protocols involving tickets. For example, an alternative for the above secrecy rule is as follows.

$$\frac{\text{“}B \text{ sends new } n_B \text{ at } j \le i\text{, encrypted by } K_{BS}\text{”} \quad \text{“}B, S \textbf{ maintain encryption of } n_B \textbf{ by } K_{BS}\text{”} \quad \text{“}K_{BS} \text{ is kept secret by } B, S\text{”}}{C \ \mathsf{Possess}\,(\mathsf{nonce}(n_B))\,@\,i \implies C = B \vee C = S} \tag{5}$$

The new second assumption “B, S maintain encryption of n_B by K_{BS}” means: if B or S ever sends a message in which n_B appears without encryption, then the agent must have received n_B in a non-encrypted form. This condition is verified easily in the Otway-Rees protocol: if n_B appears not encrypted in a message sent by B, then that must be in a ticket which is received by B beforehand. Hence we can conclude secrecy of the nonce n_B.

This challenge caused by tickets is implicit in [27]. Our way of coping with the challenge can be compared to the one taken there, using the notion of *honest ideals*. The fundamental Theorem 6.11 of [27] about honest ideals essentially proves that the three assumptions of our stronger rule (5) yield the second assumption of our weaker rule (4). In this sense, our strengthening of logical rules in this section corresponds to the introduction of honest ideals like in [11] or [27, Section 6].

7.2 New formulas

Due to the presence of tickets, we now have to take a closer look at an agent A's action of sending: is A just passing a message it has received as a ticket, or is she actually operating on a message *e.g.* decrypting an encryption therein? For this distinction we introduce two new basic formulas OriginatorOf and DecryptSends which are essentially refinements of Sends.

The formula A OriginatorOf m roughly says that m is a part of the sent message but

not a part of any messages received before. Formally,

$$\begin{array}{ll} & A \ \mathsf{OriginatorOf}\ (m)\ (bc, b, i) \\ \stackrel{\text{def}}{=} & \exists m_1 \in \mathsf{MSG}.\ (A \ \mathsf{Sends}\ (m_1) \mathbin{@} i \ \wedge\ m \preceq \mathsf{n2p}(m_1)) \\ \wedge & \forall j \in \mathbb{N}.\ \forall m_2 \in \mathsf{MSG}.\ (j \leq i \ \wedge\ A \ \mathsf{Receives}\ (m_2) \mathbin{@} j \implies m \not\preceq m_2) \end{array}$$

For example, in the Otway-Rees protocol B OriginatorOf $K_{AS}\{\, n_A, K_s \,\}$ is always false since if B sends a message of that form it must be a ticket received before. We can prove the following lemma (or logical rule):

$$\begin{array}{ll} \text{(ORISAYS)} & \mathsf{OriginatorOf}\ \text{implies}\ \mathsf{Says}. \\ & A \ \mathsf{OriginatorOf}\ m \mathbin{@} i \\ \implies & \exists m' \in \mathsf{MSG}.\ (\ A \ \mathsf{Says}\ (m') \mathbin{@} i \ \wedge\ m = \mathsf{n2p}(m')\ ) \end{array}$$

The label (ORISAYS) will be used later in Appendix B.

For an arbitrary set $\mathcal{K}$ of keys, the intuitive meaning of a formula A DecryptSends $(m, \mathcal{K})$ is that A decrypts an encryption (by a key from a set $\mathcal{K}$) of m and sends m. A typical example of m is a nonce. Formally,

$$\begin{array}{ll} & A \ \mathsf{DecryptSends}\ (m, \mathcal{K})\ (bc, b, i) \\ \stackrel{\text{def}}{=} & \exists m_1 \in \mathsf{MSG}.\ (A \ \mathsf{Sends}\ (m_1) \mathbin{@} i \ \wedge\ m \preceq \mathsf{n2p}(m_1) \ \wedge\ m \not\preceq_{\mathcal{K}} \mathsf{n2p}(m_1)) \\ \wedge & \forall j \in \mathbb{N}.\ \forall m_2 \in \mathsf{MSG}.\ (j \leq i \ \wedge\ A \ \mathsf{Receives}\ (m_2) \mathbin{@} j \implies m \preceq_{\mathcal{K}} m_2) \end{array}$$

Recall that $m \preceq_{\mathcal{K}} m'$ means: every occurrence of m in m' is under encryption by some key in $\mathcal{K}$. Hence $m \not\preceq_{\mathcal{K}} m'$ means that there is at least one occurrence of m in m' without encryption by any key from $\mathcal{K}$. Therefore the condition $m \preceq \mathsf{n2p}(m_1)$ above is in fact redundant (but still there for the ease of understanding).

7.3 Strengthened logical rules

We shall present two strengthened logical rules which play crucial roles in the verification of Otway-Rees. Their soundness against our semantics is proved in PVS. Other rules which are less significant are presented in Appendix A.

Our first rule is for authentication and called *incoming authentication test*: the name is after the corresponding lemma [12] for the strand space formalism.

$$\begin{array}{ll} \text{(INCTEST)} & \text{Incoming authentication test. } \mathcal{A} \text{ is a set of agents.} \\ & [\text{1. Challenge}] \quad A \ \mathsf{Sends}\ m_c \mathbin{@} i_c \quad \wedge \quad \mathsf{newnonce}(n) \preceq m_c \\ \wedge & [\text{2. Response}] \quad A \ \mathsf{Sees}\ m_r \mathbin{@} i_c + i_w \quad \wedge \quad \mathsf{nonce}(n) \preceq K\{\, m_1 \,\} \preceq m_r \\ \wedge & [\textbf{3. } K\{\, m_1 \,\} \textbf{ does not originate from } A \textbf{ herself}] \\ & \quad \forall j \in [i_c,\, i_c + i_w].\quad \neg(A \ \mathsf{OriginatorOf}\ K\{\, m_1 \,\} \mathbin{@} j) \\ \wedge & [\text{4. } K \text{ is secretly shared}] \quad \forall j \in \mathbb{N}.\quad \mathsf{Secret}(\mathcal{A})(K) \mathbin{@} j \\ \implies & [\text{Authentication result}] \\ & \quad \exists B \in \mathcal{A} \setminus \{A\}.\quad \exists j \in (i_c,\, i_c + i_w).\quad B \ \mathsf{OriginatorOf}\ K\{\, m_1 \,\} \mathbin{@} j \end{array}$$

The basic idea is a common challenge-response style authentication. The random value $\mathsf{nonce}(n)$ is not encrypted in the form of $K\{\, m_1 \,\}$ in the (outgoing) challenge, because of the assumption 3: in particular,

$$\neg(A \ \mathsf{OriginatorOf}\ K\{\, m_1 \,\} \mathbin{@} i_c)$$

Note that $\mathsf{nonce}(n) \preceq m_1$ by the assumption 2. However at stage i_r, A receives an (incoming) response which contains $K\{\, m_1 \,\}$. This so-called *challenge component* $K\{\, m_1 \,\}$ must originate from someone who possesses the key K, hence someone in $\mathcal{A}$ by the assumption 4. By the assumption 3 it must not be A herself. Moreover, the creation of $K\{\, m_1 \,\}$ must be after the creation of $\mathsf{nonce}(n)$. This is an *incoming* test because the incoming response involves particular encryption.

It is straightforward to get a variation of this rule in which $\mathsf{nonce}(n)$ and $\mathsf{newnonce}(n)$ are replaced by $\mathsf{seckey}(K_s)$ and $\mathsf{newseckey}(K_s)$. Less straightforward is a rule for *outgoing* authentication test [12]—where a nonce is encrypted in the outgoing challenge but not in the incoming response. We have the rule in the PVS specification [15] but it is not used for the verification of Otway-Rees.

Compared to the similar authentication rule earlier in Section 5.2, the third assumption is weakened from "A does not say $K\{\, m_1 \,\}$"—which is not trivially valid in Otway-Rees due to tickets—to "$K\{\, m_1 \,\}$ does not *originate* from A".

Our second strengthened rule is a secrecy rule corresponding to the above (5). For future reference we present a rule for secrecy of a freshly generated key: the corresponding rule for secrecy of a nonce is similar.

(SKSEC) Secrecy of session keys encrypted with uncompromised keys.
$\mathcal{A}$ is a set of agents and $\mathcal{K}$ is a set of keys.

$$
\begin{array}{ll}
 & [\text{1. Long-term keys are kept secret}] \\
 & \quad \forall j \le i.\ \forall K_l \in \mathcal{K}.\ \forall A \in \mathcal{A}.\ (\ A\ \mathsf{Possess}\ (K_l)\ @\, j \implies A \in \mathcal{A}\ ) \\
\wedge & [\textbf{2. Legitimate parties keep encryptions}] \\
 & \quad \forall j \in [i_0,\, i].\ \forall A \in \mathcal{A}.\ \neg(\ A\ \mathsf{DecryptSends}\ (\mathsf{seckey}(K_s),\, \mathcal{K})\ @\, j\ ) \\
\wedge & [\text{3. Generation of session key}] \\
 & \quad i_0 \le i\ \wedge\ A_0 \in \mathcal{A}\ \wedge\ A_0\ \mathsf{Says}\ (\mathsf{newseckey}(K_s))\ @\, i_0 \\
\implies & [\text{Secrecy result}]\quad \forall A \in \mathcal{A}.\ (\ A\ \mathsf{Possess}\ (K_s)\ @\, i \implies A \in \mathcal{A}\ )
\end{array}
$$

As explained in Section 7.1, the assumption 2 is weaker than the corresponding condition "agents in $\mathcal{A}$ send K_s only encrypted by a key in $\mathcal{K}$". This is crucial for the rule to be usable for Otway-Rees.

7.4 Verification of the Otway-Rees protocol

Now we shall see that the refined rules are appropriately adapted to the Otway-Rees protocol and to other protocols that make similar use of tickets. Since the Otway-Rees protocol is aimed at key-distribution, we want to establish the following security properties.

- *Secrecy of the session key* K_s: K_s is known only by the legitimate agents A, B, and S.
- *Freshness of* K_s: K_s was generated recently by S, say "after B sent the message ...". (Algorithmic) cryptology says that the more data a key is used to encrypt, the more likely the key gets compromised by cryptanalysis. This type of attacks via cryptanalysis are not present in the Dolev-Yao model, hence not in our semantics nor in the strand space formalism [27]. Nevertheless, freshness of a session key is sufficient to mitigate those risks [13].
- *Agreement on* K_s: B knows, after his run of protocol as a responder, that A was also running the protocol as an initiator with the same data items; in particular A has obtained the same session key. This is B's guarantee on agreement, and we can state A's guarantee in a similar manner.

As a showcase, in Appendix B we sketch the derivation (in PVS) of B's guarantee of the first two properties. Namely:

$$\begin{array}{ll} & B\ \mathsf{Sends}\ (\ n, A, B, t_1, K_{BS}\{\ \mathsf{newnonce}(n_B), n, A, B\ \}\ )\ @\, i \\ \wedge & B\ \mathsf{Sees}\ (\ K_{BS}\{\ n_B, K_s\ \}\ )\ @\, i+j \\ \Longrightarrow & B\ \mathsf{knows} \left[\begin{array}{ll} \mathsf{HenceForth} \neg\, (\mathsf{spy\ Possess}\ (K_s))\ @\, i+j & \wedge \\ \exists k \in [i,\ i+j]. & S\ \mathsf{Says}\ (\mathsf{newseckey}(K_s))\ @\, k \end{array}\right] @\, i+j \end{array} \tag{6}$$

In this formula t_1 is an arbitrary message. The proof of this secrecy property is rather complicated because it involves authentication in the following way. The secrecy is easy if the session key is issued by the legitimate server S under proper encryptions. The hard part is that B must be sure this is indeed the case. The guarantee for A is proved similarly.

The third property of agreement on distributed keys is the main topic in the next Section 7.5.

7.5 Prevention of certain replay attacks

There is an attack on the Otway-Rees protocol presented in [27] which distributes different session keys to A and B by replaying a request to S. Due to this attack the agreement of session keys fails for Otway-Rees. Here spy_X denotes the intruder impersonating a legitimate agent X.

$$\begin{array}{rl} A \rightarrow B: & n, A, B, K_{AS}\{\ n_A, n, A, B\ \} \\ B \rightarrow \mathsf{spy}_S: & n, A, B, K_{AS}\{\ n_A, n, A, B\ \}, K_{BS}\{\ n_B, n, A, B\ \} \\ \mathsf{spy}_B \rightarrow S: & n, A, B, K_{AS}\{\ n_A, n, A, B\ \}, K_{BS}\{\ n_B, n, A, B\ \} \\ S \rightarrow B: & n, K_{AS}\{\ n_A, K_s\ \}, K_{BS}\{\ n_B, K_s\ \} \\ B \rightarrow \mathsf{spy}_A: & n, K_{AS}\{\ n_A, K_s\ \} \\ \mathsf{spy}_B \rightarrow S: & n, A, B, K_{AS}\{\ n_A, n, A, B\ \}, K_{BS}\{\ n_B, n, A, B\ \} \\ S \rightarrow \mathsf{spy}_B: & n, K_{AS}\{\ n_A, K'_s\ \}, K_{BS}\{\ n_B, K'_s\ \} \\ \mathsf{spy}_B \rightarrow A: & n, K_{AS}\{\ n_A, K'_s\ \} \end{array} \tag{7}$$

We emphasise that this attack (possible in the original strand space model) is no longer possible in our semantics. In fact we have proved in PVS A's guarantee on agreement of session keys.

$$\begin{array}{ll} & i_0 \le i_1 \le i_2 \le i_3 \\ \wedge & A\ \mathsf{Sends}\ (\mathsf{newnonce}(n), A, B, K_{AS}\{\ \mathsf{newnonce}(n_A), \mathsf{newnonce}(n), A, B\ \})\ @\, i_0 \\ \wedge & A\ \mathsf{Knows}\ \Big(B\ \mathsf{Sends}\ (n, A, B, t_1, K_{BS}\{\ \mathsf{newnonce}(n_B), n, A, B\ \})\Big)\ @\, i_1 \\ \wedge & A\ \mathsf{Knows}\ (B\ \mathsf{Sees}\ K_{BS}\{\ n_B, K_s\ \})\ @\, i_2 \\ \wedge & A\ \mathsf{Sees}\ K_{AS}\{\ n_A, K'_s\ \}\ @\, i_3 \\ \Longrightarrow & K_s = K'_s\ \wedge\ A\ \mathsf{Knows}\ (\mathsf{HenceForth}\ (\ \mathsf{Secret}(\{A, B, S\})(K_s)\ ))\ @\, i_3 \end{array} \tag{8}$$

It states: if A has properly finished her role in a run and if A somehow knows that B has also finished his role with the matching nonce n (which acts as an identifier of the run), then A is sure that the distributed session keys agree. The guarantee for B is formulated in a similar way and proved in PVS.

How does this difference arise? In our semantics, the server S keeps track of all the nonces to prevent replays. In particular, S stores among its possessions all n's (identifiers of runs) in the requests, and S does not react to a request containing a nonce n which S has

ever seen. This additional feature of our semantics prevents the 7th message of the above attack (7) from being sent.

This feature is possible in our semantics because each agent has exactly one "strand" (with corresponding possessions), in which it can interleave multiple runs of the protocol. It is not the case in the original strand space formalism: there an agent can have multiple strands, each of which corresponds to the agent's role in a single run. There is no mechanism which allows different strands to communicate with each other. In this sense, in the strand space formalism an agent is "multi-threaded", as opposed to the single-threaded model in our semantics. Actual implementation determines which semantics is more appropriate, hence enables/disables certain attacks such as (7).

Our model of nonce management can be implemented by including a timestamp as a part of a nonce: we can fix a certain time span within which S keeps track of nonces, and S responds only to nonces with timestamps in that span.

8 Conclusions and further work

We have achieved a unification in the area of security protocols by combining the best of several approaches [27, 4, 23, 25], namely a tool-supported sound logic. It is a further intellectual challenge to include process-based approaches (such as [1]) within this semantical framework.

More practically-oriented further work lies in the application of this approach to other, more complex protocols, and to other properties than secrecy. This is ongoing work, that may require adaptation and/or extension of the current semantics. Once a stable and useful set of (semantic) rules has been identified, one may use it to formulate a proper (syntactic) logic for security protocols. Another line of work, possibly more suitable, is to use this formalised framework mainly to establish soundness of logical rules and to perform actual verifications with more automatic tools.

As mentioned in Section 4.3, formal analysis of other security properties such as anonymity is another possible direction of the further research. We may re-use the current PVS infrastructure there.

Acknowledgements

Thanks are due to Erik Poll and Martijn Warnier for helpful comments and discussions, and also to the referees for their constructive remarks and suggestions.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols. *Journ. ACM*, 148(1):1–70, 1999.

[2] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Number 53 in Tracts in Theor. Comp. Sci. Cambridge Univ. Press, 2001.

[3] P. Broadfoot and A. Roscoe. Proving security protocols with model checkers by data independence techniques. *Journ. of Computer Security*, 7:147–190, 1999.

[4] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proc. Royal Soc.*, Series A, Volume 426:233–271, 1989.

[5] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Univ. of York.
www-users.cs.york.ac.uk/~jac/papers/drareviewps.ps, 1997.

[6] B. Davey and H. Priestley. *Introduction to Lattices and Order*. Math. Textbooks. Cambridge Univ. Press, 1990.

[7] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, 1983.

[8] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge, MA, 1995.

[9] F. Garcia, I. Hasuo, W. Pieters, and P. van Rossum. Provable anonymity. In R. Küsters and J. Mitchell, editors, *3rd ACM Workshop on Formal Methods in Security Engineering (FMSE05)*, pages 63–72, Alexandria , VA, U.S.A., November 2005. ACM Press.

[10] R. Goldblatt. *Logics of Time and Computation*. CSLI Lecture Notes 7, Stanford, 2nd rev. edition, 1992.

[11] J. Guttman, J. Herzog, and F.T. Fàbrega. Honest ideals on strand spaces. In *Computer Security Foundations Workshop*, 1998.

[12] J. Guttman and F. Thayer Fàbrega. Authentication tests and the structure of bundles. *Theor. Comp. Sci.*, 283(2):333–380, 2002.

[13] J. Guttman. Key compromise, strand spaces, and the authentication tests. In *Mathematical Foundations of Programming Semantics 17*, volume 47 of *Elect. Notes in Theor. Comp. Sci.*, pages 1–21. Elsevier, Amsterdam, 2001.

[14] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *J. Comput. Secur.*, 11(2):217–244, 2003.

[15] B. Jacobs and I. Hasuo. PVS sources for semantics and logic of security protocols.
www.cs.ru.nl/B.Jacobs/PVS/protocols-3.0.zip.

[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 1055 in Lect. Notes Comp. Sci., pages 147–166. Springer, Berlin, 1996.

[17] C. Meadows. The NRL protocol analyzer: An overview. *Journ. of Logic Programming*, 26(2):113–131, 1996.

[18] C. Meadows. Identifying potential type confusion in authenticated messages. Workshop on Foundations of Computer Security, Techn. Rep. DIKU-02-12, Dep. Comp. Sci., Univ. Copenhagen, 2002.

[19] J. Millen. A necessarily parallel attack. Workshop on Formal Methods and Security Protocols.
www.csl.sri.com/users/millen/, 1999.

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

[21] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[22] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. on Softw. Eng.*, 21(2):107–125, 1995.

[23] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journ. of Computer Security*, 6:85–128, 1998.

[24] P. Ryan, S. Schneider, M. Goldschmith, G. Lowe, and A. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

[25] P. Syverson. Towards a strand semantics for authentication logics. In S. Brookes, A. Jung, M. Mislove, and A. Scedrov, editors, *Mathematical Foundations of Progamming Semantics*, number 20 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1999.

[26] P. Syverson and I. Cervesato. The logic of authentication protocols. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Foundations of Security Analysis and Design*, number 2171 in Lect. Notes Comp. Sci., pages 63–136. Springer, Berlin, 2001.

[27] F. Thayer Fàbrega, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journ. of Computer Security*, 7:191–230, 1999.

A Logical rules for Otway-Rees

The following logical rules—together with (ORISAYS), (INCTEST) and (SKSEC) in Sections 7.2 and 7.3—are used for the verification of Otway-Rees.

The following rule for authentication is called *unsolicited authentication test* after [12], as opposed to *incoming* or *outgoing* test. It is unsolicited since the incoming "response" is not really a response to any specific challenge.

(USTEST) Unsolicited authentication test.

$$\begin{array}{rl} & [\text{Response}] \quad A \ \mathsf{Receives}\ (m) \,@\, i \quad \wedge \quad K\{\, m_1 \,\} \preceq m \\ \wedge & [K \text{ is secretly shared}] \quad \mathsf{Secret}(\mathcal{A})(K) \,@\, i \\ \Longrightarrow & [\text{Authentication result}] \\ & \quad \exists B \in \mathcal{A}. \quad \exists j \in [0, i]. \quad B \ \mathsf{OriginatorOf}\ K\{\, m_1 \,\} \,@\, j \end{array}$$

The remaining rules state basic facts about our semantics. They are relatively easier to prove and used in the verification of Otway-Rees.

(NRAN) Randomness of newnonce.

$$\begin{array}{lll} & A\ \mathsf{Sends}\ (m_A)\ @\ i_A & \wedge\quad \mathsf{newnonce}(n) \preceq m_A \\ \wedge & B\ \mathsf{Sends}\ (m_B)\ @\ i_B & \wedge\quad \mathsf{newnonce}(n) \preceq m_B \\ \Longrightarrow & A = B \quad \wedge \quad i_A = i_B & \end{array}$$

(SKRAN) Randomness of newseckey.

This is the same as (NRAN) but nonces are replaced by seckeys.

(LTKSEC) Secrecy of never-sent keys (such as long-term symmetric keys).

$$\begin{array}{lll} & [\text{Initially } K \text{ is secretly shared}] & \mathsf{Secret}(\{A,S\})(K)\ @\ 0 \\ \wedge & [A \text{ never sends } K] & \forall i \in \mathbb{N}.\quad (A\ \mathsf{Sends}\ (m)\ @\ i \implies K \not\preceq m) \\ \wedge & [S \text{ never sends } K] & \forall i \in \mathbb{N}.\quad (S\ \mathsf{Sends}\ (m)\ @\ i \implies K \not\preceq m) \\ \Longrightarrow & [\text{Secrecy result}] & \forall i \in \mathbb{N}.\quad \mathsf{Secret}(A,S)(K) i \end{array}$$

B Security proof of Otway-Rees

In the following derivation of (6), (PROT) designates use of the specification of the Otway-Rees protocol. Free variables are immediately replaced by fresh (Skolem) constants: for example the steps 5 and 6 below can be read as "there exists $i_1 \in \mathbb{N}$ such that ...".

1.	B Sends $(n, A, B, t_1, K_{BS}\{\ \mathsf{newnonce}(n_B), n, A, B\ \})\ @\ i$	Assumption
2.	B Sees$(K_{BS}\{\ n_B, K_s\ \})\ @\ i+j$	Assumption
3.	$\forall k \in [i,\ i+j].\quad \neg(\ B$ OriginatorOf $K_{BS}\{\ n_B, K_s\ \}\ @\ k\ )$	(PROT)
4.	$\forall k \in \mathbb{N}.\quad \mathsf{Secret}(\{B,S\})(K_{BS})\ @\ k$	(PROT), (LTKSEC)
5.	$i < i_1 < i+j$	1, 2, 3, 4, (INCTEST)
6.	S OriginatorOf $K_{BS}\{\ n_B, K_s\ \}\ @\ i_1$	1, 2, 3, 4, (INCTEST)
7.	S Says $m_1\ @\ i_1$	6, (ORISAYS)
8.	$K_{BS}\{\ n_B, K_s\ \} = \mathsf{n2p}(m_1)$	6, (ORISAYS)
9.	S Says $K_{BS}\{\ n_B, \mathsf{newseckey}(K_s)\ \}\ @\ i_1$	7, 8, (PROT)
10.	S Receives $n', D, B, K_{DS}\{\ n_D, n', D, B\ \}, K_{BS}\{\ n_B, n', D, B\ \}\ @\ i_1 - 1$ $\vee\quad S$ Receives $n', B, D, K_{BS}\{\ n_B, n', B, D\ \}, K_{DS}\{\ n_D, n', B, D\ \}\ @\ i_1 - 1$	9, (PROT)

We refute the second disjunct of 10 as follows.

A-1.	S Receives $n', B, D, K_{BS}\{\ n_B, n', B, D\ \}, K_{DS}\{\ n_D, n', B, D\ \}\ @\ i_1 - 1$	Assumption
A-2.	$E = B \quad \vee \quad E = S$	A-1, 4, (USTEST)
A-3.	E OriginatorOf $K_{BS}\{\ n_B, n', B, D\ \}\ @\ i_2$	A-1, 4, (USTEST)
A-4.	$\forall k \in \mathbb{N}.\quad \neg(\ S$ OriginatorOf $K_{BS}\{\ n_B, n', B, D\ \}\ @\ k\ )$	(PROT)
A-5.	B OriginatorOf $K_{BS}\{\ n_B, n', B, D\ \}\ @\ i_2$	A-2, A-3, A-4
A-6.	B Says $K_{BS}\{\ \mathsf{newnonce}(n_B), n', B, D\ \}\ @\ i_2$	A-5, (PROT), (ORISAYS)
A-7.	$i_2 = i \quad \wedge \quad n' = n$	1, A-6, (NRAN)
A-8.	B Receives $n, A, B, t_1\ @\ i-1$	1, (PROT)
A-9.	$\mathsf{newnonce}(n_B) \not\preceq t_1$	A-8, $\mathsf{newnonce}(n_B)$ only occurs in sent messages
A-10.	$A = B$, hence $\bot$	1, A-6, A-7, A-9
A.	$\neg(\ S$ Receives $n', B, D, K_{BS}\{\ n_B, n', B, D\ \}, K_{DS}\{\ n_D, n', B, D\ \}\ @\ i_1 - 1\ )$	A-1, A-10

We turn back to the main line.

11.	S Receives $n', D, B, K_{DS}\{\, n_D, n', D, B \,\}, K_{BS}\{\, n_B, n', D, B \,\} @ i_1 - 1$	10, A
12.	$D = A \quad \wedge \quad n' = n$	Like A-10 is derived from A-1, using (USTEST)
13.	S Sends $n, K_{AS}\{\, n_A, \mathsf{newseckey}(K_s) \,\}, K_{BS}\{\, n_B, \mathsf{newseckey}(K_s) \,\} @ i_1$	11, 12, (PROT), 9
14.	$\forall k \in \mathbb{N}. \quad \mathsf{Secret}(\{A, S\})(K_{AS}) @ k$	(PROT), (LTKSEC)
15.	$\forall k \in \mathbb{N}. \quad \forall K \in \{K_{AS}, K_{BS}\}. \quad \forall F \in \mathsf{AG}.$ $\big(\, F \text{ Possess } K @ k \quad \Longrightarrow \quad F \in \{A, B, S\} \,\big)$	4, 14
16.	$\forall k \in \mathbb{N}. \forall F \in \{A, B, S\}. \neg(\, F \text{ DecryptSends } (K_s, \{K_{AS}, K_{BS}\}) @ k \,)$	(PROT), (SKRAN)
17.	$i + j \leq l$	Expanding HenceForth
18.	$i_1 \leq l \ \wedge \ S \in \{A, B, S\} \ \wedge \ S$ Says $\mathsf{newseckey}(K_s) @ i_1$	5, 17, 13
19.	$\forall F \in \mathsf{AG}. \ (\, F \text{ Possess } K_s @ l \quad \Longrightarrow \quad F \in \{A, B, S\} \,)$	15, 16, 18, (SKSEC)
20.	$\neg(\mathsf{spy}$ Possess $K_s @ l)$	19
21.	$\forall l \geq i + j. \quad \neg(\mathsf{spy}$ Possess $K_s @ l)$	17, 20
22.	HenceForth $(\, \neg(\mathsf{spy}$ Possess $K_s) \,) @ i + j$	21
23.	$\exists k \in [i,\ i + j]. \quad S$ Says $\mathsf{newseckey}(K_s) @ k$	18, 5

By steps 22 and 23, we have shown that

$$
\begin{array}{lll}
 & B \text{ Sends } \big(\, n, A, B, t_1, K_{BS}\{\, \mathsf{newnonce}(n_B), n, A, B \,\}\,\big) @ i \\
\wedge & B \text{ Sees}\big(\, K_{BS}\{\, n_B, K_s \,\}\,\big) @ i + j \\
\Longrightarrow & \left[\begin{array}{l} \big(\, \mathsf{HenceForth}\ \neg(\mathsf{spy}\ \mathsf{Possess}\ K_s) @ i + j \,\big) \quad \wedge \\ \big(\, \exists k \in [i,\ i + j]. \quad S\ \mathsf{Says}\ (\mathsf{newseckey}(K_s)) @ k \,\big) \end{array}\right]
\end{array}
$$

By applying rules on Knows we obtain B's guarantee of secrecy and freshness of the session key K_s.