Semantics and Logic for Security Protocols

Bart Jacobs
Department of Computer Science, Radboud University
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands.
Email: B.Jacobs@cs.ru.nl URL: http://www.cs.ru.nl/B.Jacobs

September 10, 2004

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 modal operators (describing the knowledge of participants). The logic's semantics is based on strand spaces. Several (secrecy) formulas are proven for the Needham-Schroeder(-Lowe) and bilateral key exchange 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 [14, 3], see also [20]) 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 [8], 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 [18], and allows verifications of actual protocols with tool support, like in [19]. Hence, in a sense, it combines the best of these three approaches [8, 4, 19].

The following aspects distinguish our formalisation.

- The logic involves both (linear) temporal operators (like henceforth) and modal 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) and "used" nonces, with an associated requirement that "new" nonces (or keys) are globally fresh.

This paper makes essential use of earlier related work [8, 4, 19, 21] but differs in several ways.

- The strand spaces of [8] 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.
- 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 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 [19] 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 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 [21] (see also [22, Section 6.2]), namely to formulate a BAN-like logic on top of a strand space semantics. However, the reference [21] 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 [21].

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 future extensions and adaptations of the model.

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 [12] 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 [21]). The contribution of this paper is however, that this idea has been elaborated in full detail, been formalised, and put to use succesfully in concrete examples. Doing so required a subtle balancing of the various possible requirements and formulations—and a non-trivial amount of PVS work!

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. In the end, the bilateral key exchange protocol (from [5, §§6.6.6]) is analysed in Section 6 as an application.

2 Results about the Needham-Schroeder public key protocol

Here we consider the well-known Needham-Schroeder public key protocol from [17], which was shown to be flawed in [13]. 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).

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

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.

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

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 [13]. It can however be proved for the repaired "Needham-Schroeder-Lowe" protocol:

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

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

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

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 $\big(X\ \mathsf{Knows}\ (\varphi)\big) \Rightarrow \varphi$ 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 $A\ \mathsf{Sends}\ (m) \mathbin{@} i$ 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 parties 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. 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. 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 $k\{\,m_1, m_2\,\}$. Signed messages are at this stage not fully 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.* [15]).

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 $\mathsf{nonce}(n) \preceq \mathsf{hash}(\mathsf{nonce}(n))$. 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[1], 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.

- *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, as $m \preceq_{\mathsf{pe}(k)} U$ and $m \preceq_{\mathsf{se}(k)} U$, where the relation is required to hold for some $m' \in U$.

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.

[1]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.

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` [19]. 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 [12] 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:

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

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 [19]—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:

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

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.

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:

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

The same result holds for $\mathsf{seckey}(k)$ in place of $\mathsf{nonce}(n)$.

4 Strands and bundles

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 [8]). 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 induction:

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

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:

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

  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:

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

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.
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:

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

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.

4.1 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 types, namely of 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	$\cdots \in \mathsf{AG}$
stage 0	$-\langle m_1, m_2\rangle$	$+\langle m_1, m_2\rangle$	$\perp$	
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 constraing $bc \in \mathsf{BunCst}$. 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. 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 [16].)

4. Received message 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 garanteed only that the spy receives an 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 the standard 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 **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 [8].

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 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 satify the bundle constraint bc. Then:

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

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

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

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

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

The modal "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 than says that $\mathsf{HenceForth}\,\big(\varphi\big) @ i$ 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.

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

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, $\mathsf{M2P}$ is monotone, so the implication $A\ \mathsf{Possess}\,\big(m\big) \Rightarrow \mathsf{HenceForth}\,\big(A\ \mathsf{Possess}\,\big(m\big)\big)$ holds. Hence the following rule is sound.

$$\frac{A\ \mathsf{Possess}\,\big(m\big)}{\mathsf{HenceForth}\,\big(A\ \mathsf{Possess}\,\big(m\big)\big)}$$

Since $\mathsf{M2P}$ yields $\mathbf{P}$-closed subsets of terms we also have the following four closure rules.

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

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

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

Our next formulas are about sending.

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

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.

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

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

There are analogous rules for $\mathsf{newseckey}(k)$ in place of $\mathsf{newnonce}(n)$.

Now we turn to receiving messages.

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

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:

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

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

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.

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

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

5.2 Rules for nonces and public encryptions

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 [11]. 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.

$$\frac{\text{“}A \text{ sends at } j\text{”} \quad \text{“}A \text{ sees at } i \geq j\text{”} \quad \text{“}A \text{ no says after } j\text{”} \quad \text{“}B\text{'s privkey secret until } i\text{”}}{\exists j_1.\, j < j_1 \leq i \wedge \text{“}B \text{ sees at } j_1\text{”} \wedge \exists j_2, C.\, j_1 < j_2 \leq i \wedge \text{“}C \neq A \text{ sends”}}$$

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 $\mathsf{Sends}\big(m\big) @ j$ 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 $\mathsf{Sees}\big(m_2\big) @ i$ where $i \geq j$ and not: $\mathsf{nonce}(n) \preceq_{\mathsf{se}(k)} m_2$. The latter says that the nonce n does not occur encrypted under k in m_2; 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 no says after j" means not: A $\mathsf{Says}\big(\mathsf{nonce}(n)\big) @ j'$, 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 $\mathsf{Sees}\big(\langle \mathsf{nonce}(n), \mathsf{n2p}(m_1)\rangle\big) @ j_1$ 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 $C \neq A \wedge C$ $\mathsf{Sends}\big(m_3\big) @ j_2$ 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-trival 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 to the above one for new secret keys instead of nonces. 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.

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

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

- "A_1 sends new n at $j \leq i$" means $A_1 \,\mathsf{Says}\,\big(\mathsf{newnonce}(n)\big) \mathbin{@} j$ 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 $A_1 \,\mathsf{Sends}\,(m) \mathbin{@} j'$ 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: $\mathsf{Secret}(A_p)(\mathsf{privkey}(\overline{k_p})) \mathbin{@} i'$ 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 [21],

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

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 sofar, *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:

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

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]. The protocol involves the following steps.

$$\begin{array}{lcl} A \longrightarrow B & : & A, \{\, n_A, A \,\}_{K_B} \\ B \longrightarrow A & : & \{\, K, \mathsf{hash}(n_A), n_B, B \,\}_{K_A} \\ A \longrightarrow B & : & K\{\, \mathsf{hash}(n_B) \,\} \end{array} \qquad (1)$$

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 is 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, so that its rule is:

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

The rules for Alice and Bob correspond to the protocol rules (1) 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.

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

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 (1) 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.

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 $K\{\,\mathsf{hash}(n)\,\}$ that is built from the incoming nonce and key. The condition that $\mathsf{nonce}(n)$ and $\mathsf{seckey}(K)$ are not possessed yet is included to prevent replays, and is actually used in the verification.

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

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

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 (1), 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 4.1. 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,

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

This holds because A and B never sent out their private keys. Next we have

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

This follows from the secrecy rules in Subsection 5.3. There is a similar implication for B with his session key:

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

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:

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

Then we can prove the following results.

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

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

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

Now we can prove:

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

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

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.

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

These two conclusions are beginning to look like common knowledge (see *e.g.* [9]). They provide a basis for authentication.

7 Conclusions and further work

We have achieved a unification in the area of security protocols by combining the best of several approaches ([8, 4, 19, 21]), 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.

Acknowledgements

Thanks are due to Martijn Warnier for helpful discussions.

References

[1] M. Abadi and A.D. 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.J. Broadfoot and A.W. 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.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Math. Textbooks. Cambridge Univ. Press, 1990.

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

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

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

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

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

[12] B. Jacobs. Pvs sources for semantics and logic of security protocols. `www.cs.ru.nl/B.Jacobs/PVS/protocols-2.0.zip`.

[13] 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.

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

[15] 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.

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

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

[18] S. Owre, J.M. 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.

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

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

[21] 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.
`www.elsevier.nl/locate/entcs/volume20.html`.

[22] 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.