Operational Semantics of Security Protocols

Cas Cremers and Sjouke Mauw

Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands.

Abstract. Based on a concise domain analysis we develop a formal semantics of security protocols. Its main virtue is that it is a generic model, in the sense that it is parameterized over e.g. the intruder model. Further characteristics of the model are a straightforward handling of parallel execution of multiple protocols, locality of security claims, the binding of local constants to role instances, and explicitly defined initial intruder knowledge. We validate our framework by analysing the Needham-Schroeder-Lowe protocol.

1 Introduction

Security protocols are often expressed in the form of a diagram displaying the interactions between the principals, such as a Message Sequence Chart. The MSC in Figure 1 describes perhaps the most well-known example of a flawed security protocol. We will explain the details in Section 4. The protocol was developed in 1978 by Roger Needham and Michael Schroeder [1] and proven correct with BAN logic [2] in 1989. In 1995 Gavin Lowe found an attack on the protocol [3], because he assumed a more powerful intruder model, allowing agents to conspire with the intruder. This so-called man-in-the-middle attack is displayed in Figure 2. Currently, this situation is explained by pointing at a shift of the assumptions on the environment of the system: from a trusted local network that should be protected against external threats to a network with internal attackers.

This example clearly shows that a theory of security protocols should be flexible enough to vary over the parameters that determine the problem, such as the intruder model. Looking at Figure 1 it is clear that this informal protocol specification states nothing about the precise intruder model assumed. In fact, more information is lacking. Information which is needed to precisely understand the meaning of this diagram. How does an agent check, for instance, if an incoming message satisfies the expected message format? If we assume that he will not check the types of the messages, yet another attack will become viable, which is called a type-flaw attack.

It is our goal to give an unambiguous and generic description of the interpretation of such security protocols and what it means for a protocol to ensure some security property. Although the security protocol takes the shape of a Message Sequence Chart, there is so much additional structure in the problem that we cannot rely on the MSC semantics to provide an answer. Therefore, we will define a formal semantics of security protocols.

Our first step to come to a formal semantics is to conduct a concise domain analysis (loosely following [4]). The purpose of this step is to informally sketch the issues that make up the problem space and its boundaries. We will identify the points of variation and decide whether these are considered as parameters of the problem or that design decisions have to be made. In this process we are guided by the following starting points. First of all, the formal model must be generic (e.g. over the intruder model). Second, the formal model should offer a framework to verify security protocols, both manually and with computer support. Third, the formal model should be easily extendable with additional features (such as forward secrecy) to make it applicable to a wide range of problems. Finally, the formal model should enable the development of meta-theory (e.g. compositionality properties of security protocols). We have chosen to define an operational semantics based on state transitions.

The rest of this paper is structured as follows. In Section 2 we will conduct a short domain analysis and introduce the basic concepts. Section 3 describes an operational semantics of security protocols, based on the domain analysis. We will validate our semantical approach by formally analysing the Needham-Schroeder protocol in Section 4. In Section 5 we discuss the relation between our approach and other published models and in Section 6 we will summarise our results and provide an outlook on future research.

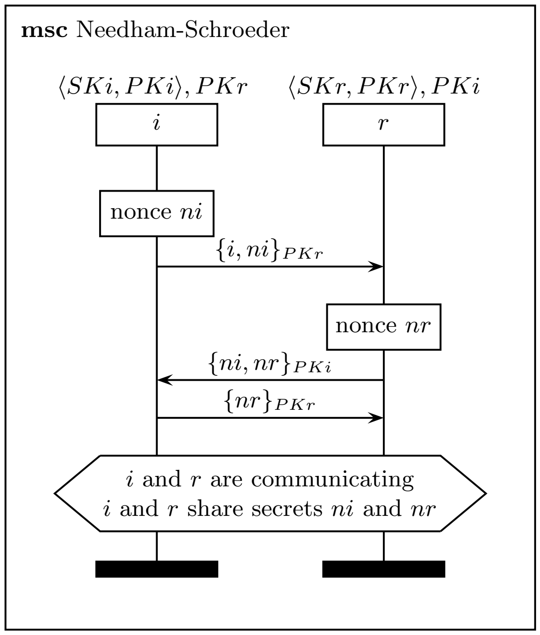
msc Needham-Schroeder
$\langle SKi, PKi\rangle, PKr$
$\langle SKr, PKr\rangle, PKi$
i
r
nonce ni
$\{i, ni\}_{PKr}$
nonce nr
$\{ni, nr\}_{PKi}$
$\{nr\}_{PKr}$
i and r are communicating
i and r share secrets ni and nr

Fig. 1. The Needham-Schroeder public key authentication protocol. (The full notation will be explained in Section 4).

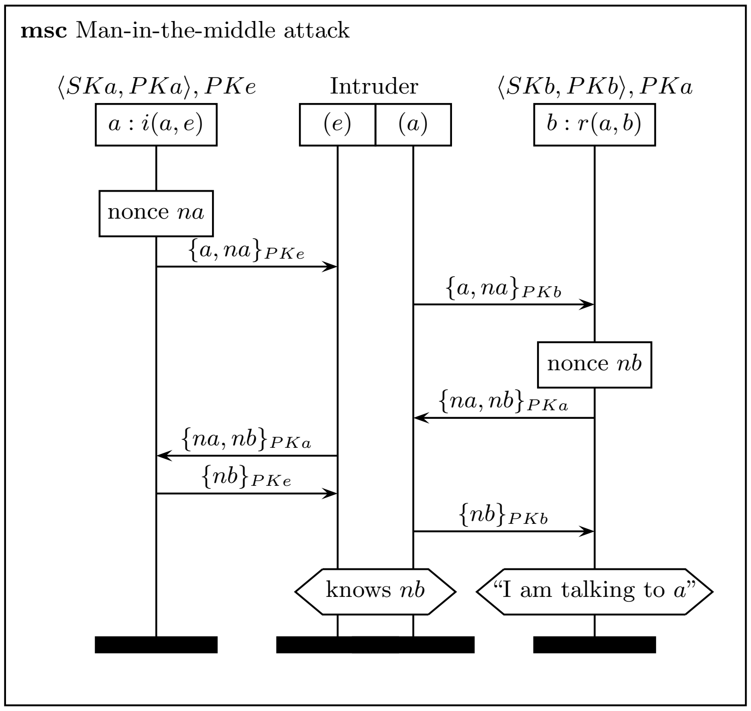

Fig. 2. Man-in-the-middle attack on the Needham-Schroeder protocol.

Acknowledgements. Thanks are due to Erik de Vink for his comments on our work and the stimulating discussions on security protocol semantics. Furthermore, we would like to thank Niek Palm for his study on the application of our semantics to the verification of a collection of security protocols.

2 Security protocols: a domain analysis

Rather than starting right away with the development of a formal semantics we first conduct a concise domain analysis. The purpose of this analysis is to make some of the design decisions explicit and to decompose the problem into smaller parts.

We start with the following informal description of involved concepts. A security protocol describes a number of behaviours. Each such behaviour we will call a role. We have, for instance, the *initiator* role and the *responder* role in a protocol. A system consists of a number of communicating agents. Each agent performs one or more roles (possibly from several security protocols). A role performed by an agent is called a run. For instance, agent a can perform two initiator runs and one responder run in parallel. The agents execute their runs to achieve some security goal (e.g. the confidential exchange of a message). While agents pursue their goals, an intruder may try to oppose them. The *capabilities* of the intruder determine its strength in attacking a protocol run. However, threats do not only come from the outside. Agents partaking in a protocol run may also conspire with the intruder and try to invalidate the security goals. In order to resist attacks, an agent can make use of *cryptographic primitives* when constructing messages.

Given this global description, we can identify the following components of the security protocol model.

Protocol specification
Agent model
Communication model
Threat model
Cryptographic primitives
Security requirements

We will discuss each of these sub-models, list their points of variation and make appropriate design decisions. Of course, every subdivision of the problem is artificial, but we found that this approach helped in adding structure and restricting the problem space. The sub models mentioned are not independent entities. For instance, the protocol specification makes use of the provided cryptographic primitives and the communication model is connected to the intruder model if the intruder has complete control over the network.

Protocol specification. The protocol specification describes the behaviour of each of the roles in the protocol. We consider this as a parameter of our semantics. We define an (abstract) syntax to specify a security protocol. Most often, a role in a security protocol is specified as a sequential list of events. In practise, a security enhanced communication protocol requires a more expressive specification language, but for an abstract description of e.g. an authentication protocol a sequential list will suffice. The set of events usually contains send and read events. Furthermore, we will consider security claims as special events. Timers (and all time related information) are not included in our model. A protocol specification is not complete without a specification of the initial knowledge required to execute a role and the declaration of functions, constants and variables occurring in the protocol specification. The protocol specification is expressed in a formal language for which we will define an abstract syntax and static requirements.

Agent model. Agents execute the roles of the protocol. The agent model is based on a *closed world assumption.* By this we mean that honest agents show no behaviour other than the behaviour described in the protocol specification. Thus, unless specified explicitly in the protocol, an honest agent will never leak classified information. The closed world assumption does not imply that an agent will only execute one run of the protocol. We assume that an agent may execute any number of runs in parallel (in an interleaved manner). Although restrictions on the number (and type) of runs may be of interest in practical applications, we will not parameterise over this property. The agent model also describes how an agent interprets a role description. An agent executes its role description sequentially, waiting at read events until an expected input message becomes available. This implies that an agent ignores unanticipated messages. More specifically, an incoming message will be matched against the expected message format as described in the protocol specification. Our semantics will be parameterized over this matching function, e.g. to allow for detection of type-flaw attacks.

Communication model. The communication model describes how the messages between the agents are exchanged. We have chosen the model of asynchronous communication. This model is more general than the synchronous communication model. Thus, if a security protocol is proven correct in the asynchronous model it will also be correct in the synchronous model. Assuming asynchronous communication, the next step is to select the type of buffering. Again, we will choose the most general model: one single multiset buffer for all agents.

Threat model. In 1983 Dolev and Yao led the basis for a network threat model that is currently the most widely used model [5]. In the Dolev-Yao model the intruder has complete control over the communication network. The intruder can intercept any message and insert any message, as long as he is able to construct its contents from his knowledge. Conspiring agents are modeled by including their initial knowledge in the knowledge of the intruder. Intruder models that are weaker than the Dolev-Yao model are also of interest, for instance when studying protocol stacks or special communication media. Wireless communication, for instance, implies that an intruder has the choice of jamming or eavesdropping, but not both for the same message. Therefore, we will consider the intruder model as a parameter of our semantics.

Cryptographic primitives. Cryptographic primitives are (idealized) mathematical constructs such as encryption. In our treatment of cryptographic primitives we use the so-called *black box approach.* This means that we do not exactly know which mathematical objects are used to implement such constructs, but that we only know their relevant properties. We will only consider symmetric and asymmetric encryption and discard other primitives, such as signing. The *perfect cryptography assumption* roughly states that nothing can be learned of a plain text from its encrypted version, without knowing the decryption key.

Security requirements. Security requirements state the purpose of a security protocol. They are mostly expressed as safety properties (i.e. something bad will never happen). In our semantics we will only study secrecy and two forms of authentication. However, the semantics is set up in such a way that other trace-based security properties are evenly expressible.

In the next section, we will make the above models precise.

3 Formal semantics

In this section we will use the domain analysis as a starting point for the development of an operational semantics. First, in Section 3.1, we define the security protocol level which specifies the roles of a protocol. The cryptographic primitives are also treated here. Next, in Section 3.2, the abstract syntax is provided with a static semantics. The roles only define behaviour schemes, which are instantiated into runs in Section 3.3. This section also contains the agent model by describing the operational rules which define the behaviour of a network of agents. The threat model is described in Sections 3.4 and 3.5. The latter contain some examples of intruder capabilities. Finally, in Section 3.6, we define secrecy and synchronisation, which is a strong authentication property.

3.1 Security protocol specification

A protocol specification defines the exchange of message terms between agents. We start by explaining a number of basic elements of these terms, such as constants, roles and variables. Next, we add constructors for pairing and tupling to construct the set $\mathit{RoleTerm}$, that will be used in role descriptions.

Basic sets. We start off with the following sets: $\mathcal{V}$ (denoting variables), $\mathcal{C}$ (denoting constants which are local to each instantiation of a role), $\mathcal{R}$ (denoting roles), and $\mathcal{F}$ (denoting function names). Functions from the set $\mathcal{F}$ are considered to be global, and have an arity which must be respected in all terms. If global constants occur in a protocol, we model them as functions of arity zero. In Table 1 we show some typical elements of these sets, as used throughout this paper.

Terms. We introduce constructors for pairing and encryption, and we assume that pairing is associative.

$$\begin{aligned} \mathit{RoleTerm} ::= {} & \mathcal{V} \mid \mathcal{R} \mid \mathcal{F}(\mathit{RoleTerm}^*) \mid \mathcal{C} \mid \\ & (\mathit{RoleTerm}, \mathit{RoleTerm}) \mid \{\mathit{RoleTerm}\}_{\mathit{RoleTerm}} \end{aligned}$$

Description	Set	Typical elements
Variables	$\mathcal{V}$	V, W, X, Y, Z
Constants	$\mathcal{C}$	$ni, nr, sessionkey$
Roles	$\mathcal{R}$	i, r, s
Functions	$\mathcal{F}$	$sk, pk, k,$ hash
Trusted agents	$\mathcal{A}_T$	a, b, c
Untrusted agents	$\mathcal{A}_U$	e

Table 1. Basic sets and some typical elements

Terms that have been encrypted with a term, can only be decrypted by either the same term (for symmetric encryption) or the inverse key (for asymmetric encryption). To determine which term needs to be known to decrypt a term, we introduce a function that yields the inverse for any role term.

$$\_^{-1} : \mathit{RoleTerm} \rightarrow \mathit{RoleTerm}$$

We require that $\_^{-1}$ is its own inverse, i.e. $(t^{-1})^{-1} = t$. Terms are reduced according to $\{\{s\}_t\}_{t^{-1}} = s$.

Throughout this article we will assume that pk and sk are functions of arity 1, that map to asymmetric keys, such that $\forall_{r \in \mathcal{R}} pk(r)^{-1} = sk(r)$ and vice versa. All other terms t are considered to be symmetric keys, for which we have $t^{-1} = t$.

Role Knowledge. Besides terms to be sent and received, a role specification describes the initial knowledge needed to execute the role We define a role knowledge set as $\mathit{RoleKnow} = \mathcal{P}(\mathit{RoleTerm})$.

Role specification. We define a role specification as a set of initial knowledge, and a list of events. We define the set of events $\mathcal{E}$ using two new sets: labels $\mathcal{L}$ and security claims Claim, which we explain below.

$$\begin{aligned}\mathcal{E} = \big\{ & \mathit{send}_\ell(r, r', t), \mathit{read}_\ell(r', r, t), \mathit{claim}_\ell(r, c\,[, t]) \;\big|\; \\ & \ell \in \mathcal{L}, r, r' \in \mathcal{R}, t \in \mathit{RoleTerm}, c \in \mathit{Claim} \big\}\end{aligned}$$

Event $\mathit{send}_\ell(r, r', t)$ denotes the sending of message t by r, apparently to r'. Likewise, $\mathit{read}_\ell(r', r, t)$ denotes the reception of message t by r', apparently sent by r. Event $\mathit{claim}_\ell(r, c\,[, t])$ expresses that r upon execution of this event expects security goal c to hold with optional parameter t. A claim event denotes a *local claim*, which means that it only concerns role r and does not express any expectations at other roles.

The labels ℓ extending the events are needed to disambiguate similar occurrences of the same event in a protocol specification. A second use of these labels will be to express the relation between corresponding send and read events, as we will see in Section 3.6.

Now we can specify a role. A role specification consists of a list of events, and some initial knowledge: $\mathit{RoleSpec} = \mathit{RoleKnow} \times \mathcal{E}^*$.

Protocol specification. A protocol specifies the behaviour for a number of roles by means of a partial function from the set $\mathit{ProtSpec} = \mathcal{R} \rightarrow \mathit{RoleSpec}$.

We will use $\mathit{MR}^p(r)$ as a shorthand for the initial knowledge of role r in a protocol specification p. In many cases we omit the parameter p if the intented protocol is clear from the context.

Example. The following role description models the initiator role of the Needham-Schroeder protocol, without any security requirements.

$$\begin{aligned} ns(i) = \big(&\{i, r, ni, sk(i), pk(i), pk(r)\},\\ &send_1(i, r, \{i, ni\}_{pk(r)}) \cdot\\ &read_2(r, i, \{ni, V\}_{pk(i)}) \cdot\\ &send_3(i, r, \{V\}_{pk(r)})\big) \end{aligned}$$

This role description follows from Figure 1 by selecting the left-most axis and its associated events. Notice that we have to clarify which constructs in the terms are variables (because they receive their value at reception of a message) and which are constants (because they are determined by the role itself). Therefore, we define $i, r \in \mathcal{R}$, $ni \in \mathcal{C}$, $sk, pk \in \mathcal{F}$, $pk(i)^{-1} = sk(i)$, $pk(r)^{-1} = sk(r)$, $1, 2, 3 \in \mathcal{L}$, and $V \in \mathcal{V}$.

3.2 Static Requirements

In the previous section we have explained the context-free abstract syntax for a protocol specification. A proper protocol specification will also have to satisfy a number of well-formedness rules.

Well-Formed Roles. For each role, we require that it meets certain criteria. These range from the fairly obvious, e.g. each event in a role definition has the same actor, to more subtle requirements regarding the messages. For the messages we require that the messages that are sent can actually be constructed by the sender. This is satisfied if the message is in the knowledge of the sending role. For variables we require that they first occur in a read event, where they are instantiated, before they can occur in a send event.

For read events the situation is a bit more complex. As can be seen in the example above, which describes the initiator role of the Needham-Schroeder protocol, a read event may impose some structure upon the incoming messages. A receiver can only match a message against such an expected pattern if his knowledge satisfies certain requirements.

We introduce a predicate *WF* (Well Formed) to express that a role definition meets these consistency requirements, using an auxiliary predicate *Readable* and a knowledge inference operator $\_ \vdash \_ : RoleKnow \times RoleTerm$.

Agents can compose and decompose pair terms. A term can be encrypted if the agent knows the encryption key, and an encrypted term can be decrypted if the agent knows the corresponding decryption key. This is expressed by the knowledge inference operator, which is defined inductively as follows.

$$\begin{aligned} t \in M &\Longrightarrow M \vdash t\\ M \vdash t_1 \wedge M \vdash t_2 &\Longleftrightarrow M \vdash (t_1, t_2)\\ M \vdash t \wedge M \vdash k &\Longrightarrow M \vdash \{t\}_k\\ M \vdash \{t\}_k \wedge M \vdash k^{-1} &\Longrightarrow M \vdash t \end{aligned}$$

Composing terms t_1, t_2 into a term t by encryption or tupling implies that t has t, t_1 and t_2 as subterms. The subterm operator $\sqsubseteq$ is inductively defined as follows.

$$\begin{array}{lll} t \sqsubseteq t & t_1 \sqsubseteq (t_1, t_2) & t_1, \ldots, t_n \sqsubseteq f(t_1, \ldots, t_n) \\ t \sqsubseteq \{t\}_k & t_2 \sqsubseteq (t_1, t_2) & \end{array}$$

The predicate *Readable* : *RoleKnow* $\times$ *RoleTerm* expresses which role terms can be used as a message pattern for a read event of an agent with a specific knowledge set. A variable can always occur in a read pattern. Any other term can only occur in a read pattern, if it can be inferred from the knowledge of the agent. Only then can it be compared to the incoming messages.

In order to be able to read a pair, we must be able to read each constituent, while extending the knowledge with what can be inferred from the other component. An encrypted message can be read if it can be inferred from the knowledge or if it can be inferred after decryption, which requires that the decryption key is in the knowledge.

$$\mathit{Readable}(M, t) = \begin{cases} \text{True} & \text{if } t \in \mathcal{V} \\ M \vdash t & \text{if } t \in \mathcal{C} \cup \mathcal{R} \cup \mathcal{F}(\mathit{RoleTerm}^*) \\ \mathit{Readable}(M \cup \{t_2\}, t_1) \wedge \mathit{Readable}(M \cup \{t_1\}, t_2) & \text{if } t \equiv (t_1, t_2) \\ (M \vdash \{t_1\}_{t_2}) \vee (M \vdash {t_2}^{-1} \wedge \mathit{Readable}(M, t_1)) & \text{if } t \equiv \{t_1\}_{t_2} \end{cases}$$

We can now construct the predicate *WF* : $\mathcal{R} \times$ *RoleSpec*, that expresses that a role is well formed. The first argument of this predicate is used to express that the active role in an event should match the role which behaviour is being defined. Terms occurring in a send or claim event must be inferable from the knowledge, while terms occurring in a read event must be readable according to the definition above.

$$\mathit{WF}(r, (M, s)) = \begin{cases} \text{True} & \text{if } s \equiv \varepsilon \\ M \vdash (r', r) \wedge \mathit{Readable}(M, t) \wedge \mathit{WF}(r, (M \cup \{t\}, s')) & \text{if } s \equiv \mathit{read}_\ell(r', r, t) \cdot s' \\ M \vdash (r, r', t) \wedge \mathit{WF}(r, (M, s')) & \text{if } s \equiv \mathit{send}_\ell(r, r', t) \cdot s' \\ M \vdash (r\ [, t]) \wedge \mathit{WF}(r, (M, s')) & \text{if } s \equiv \mathit{claim}_\ell(r, c\ [, t]) \cdot s' \\ \text{False} & \text{otherwise} \end{cases}$$

For a protocol specification p we require that all roles are well formed with respect to their initial knowledge, which is expressed by: $\forall_{r \in dom(p)}\, \mathit{WF}(r, p(r))$.

Examples. The next two examples are incorrect role descriptions:

$$\begin{array}{ll} \mathit{wrong1}(i) = (\{i, r, k\}, & \mathit{wrong2}(i) = (\{i, r, k\}, \\ \quad \mathit{send}_1(i, r, \{i, r, V\}_k) \cdot & \quad \mathit{read}_1(r, i, \{i, r, \{V\}_{k2}\}_k) \cdot \\ \quad \mathit{read}_2(r, i, \{V, r\}_k)\ ) & \quad \mathit{send}_2(i, r, \{V\}_{k2})\ ) \end{array}$$

Role description *wrong1* is not well formed because it sends variable V before it is read. The read event in *wrong2* contains a subterm $\{V\}_{k2}$. The intention is that V is initialised through this read. However, since $k2$ is a symmetric key, and $k2$ is not in the knowledge of the role, the value of V cannot be determined through this read. Therefore, this role description is not well formed. The correct role description would be the following:

$$\begin{aligned} \mathit{wrong2corrected}(i) = (&\{i, r, k\}, \\ &\mathit{read}_1(r, i, \{i, r, W\}_k)\cdot \\ &\mathit{send}_2(i, r, W)\ ) \end{aligned}$$

3.3 Runs

The protocol specification describes a set of roles. These roles serve as a blueprint for what the actual agents in a system should do. A run is defined as an instantiated role. In order to instantiate a role we have to bind the role names to the names of actual agents and we have to make the local constants unique for each instantiation. Furthermore, we have to take into account that the bindings of values to the variables are local to a run too. Thus, the set of terms occurring in a run differs from the set of terms used in role descriptions.

Run terms. We assume existence of a set *Runid* to denote run identifiers and a set $\mathcal{A}$ to denote agents. The set $\mathcal{A}$ is partitioned into sets $\mathcal{A}_T$ (denoting the *trusted agents*) and $\mathcal{A}_U$ (denoting the *untrusted agents*). Run terms are defined similarly to role terms. The difference is that abstract roles are replaced by concrete agents, that local constants are made unique by extending them with a run identifier, and that variables are instantiated by concrete values. The run term set also includes the set $\mathcal{C}^I$ of terms constructed by an intruder. This set will only be used from Section 3.4 onwards, and it will be explained there. As for role terms, we have associativity of pairing.

$$\begin{aligned} \mathit{RunTerm} ::= \mathcal{A} \mid \mathcal{F}(\mathit{RunTerm}^*) \mid \mathcal{C}\sharp\mathit{Runid} \mid \mathcal{C}^I \mid \\ (\mathit{RunTerm}, \mathit{RunTerm}) \mid \{\mathit{RunTerm}\}_{\mathit{RunTerm}} \end{aligned}$$

Instantiation. A role term is transformed into a run term by applying an instantiation.

$$\mathit{Inst} = \mathit{Runid} \times (\mathcal{R} \to \mathcal{A}) \times (\mathcal{V} \to \mathit{RunTerm})$$

The first component of an instantiation determines with which run identifier the constants are extended. The second component determines the instantiation of roles by agents. The third determines the valuation of the variables.

We extend the inverse function to *RunTerm*. The functions $\mathit{roles} : \mathit{RoleTerm} \to \mathcal{P}(\mathcal{R})$ and $\mathit{vars} : \mathit{RoleTerm} \to \mathcal{P}(\mathcal{V})$ determine the roles and variables occurring in a term. We extend these functions to the domain of *RoleSpec* in the obvious way.

For instantiation $(rid, \rho, \sigma) \in \mathit{Inst}$, $f \in \mathcal{F}$ and terms $t, t_1, \ldots, t_n \in \mathit{RoleTerm}$ such that $\mathit{roles}(t) \subseteq \mathit{dom}(\rho)$ and $\mathit{vars}(t) \subseteq \mathit{dom}(\sigma)$, we define instantiation by:

$$(rid, \rho, \sigma)(t) = \begin{cases} \rho(r) & \text{if } t \equiv r \in \mathcal{R} \\ f((rid, \rho, \sigma)(t_1), \ldots, (rid, \rho, \sigma)(t_n)) & \text{if } t \equiv f(t_1, \ldots, t_n) \\ c\sharp rid & \text{if } t \equiv c \in \mathcal{C} \\ \sigma(v) & \text{if } t \equiv v \in \mathcal{V} \\ ((rid, \rho, \sigma)(t_1), (rid, \rho, \sigma)(t_2)) & \text{if } t \equiv (t_1, t_2) \\ \{(rid, \rho, \sigma)(t_1)\}_{(rid, \rho, \sigma)(t_2)} & \text{if } t \equiv \{t_1\}_{t_2} \end{cases}$$

Example. We give two examples of instantiations that might occur in the execution of a protocol:

$$\begin{aligned} \big(1, \{i \mapsto a, r \mapsto b\}, \emptyset\big)\ \big(\{i, ni\}_{pk(r)}\big) &= \{a, ni\sharp 1\}_{pk(b)} \\ \big(2, \{i \mapsto c, r \mapsto d\}, \{W \mapsto ni\sharp 1\}\big)\ \big(\{W, nr, r\}_{pk(i)}\big) &= \{ni\sharp 1, nr\sharp 2, d\}_{pk(c)} \end{aligned}$$

Runs. A run is an instantiated role specification. As the knowledge of a role is already statically defined by the role description, we can omit it from the run specification and define $\mathit{Run} = \mathit{Inst} \times \mathcal{E}^*$. As we will see later on, each run in the system will have a unique run identifier by construction.

State. The system that we consider consists of a number of runs executed by some agents. Communication between the runs is asynchronous (buffered). In order to conveniently model the intruder behaviour, we will route communication through two buffers: one output buffer from the sending run and one input buffer from the receiving run (for a discussion on the expressive power of such construction, see [6]). The intruder capabilities will determine how the messages are transferred from the output buffer to the input buffer.

Both the output buffer and the input buffer store sent messages. Messages contain a sender, a recipient, and a run term: $\mathit{MSG} = \mathcal{A} \times \mathcal{A} \times \mathit{RunTerm}$. Notice that, if we identify set product with pairing, we obtain $\mathit{MSG} \subset \mathit{RunTerm}$. A buffer is a multiset of such messages: $\mathit{Buffer} = \mathcal{M}(\mathit{MSG})$.

Since the knowledge of the intruder is dynamic, we will consider this a component in the state of the system, too. It consists of instantiated terms as they occur in the runs, and is represented by $\mathit{RunKnow} = \mathcal{P}(\mathit{RunTerm})$.

The state of a network of agents executing roles in a security protocol is defined by

$$\mathit{State} = \mathit{RunKnow} \times \mathit{Buffer} \times \mathit{Buffer} \times \mathcal{P}(\mathit{Run}),$$

and thus contains the intruder knowledge, the contents of the output buffer, the contents of the input buffer, and the (remainders of the) runs that still have to be executed.

Match. Messages from the buffer are accepted by agents if they match a certain pattern, specified in the read event. We introduce a predicate *Match* that expresses that a message matches the pattern for some instantiation of the variables. The definition of this predicate is a parameter of our system, but we will give an example of a straightforward typed match.

For each variable, we define a set of run terms which are allowed values. We introduce an auxiliary function $\mathit{type} : \mathcal{V} \to \mathcal{P}(\mathit{RunTerm})$, that defines the set of run terms that are valid values for a variable. Then we define the predicate *Welltyped* on $(\mathcal{V} \to \mathcal{P}(\mathit{RunTerm}))$, that expresses that a substitution is well-typed: $\mathit{Welltyped}(\sigma) = \forall_{v \in \mathit{dom}(\sigma)} \big(\sigma(v) \in \mathit{type}(v)\big)$.

Using this predicate, we define the typed matching predicate $\mathit{Match} : \mathit{Inst} \times \mathit{RoleTerm} \times \mathit{RunTerm} \times \mathit{Inst}$. The purpose of this predicate is to match an incoming message (the third argument) to a pattern specified by a role term (the second argument). This pattern is already instantiated (the first argument), but may still contain free variables. The idea is to assign values to the free variables such that the incoming message equals the instantiated role term. The old instantiation extended with these new assignments provides the resulting instantiation (the fourth argument).

$$\begin{aligned}\mathit{Match}(\mathit{inst}, \mathit{pt}, m, \mathit{inst}') \iff\ & \mathit{inst} = (\mathit{rid}, \rho, \sigma) \wedge \mathit{inst}' = (\mathit{rid}, \rho, \sigma') \wedge \\ & \sigma \subseteq \sigma' \wedge \mathit{dom}(\sigma') = \mathit{dom}(\sigma) \cup \mathit{vars}(\mathit{pt}) \wedge \\ & \mathit{Welltyped}(\sigma') \wedge (\mathit{rid}, \rho, \sigma')(\mathit{pt}) = m\end{aligned}$$

Assume $\rho = \{i \mapsto a, r \mapsto b\}$. Then, some examples for which the predicate is true are:

	inst	*pt*	*m*	*inst′*		
Match(	$(1, \rho, \emptyset)$,	X,	$nr\sharp 2$,	$(1, \rho, \{X \mapsto nr\sharp 2\})$	)	$\iff$ *True*
Match(	$(1, \rho, \emptyset)$,	$\{r, ni\}_{pk(i)}$,	$\{b, ni\sharp 1\}_{pk(a)}$,	$(1, \rho, \emptyset)$	)	$\iff$ *True*

Some examples where the predicate does not hold, if we assume matching is typed, and the type of X is the set $\mathcal{A} \cup \mathcal{C}\sharp\mathit{Runid} \cup \mathcal{C}^I$

	inst	*pt*	*m*	*inst′*		
Match(	$(1, \rho, \emptyset)$,	nr,	$nr\sharp 2$,	_	)	$\iff$ *False*
Match(	$(1, \rho, \emptyset)$,	X,	$(nr\sharp 2, ni\sharp 1)$,	_	)	$\iff$ *False*
Match(	$(1, \rho, \emptyset)$,	$\{i, ni\}_{pk(i)}$,	$\{b, ni\sharp 1\}_{pk(a)}$,	_	)	$\iff$ *False*

By varying over the function *type* we can express whether the protocol is vulnerable to type flaw attacks or not. This also allows for expressing that only basic type flaws can be detected by the agents.

Derivation rules. The behaviour of the system is defined as a transition relation (see [7]) between system states. A transition is labeled with an element of the set $\mathit{Transitionlabel} ::= (\mathit{Inst}, \mathcal{E}) \mid \mathit{create}(\mathit{Run}) \mid \mathit{Networkrules}(\mathit{MSG})$. The set of network/intruder rules *Networkrules* is a parameter of the system, and we will discuss some of the possibilities in Section 3.5.

A protocol description allows for the creation of runs. The runs that can be created are defined by the function $\mathit{runsof} : \mathit{ProtSpec} \to \mathcal{P}(\mathit{Run})$:

$$\mathit{runsof}(p) = \\ \Big\{\big((\mathit{rid}, \rho, \emptyset), p(r)\big) \ \Big|\ r \in \mathit{dom}(p) \wedge \mathit{rid} \in \mathit{Runid} \wedge \rho \in \mathit{roles}(p(r)) \times \mathcal{A}\Big\}$$

For $F \in \mathcal{P}(Run)$ we use $F[r'/r]$ to denote the substitution of r by r' in F. We define the set of active run identifiers as

$$runids(F) = \left\{ rid \;\middle|\; \big((rid, \rho, \sigma), ev\big) \in F \right\}$$

Let $p \in ProtSpec$. Then the basic derivation rules for the system are given in Table 2. The *create* rule expresses that a new run can only be created if its run identifier has not been used yet. The *send* rule states that if a run executes a send event, the sent message is added to the output buffer and the executing run proceeds to the next event. The *read* rule determines when an input event can be executed. It requires that the (partially) instantiated pattern specified in the read event should match any of the messages from the input buffer. Upon execution of the read event, this message is removed from the input buffer and the executing run advances to the next event. The *claim* rule expresses that an enabled claim event can always be executed. Notice that in all these cases the intruder knowledge is not affected. The dynamical behaviour of the intruder knowledge will be defined by the network/intruder rules in Section 3.5.

A state transition is the conclusion of finitely many applications of these rules. In this way, starting from the initial state, we can derive all possible behaviours of a system executing security protocol p. This is what we consider the operational semantics of p.

$$[create]\,\frac{run = ((rid, \rho, \sigma), s) \in runsof(p), rid \notin runids(F)}{\langle M, BS, BR, F\rangle \xrightarrow{create(run)} \langle M, BS, BR, F \cup \{run\}\rangle}$$

$$[send]\,\frac{run = (inst, send_\ell(m) \cdot s) \in F}{\langle M, BS, BR, F\rangle \xrightarrow{(inst, send_\ell(m))} \langle M, BS \cup \{inst(m)\}, BR, F[(inst, s)/run]\rangle}$$

$$[read]\,\frac{run = (inst, read_\ell(pt) \cdot s) \in F, m \in BR, Match(inst, pt, m, inst')}{\langle M, BS, BR, F\rangle \xrightarrow{(inst', read_\ell(pt))} \langle M, BS, BR \setminus \{m\}, F[(inst', s)/run]\rangle}$$

$$[claim]\,\frac{run = (inst, claim_\ell(r, c\,[,t]) \cdot s) \in F}{\langle M, BS, BR, F\rangle \xrightarrow{(inst, claim_\ell(r, c\,[,t]))} \langle M, BS, BR, F[(inst, s)/run]\rangle}$$

Table 2. SOS rules.

Initial state. In the initial state of the system both buffers are empty, and no runs have been created yet. Thus the initial state of the system is given by

$$s_0 = \langle M_0, \emptyset, \emptyset, \emptyset \rangle$$

where M_0 refers to the intruder knowledge, which we define in the next section.

3.4 Initial intruder knowledge

We assume the intruder can create a possibly infinite number of constants, defined as the set $\mathcal{C}^I$. The initial knowledge of the intruder includes this set. We model untrusted agents by including their initial knowledge in the initial intruder knowledge.

We could choose to define the initial knowledge of the intruder as the static knowledge of all the roles, for all untrusted agents. However, for some protocols we require that the untrusted agents cannot play certain roles. It is e.g. undesirable that an untrusted agent plays the role of the certificate server that knows the secret keys of all the agents. We define these roles as the set of trusted roles $\mathcal{R}_T$. All other roles are called the untrusted roles $\mathcal{R}_U$. Unless stated otherwise, we assume $\mathcal{R}_T = \emptyset$, and thus $\mathcal{R}_U = \mathcal{R}$.

The intruder learns all initial knowledge of a role before it is instantiated in a specific run. Thus, This excludes any local constants, as well variable names (because they are not instantiated yet). The initial intruder knowledge will consist of e.g. the names and public keys of all agents, and the secret keys of the intruder. The following table shows some examples for the knowledge of a role $i \in \mathcal{R}$.

$$\begin{aligned} \{i\} \subseteq MR(i) &\Rightarrow \mathcal{A}_U \subseteq M_0 \\ \{r\} \subseteq MR(i) &\Rightarrow \mathcal{A} \subseteq M_0 \\ \{pk(r), sk(i)\} \subseteq MR(i) &\Rightarrow \{pk(a), sk(e) \mid a \in \mathcal{A} \wedge e \in \mathcal{A}_U\} \subseteq M_0 \end{aligned}$$

If the i role knowledge contains e.g. $sk(i), pk(r)$, we see that the intruder knowledge contains $sk(e)$ for each untrusted agent e acting in this role. Untrusted agents are however able to communicate with trusted agents, and thus $pk(a)$ is in the initial intruder knowledge for each agent a.

To instantiate the role knowledge, we only need to know how the role names are mapped to agent names: information about a run or instantiation of the variables is not needed. For a protocol p, an untrusted agent e in an untrusted role r, the knowledge that is passed to the intruder is defined as

$$\bigcup_{\substack{\rho \in \mathcal{R} \to \mathcal{A} \\ \rho(r) = e}} \left\{ (\_, \rho, \_)t \;\middle|\; t \in MR(r) \wedge \forall_{t' \sqsubseteq t}(t' \notin \mathcal{V} \cup \mathcal{C}) \right\}$$

For a protocol p, we define the initial intruder knowledge as the union of this knowledge of all untrusted agents and roles:

$$M_0 = \mathcal{C}^I \cup \bigcup_{\substack{\rho \in \mathcal{R} \rightarrow \mathcal{A} \\ r \in \mathcal{R}_U \\ \rho(r) \in \mathcal{A}_U}} \left\{ (\_, \rho, \_)t \;\middle|\; t \in MR(r) \wedge \forall_{t' \sqsubseteq t} (t' \notin \mathcal{V} \cup \mathcal{C}) \right\}$$

For example, for the Needham-Schroeder protocol, the initial intruder knowledge would simply consist of the set $\mathcal{C}^I$, the names and public keys of all agents, and the secret keys of the untrusted agents.

3.5 Network/Intruder rules

In the context of security protocol verification the Dolev-Yao intruder model is commonplace. In this model, the intruder has complete control over the network. Messages can be learnt, deflected, and created by such an intruder. However, often this intruder model is too powerful, for example when an intruder can only eavesdrop on the network, or in wireless communications. In such cases, it might be desirable to develop more lightweight protocols that are correct for this weaker intruder model. Therefore, we parameterise over the intruder model, which is defined as a set of capabilities. Each intruder rule defines a capability by explaining the effect of the intruder action on the output buffer, the input buffer and the intruder knowledge. In Table 3 we give some examples of intruder rules. The *transmit* rule describes transmission of a message from the output buffer to the input buffer without interference from the intruder. If the intruder has eavesdropping capabilities, as stated in the *eavesdrop* rule he can learn the message during transmission. The *deflect* rule states that an intruder with deflection capabilities can delete any message from the output buffer. The difference witht the *jam* rule is that the intruder can read the deflected message and add it to its knowledge. The *inject* rule describes the injection of any message inferable from the intruder knowledge into the input buffer.

Next, we define some interesting intruders. In a network without an intruder we only have the *transmit* rule, so $\mathit{NoIntruder} = \{\mathit{transmit}\}$. In the Dolev-Yao model the intruder has full control over the network. Every message is read and analysed, and anything that can be constructed can be inserted into the network, so $\mathit{DolevYao} = \{\mathit{deflect}, \mathit{inject}\}$. A wireless communication network is weaker than Dolev-Yao, because it does not allow learning from a message and blocking it at the same time. Thus we define $\mathit{Wireless} = \{\mathit{eavesdrop}, \mathit{jam}, \mathit{inject}\}$. If the intruder can only eavesdrop, we have $\mathit{ReadOnly} = \{\mathit{eavesdrop}\}$.

It is possible to construct more intruder rules, for intruder capabilities such as rerouting of messages or the modification of messages.

3.6 Security properties

Traces. We will discuss some trace based security properties, therefore, we define the traces generated by the above derivation rules. For $\alpha = \alpha_0 \ldots \alpha_{n-1} \in$

$$[transmit]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{transmit(m)} \langle M, BS \setminus \{m\}, BR \cup \{m\}, F\rangle}$$

$$[deflect]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{deflect(m)} \langle M \cup \{m\}, BS \setminus \{m\}, BR, F\rangle}$$

$$[inject]\frac{M \vdash m}{\langle M, BS, BR, F\rangle \xrightarrow{inject(m)} \langle M, BS, BR \cup \{m\}, F\rangle}$$

$$[eavesdrop]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{eavesdrop(m)} \langle M \cup \{m\}, BS \setminus \{m\}, BR \cup \{m\}, F\rangle}$$

$$[jam]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{jam(m)} \langle M, BS \setminus \{m\}, BR, F\rangle}$$

Table 3. Network/intruder rules.

$Transitionlabel^*$ we use $s_0 \overset{\alpha}{\rightarrow} s_n$ to denote $\exists_{s_1,\ldots s_{n-1}} s_0 \overset{\alpha_0}{\rightarrow} s_1 \ldots s_{n-1} \overset{\alpha_{n-1}}{\rightarrow} s_n$. We use $s \overset{\alpha}{\rightarrow}$ to denote $\exists_{s'} s \overset{\alpha}{\rightarrow} s'$. The length of a sequence of labels α is denoted by $\mid \alpha \mid$.

The set of traces $Tr : ProtSpec \rightarrow \mathcal{P}(Transitionlabel^*)$ is defined as $\{a \in Transitionlabel^* \mid s_0 \overset{a}{\rightarrow}\}$, where s_0 is the initial state of the protocol. For trace α, we use α_i to denote the i^{th} action label from α.

We reconstruct state information from a trace as follows. If α_i is an action from trace α, then M_i^α (or simply M_i) is the intruder knowledge right before the execution of α_i.

Secrecy. For $t \in RoleTerm$, we introduce the claim $claim_\ell(r, secret, t)$.

A protocol p is correct with respect to secrecy if the following holds for all traces $\alpha \in Tr(p)$ and $i \in N$.

$$\alpha_i = ((rid, \rho, \sigma), claim_\ell(r, secret, t)) \ \wedge \ rng(\rho) \subseteq \mathcal{A}_T \Rightarrow \\ \forall_{i \le j \le |\alpha|} (rid, \rho, \sigma)(t) \notin M_j^\alpha$$

Synchronisation. We define a strong authentication requirement called synchronisation. A thorough description of this form of authentication can be found in [8]. A synchronisation claim boils down to the requirement that the corresponding sends and reads of two communicating runs exactly match each other. This property resembles the notion of intensional specifications [9] and is stronger than the well-known agreement property, which can also be described in our framework.

Synchronisation is defined with help of some auxiliary functions and predicates. The first predicate expresses that for label ℓ two runs agree on the occurrences of the $send_\ell$ event and the $read_\ell$ event. We use the function $sendrole(\ell)$ to denote the role in which the event $send_\ell$ occurs. The function $readrole(\ell)$ is defined likewise.

We define the projection function $runidof : Inst \to Runid$ by $runidof(rid, \rho, \sigma) = rid$. For all traces α, $k \in N$, labels ℓ and run identifiers rid_1, rid_2, the single-label synchronisation predicate *1L-SYNCH* is given by

$$\begin{aligned}
&\mathit{1L\text{-}SYNCH}(\alpha, k, \ell, rid_1, rid_2) \iff \\
&\quad \exists_{i,j \in N, inst_1, inst_2 \in Inst, m_1, m_2 \in MSG} \\
&\qquad i < j < k \wedge \\
&\qquad \alpha_i = (inst_1, send_\ell(m_1)) \ \wedge \ runidof(inst_1) = rid_1 \wedge \\
&\qquad \alpha_j = (inst_2, read_\ell(m_2)) \ \wedge \ runidof(inst_2) = rid_2 \wedge \\
&\qquad inst_1(m_1) = inst_2(m_2)
\end{aligned}$$

This predicate is generalised to sets of labels in the following way. For all traces α, $k \in N$, label set L, and $cast : \mathcal{R} \to Runid$, the multi-label synchronisation predicate *ML-SYNCH* is given by

$$\begin{aligned}
&\mathit{ML\text{-}SYNCH}(\alpha, k, L, cast) \iff \\
&\quad \forall_{\ell \in L} \ \mathit{1L\text{-}SYNCH}\big(\alpha, k, \ell, cast(sendrole(\ell)), cast(readrole(\ell))\big)
\end{aligned}$$

If $\mathit{ML\text{-}SYNCH}(\alpha, k, L, cast)$ holds, we say that the set of labels L has correctly occurred in a trace α before position k with respect to the instantiation $cast$.

In order to determine the relevant set of labels which should be checked if a synchronisation claim occurs, we define the set $prec(p, cl)$. This set contains the causally preceding communications of a claim role event labeled with cl, for a security protocol p and is given by

$$prec(p, cl) = \{\ell \mid read_\ell(\_, \_, \_) \prec claim_{cl}(\_, \_)\}$$

We introduce the claim $nisynch \in Claim$. A protocol p is correct with respect to *NI-SYNCH* if the following holds for all traces $\alpha \in Tr(p)$.

$$\begin{aligned}
&\alpha_i = (rid, \rho, \sigma, claim_\ell(r, nisynch)) \wedge rng(\rho) \subseteq \mathcal{A}_T \\
&\qquad \Rightarrow \exists_{cast:\mathcal{R} \to Runid} \ (cast(r) = rid \wedge \mathit{ML\text{-}SYNCH}(\alpha, i, prec(p, \ell), cast))
\end{aligned}$$

4 The Needham-Schroeder(-Lowe) protocol

In this section we will take a closer look at the Needham-Schroeder protocol from Figure 1 and illustrate our definitions. The protocol goal is to ensure mutual authentication and as a side effect secrecy of the involved nonces. Starting point of the protocol is a public key infrastructure. This is depicted by the initial knowledge above each of the roles in the protocol. The initiator starts the protocol by sending an encrypted initialisation request to the responder. The nonce is used to prevent play-back attacks. Only the responder is able to unpack this message and replies by sending the initiator's nonce together with his own fresh nonce. Then the initiator proves his identity by replying the responder's nonce.

The man-in-the-middle attack in Figure 2 only requires two runs. One of trusted agent a performing the initiator role in a session with untrusted agent m and one of trusted agent b performing the responder role in a session with agent a. The intruder impersonates both m and a and in this way uses a as an oracle to unpack message from b. At the end he has fooled b into thinking that he is talking to a, while he is talking to the intruder.

Knowing this attack, it is straightforward to reconstruct it formally with our semantics. Our experience shows that when trying to prove a flawed protocol correct, the way in which the proof fails often indicates the attack. Rather than showing the details here, we will prove correctness of the fixed Needham-Schroeder protocol, which is called the Needham-Schroeder-Lowe protocol. The protocol is hardened by extending message 2 with the responder name. It is specified as follows.

$$\begin{array}{ll}
nsl(i) = (\{i, r, ni, pk(r), pk(i), sk(i)\}, & nsl(r) = (\{i, r, nr, pk(i), pk(r), sk(r)\}, \\
\quad send_1(i, r, \{i, ni\}_{pk(r)})\cdot & \quad read_1(i, r, \{i, W\}_{pk(r)})\cdot \\
\quad read_2(r, i, \{ni, V, r\}_{pk(i)})\cdot & \quad send_2(r, i, \{W, nr, r\}_{pk(i)})\cdot \\
\quad send_3(i, r, \{V\}_{pk(r)}))\cdot & \quad read_3(i, r, \{nr\}_{pk(r)}))\cdot \\
\quad claim_4(i, secret, ni)\cdot & \quad claim_7(r, secret, nr)\cdot \\
\quad claim_5(i, secret, V)\cdot & \quad claim_8(r, secret, W)\cdot \\
\quad claim_6(i, nisynch)\ ) & \quad claim_9(r, nisynch)\ )
\end{array}$$

We assume that there are no trusted roles. For this protocol, the initial intruder knowledge (cf. Section 3.4) is given by

$$M_0 = \mathcal{C}^I \cup \bigcup_{a \in \mathcal{A}} \{a, pk(a)\} \cup \bigcup_{e \in \mathcal{A}_U} \{sk(e)\}$$

First we introduce some notation and present results which support verification. We define $msgs(p)$ as the set of all role messages sent in the protocol. The first lemma helps to infer that secret information which is never transmitted, remains secret forever.

Lemma 1. *Let p be a protocol, i an instantiation and t a basic term. If t is not a subterm of any message that is ever sent, and $i(t)$ is not a subterm of the initial intruder knowledge, then $i(t)$ will never be known by the intruder. Formally:*

$$\forall_{t' \in msgs(p)} t \not\sqsubseteq t' \ \wedge \ \forall_{m : M_0 \vdash m} i(t) \not\sqsubseteq m \ \Rightarrow \ \forall_{\alpha \in Tr(p), 0 \leq j \leq |\alpha|} M_j^\alpha \nvdash i(t)$$

The correctness of this lemma follows from the SOS-rules.

The next lemma expresses that roles are executed from the beginning to the end. The predicate $e \prec_r e'$ means that event e precedes event e' in the specification of role r.

Lemma 2. *Let α be a trace of a protocol, let (rid, ρ, σ) be an instantiation and e', e events, such that $e' \prec_r e$ for some role r. If for some i $(0 \leq i <\mid \alpha \mid)$ $\alpha_i = (rid, \rho, \sigma, e)$ then there exists j $(0 \leq j < i)$ and $\sigma' \subseteq \sigma$ such that $\alpha_j = (rid, \rho, \sigma', e')$.*

The correctness of this lemma follows from Table 2 by observing that every run is "pealed off" from the beginning, while taking into account that the *Match* predicate is defined such that it only extends the valuation of the variables.

The next lemma is used to infer from an encrypted message reception that the message must have been sent by an agent if it contains a component which is not known to the intruder. In most applications of this lemma we can infer l' by inspection of the role specification and we have $(rid, \rho, \sigma)(\{m\}_k) = (rid', \rho', \sigma')(m')$, rather than a subterm relation.

Lemma 3. *Let α be a trace and let i be an index of α. If*
$\alpha_i = ((rid, \rho, \sigma), read_\ell(x, y, \{m\}_k))$ and $M_0 \nvdash (rid, \rho, \sigma)(\{m\}_k)$, and $M_i^\alpha \nvdash (rid, \rho, \sigma)(m)$, then there exists index $j < i$ such that
$\alpha_j = (rid', \rho', \sigma', send_{\ell'}(x', y', m'))$ and $(rid, \rho, \sigma)(\{m\}_k) \sqsubseteq (rid', \rho', \sigma')(m')$.

The correctness of this lemma follows from the fact that if the intruder does not know m when the message containing $\{m\}_k$ is read, he could not have constructed the encryption. Thus, it must have been sent as a subterm earlier.

The final lemma is characteristic for our model. It expresses that when two instantiations of a constant (such as a nonce or session key) are equal, they were created in the same run.

Lemma 4. *Let (rid, ρ, σ) and (rid', ρ', σ') be instantiations, and let $n \in \mathcal{C}$. If $(rid, \rho, \sigma)(n) = (rid', \rho', \sigma')(n)$ we have $rid = rid'$.*

Theorem 1. *The Needham-Schroeder-Lowe protocol is correct in the Dolev-Yao intruder model with conspiring agents and without type flaws.*

Proof. We will sketch the proofs for $claim_7$ and $claim_9$. The other claims are proven analogously.

First observe that the intruder will never learn secret keys of trusted agents. This follows directly from Lemma 1, since none of the messages contain an encryption key in the message text. Since the set of keys known to the intruder is constant, it must be the case that if the intruder learns a basic term he learns it from unpacking an intercepted message which was encrypted with the key of an untrusted agent.

Proof outline We construct proofs for the Needham-Schroeder-Lowe protocol. The proof construction would fail for the Needham-Schroeder protocol, and we will use a marker † to indicate where the difference occurs. After the proof of $claim_7$, we briefly discuss this difference.

Both proofs will roughly follow the same structure. We examine the occurrence of a claim event in a trace of the system. Based on the rules of the semantics, we gradually derive more information about the trace, until we can conclude that the required property holds.

Proof of $claim_7$. In order to prove $claim_7$ we assume that α is a trace with index $r7$, such that $\alpha_{r7} = ((rid_{r7}, \rho_{r7}, \sigma_{r7}), claim_7(r, secret, nr))$ and $rng(\rho_{r7}) \subseteq \mathcal{A}_T$. Now we assume that the intruder learns nr and we will derive a contradiction. Let k be the smallest index such that $(rid_{r7}, \rho_{r7}, \sigma_{r7})(nr) \in M_{k+1}$, and thus $(rid_{r7}, \rho_{r7}, \sigma_{r7})(nr) \notin M_k$. Inspection of the derivation rules learns that this increase in knowledge is due to an application of the send rule, followed by an application of the deflect rule. Therefore, there must be a smallest index $p < k$ such that $\alpha_p = ((rid', \rho', \sigma'), send_\ell(m))$ and $(rid_{r7}, \rho_{r7}, \sigma_{r7})(nr) \sqsubseteq (rid', \rho', \sigma')(m)$. Since we have three possible send events in the NSL protocol, we have three cases: $\ell = 1, 2$, or 3.

[$\ell = 1$] In the first case we have $\alpha_p = ((rid', \rho', \sigma'), send_1(i, r, \{i, ni\}_{pk(r)}))$. Since constants i and ni both differ from nr, the intruder cannot learn $(rid_{r7}, \rho_{r7}, \sigma_{r7})(nr)$ from $(rid', \rho', \sigma')(i, r, \{i, ni\}_{pk(r)})$, which yields a contradiction.

[$\ell = 2$] In the second case $\alpha_p = ((rid', \rho', \sigma'), send_2(r, i, \{W, nr, r\}_{pk(i)}))$. The intruder can learn nr because $\rho'(i)$ is an untrusted agent and either $(rid_{r7}, \rho_{r7}, \sigma_{r7})(nr) = (rid', \rho', \sigma')(W)$ or $(rid_{r7}, \rho_{r7}, \sigma_{r7})(nr) = (rid', \rho', \sigma')(nr)$. We discuss both options separately.

(i) For the former equality we derive that $(rid', \rho', \sigma')(W) \notin M_p$, so we can apply Lemmas 2 and 3 to find $i1$ with $\alpha_{i1} = ((rid_{i1}, \rho_{i1}, \sigma_{i1}), send_1(i, r, \{i, ni\}_{pk(r)}))$. This gives $(rid_{i1}, \rho_{i1}, \sigma_{i1})(ni) = (rid', \rho', \sigma')(W) = (rid_{r7}, \rho_{r7}, \sigma_{r7})(nr)$, which cannot be the case since ni and nr are distinct constants.

(ii) That the latter equality yields a contradiction is easy to show. Using Lemma 4 we derive $rid_{r7} = rid'$ and since run identifiers are unique, we have $\rho_{r7} = \rho'$. So $\rho_{r7}(i) = \rho'(i)$, which contradicts the assumption that $\rho_{r7}(i)$ is a trusted agent.

[$\ell = 3$] In the third case we have $\alpha_p = ((rid', \rho', \sigma'), send_3(i, r, \{V\}_{pk(r)}))$. In order to learn $(rid_{r7}, \rho_{r7}, \sigma_{r7})(nr)$ from $(rid', \rho', \sigma')(i, r, \{V\}_{pk(r)})$ we must have that
$(rid', \rho', \sigma')(V) = (rid_{r7}, \rho_{r7}, \sigma_{r7})(nr)$ and that $\rho'(r)$ is an untrusted agent. Using Lemma 2 we find index $i2$ such that $\alpha_{i2} = ((rid', \rho', \sigma'), read_2(r, i, \{ni, V, r\}_{pk(i)})$. Because $(rid', \rho', \sigma')(V) \notin M_p$ we can apply Lemma 3 to find index $r2$ with $\alpha_{r2} = ((rid_{r2}, \rho_{r2}, \sigma_{r2}), send_2(r, i, \{W, nr, r\}_{pk(i)})$.
This gives $\rho'(r) = \rho_{r2}(r)$. (†)

Next, we derive $(rid_{r2}, \rho_{r2}, \sigma_{r2})(nr) = (rid', \rho', \sigma')(V) = (rid_{r7}, \rho_{r7}, \sigma_{r7})(nr)$. Applying Lemma 4 yields $rid_{r2} = rid_{r7}$ and thus $\rho_{r2} = \rho_{r7}$, so $\rho'(r) = \rho_{r2}(r) = \rho_{r7}(r)$. Because $\rho'(r)$ is an untrusted agent while $\rho_{r7}(r)$ is trusted, we obtain a contradiction. This finishes the proof of $claim_7$.

Note †: Please notice that the step in the proof marked with † fails for the Needham-Schroeder protocol, which gives an indication of why the hardening of the second message exchange is required.

Proof of $claim_9$. Let $\alpha \in Tr(nsl)$ be a trace of the system. Suppose that for some $r9$ and $(rid_r, \rho_r, \sigma_{r9}) \in Inst$, with $rng(\rho_r) \subseteq \mathcal{A}_T$, we have
$\alpha_{r9} = ((rid_r, \rho_r, \sigma_{r9}), claim_9(r, nisynch))$. In order to prove this synchronisation

claim correct, we must find a run executing the initiator role which synchronises on the events labeled 1, 2, and 3, since $\mathit{prec}(\mathit{nsl}, 9) = \{1, 2, 3\}$. By applying Lemma 2, we find $r1$, $r2$, $r3$ $(0 \leq r1 < r2 < r3 < r9)$ and $\sigma_{r1} \subseteq \sigma_{r2} \subseteq \sigma_{r3} \subseteq \sigma_{r9}$, such that

$$\begin{aligned}
\alpha_{r1} &= ((\mathit{rid}_r, \rho_r, \sigma_{r1}), \mathit{read}_1(i, r, \{i, W\}_{pk(r)}))\\
\alpha_{r2} &= ((\mathit{rid}_r, \rho_r, \sigma_{r2}), \mathit{send}_2(r, i, \{W, nr, r\}_{pk(i)}))\\
\alpha_{r3} &= ((\mathit{rid}_r, \rho_r, \sigma_{r3}), \mathit{read}_3(i, r, \{nr\}_{pk(r)})).
\end{aligned}$$

We have already proved that nr remains secret, so we can apply Lemma 3 and find index $i3$ and $(\mathit{rid}_i, \rho_i, \sigma_{i3})$ such that $i3 < r3$ and
$\alpha_{i3} = ((\mathit{rid}_i, \rho_i, \sigma_{i3}), \mathit{send}_3(i, r, \{V\}_{pk(r)})) \wedge (\mathit{rid}_r, \rho_r, \sigma_{r3})(nr) = (\mathit{rid}_i, \rho_i, \sigma_{i3}(V)$.
By applying Lemma 2 we obtain $i1 < i2 < i3$ such that

$$\begin{aligned}
\alpha_{i1} &= ((\mathit{rid}_i, \rho_i, \sigma_{i1}), \mathit{send}_1(i, r, \{i, ni\}_{pk(r)}))\\
\alpha_{i2} &= ((\mathit{rid}_i, \rho_i, \sigma_{i2}), \mathit{read}_2(r, i, \{ni, V, r\}_{pk(i)}))\\
\alpha_{i3} &= ((\mathit{rid}_i, \rho_i, \sigma_{i3}), \mathit{send}_3(i, r, \{V\}_{pk(r)})).
\end{aligned}$$

Now that we have found out that run rid_i is a candidate, we only have to prove that it synchronises with run rid_r. Therefore, we have to establish $r2 < i2$, $i1 < r1$ and that the corresponding send and read events match each other.

First, we observe α_{i2}. Since $(\mathit{rid}_r, \rho_r, \sigma_{r3})(nr)$ is secret, $(\mathit{rid}_i, \rho_i, \sigma_{i2})(V)$ is secret too and we can apply Lemma 3, obtaining index $r2' < i2$ such that $\alpha_{r2'} = ((\mathit{rid}_{r'}, \rho_{r'}, \sigma_{r2'}), \mathit{send}_2(r, i, \{W, nr, r\}_{pk(i)}))$ such that we have $(\mathit{rid}_i, \rho_i, \sigma_{i2})(\{ni, V, r\}_{pk(i)}) = (\mathit{rid}_{r'}, \rho_{r'}, \sigma_{r2'})(\{W, nr, r\}_{pk(i)})$. This implies that we have $(\mathit{rid}_r, \rho_r, \sigma_{r3})(nr) = (\mathit{rid}_i, \rho_i, \sigma_{i3}(V) = (\mathit{rid}_{r'}, \rho_{r'}, \sigma_{r2'})(nr)$, so from Lemma 4 we have $\mathit{rid}_r = \mathit{rid}_{r'}$, and thus $r2 = r2'$. This establishes synchronisation of events α_{i2} and α_{r2}.

Next, we look at α_{r1}. Because $(\mathit{rid}_r, \rho_r, \sigma_{r1})(W)$ is secret (cf. claim 8), we can apply Lemma 3, which gives index $i1' < r1$ such that
$\alpha_{i1'} = ((\mathit{rid}_{i'}, \rho_{i'}, \sigma_{i1'}), \mathit{send}_1(i, r, \{i, ni\}_{pk(r)}))$ and $(\mathit{rid}_r, \rho_r, \sigma_{r1})(\{i, W\}_{pk(r)})) = (\mathit{rid}_{i'}, \rho_{i'}, \sigma_{i1'})(\{i, ni\}_{pk(r)})$. Correspondence of α_{i2} and α_{r2} gives
$(\mathit{rid}_i, \rho_i, \sigma_{i2})(ni) = (\mathit{rid}_r, \rho_r, \sigma_{r2})(W) = (\mathit{rid}_r, \rho_r, \sigma_{r1})(W) = (\mathit{rid}_{i'}, \rho_{i'}, \sigma_{i1'})(ni)$.
By lemma 4 rid_i and $\mathit{rid}_{i'}$ are equal, which establishes synchronisation of events α_{r1} and α_{i1}. This finishes the synchronisation proof of claim_9.

5 Related work

There is a wealth of different approaches for the modeling of security protocols. Very often the focus is on verification tools, yielding a model which is only informally or implicitly defined.

We will briefly compare our approach to the three prominent approaches: BAN logic (because of its historic interest), Casper/FDR (because it has powerful tool support), and Strand spaces (because this approach has much in common with ours). We conclude with short remarks on the spi calculus and modeling security protocols as open systems.

In 1989 Burrows, Abadi and Needham published their ground breaking work on a logic for the verification of authentication properties [2]. In this so-called BAN-logic, predicates have the form "P believes X". Such predicates are derived from a set of assumptions, using derivation rules like "If P believes that P and Q share key K, and if P sees message $\{X\}_K$ then P believes that Q once said X". Note that this rule implies a peculiarity of the agent model, which is not required in most other approaches, viz. an agent can detect (and ignore) his own messages. The BAN-logic has a fixed intruder model, which does not consider conspiring agents. The Needham-Schroeder protocol (see Figure 1) was proven correct in BAN-logic because the man-in-the-middle attack from Figure 2 could not be modeled. Another major difference with our approach is that the BAN-logic uses a rather weak notion of authentication. The authentication properties verified for most protocols have the form "A believes that A and B share key K" (or "...share secret X"), and "A believes that B believes that A and B share key K". This weak form of agreement is sometimes even further reduced to *recent aliveness*. Furthermore, type-flaw attacks cannot be detected using BAN-logic. An interesting feature is that BAN logic treats time stamps at an appropriate abstract level, while an extension of our semantics with time stamps is not obvious. Due to the above mentioned restrictions interest in BAN logic has decreased. Recent research concerns its extension and the development of models for the logic.

Developed originally by Gavin Lowe, the Casper/FDR tool set as described in [10] is not a formal security protocol semantics, but a model checking tool. However, as the input is translated into a CSP process algebraic model, there is an implicit semantics. The reason we mention it here, is that Casper/FDR is a mature tool set, and none of the other semantics we mention has such a tool set available. In the research for Casper/FDR many interesting security properties have been formulated in terms of CSP models (see e.g. [11]) and some of these have been consequently adapted in other models. An advantage of using process algebra for modeling security protocols is that the model is easily extended. However, for Casper/FDR there is no explicit formal semantics of the protocol language and properties except in terms of CSP. Because of this, it is difficult to get results about properties besides using the tools.

The Strand space approach [12] is closely related to the use of Message Sequence Charts which we advocate for the description of protocols and protocol runs. Roughly, the difference is that we provide a totally ordered semantics, whereas Strand spaces describe a partial order on the events. The notion of a strand is similar to our notion of run, and a strand space is the set of all possible combinations of strands, reflecting our semantical model of interleaved runs. Strand spaces seem to be very tightly linked to the Dolev-Yao intruder model and although the intruder is modeled as a collection of strands, just like normal agents, it is not easy to vary over the intruder network model. With respect to the security properties, we mention that both secrecy and agreement are expressible in the Strand spaces model. Additional research must indicate whether synchronisation can be expressed. Finally, we mention that our focus on security claims which are local to the agent's run is not reflected in Strand spaces.

As an example of a process calculus approach, we have the spi calculus developed by Abadi and Gordon in [13]. It is an extension of the pi calculus in [14]. Although this has advantages, it also inherits properties of the pi calculus that do not immediately seem useful for security protocol analysis. As an example, expressing that a run is synchronising with another run over multiple messages is non-trivial, because it can be hard to tell two runs of the same role (with identical parameters) apart. To always be able to distinguish two runs, additional constructs are needed as in [15]. Having an explicit run identifier in the semantics makes it easier to express such properties.

Recently, Martinelli has proposed to analyse security protocols as open systems in [16]. A process calculus for security protocols is proposed, where the intruder process is left unspecified. This allows for protocol properties to be checked with respect to any intruder, which (for safety properties) amounts to the Dolev-Yao intruder model. Properties can also be checked or with respect to a specific intruder, which is similar to having different intruder rules in our semantics. Two main drawbacks are that the analysis assumes a finite number of agents and runs, and that it cannot be used to find type flaw attacks.

In the methods mentioned here, the construction of the initial intruder knowledge is left implicit.

6 Conclusions and future research

We have developed a generic canonical model for fundamental analysis of security protocols. Some characteristics of this model are that we give explicit static requirements for valid protocols, and that the model is parametric with respect to the matching function and intruder network capabilities. Multi-protocol analysis, by which we mean the analysis of running several different protocols or protocol roles concurrently, is handled in an intuitive way by simply adding more role descriptions to the model. In line with this, security properties are defined as local claims. Furthermore, local constants are bound to runs, which can assist in the construction of proofs.

As future work, we will be formulating metaresults. For instance, we are interested in results about the composition of protocols, and the decomposition of problems into simpler components. Related to this are transformations of protocols in a given intruder model such that the same security properties are met.

Results in composition of protocols can lead to security by construction. Given a set of security properties and an intruder model, we would like to construct a correct protocol.

We have already developed a tool for model checking secrecy based on this model [17]. Future work will be to develop this into a mature toolset. Parallel to this we are investigating state space reduction techniques for certain settings in our model, such as only eavesdropping, and specific properties.

References

1. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21** (1978) 993–999
2. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Transactions on Computer Systems **8** (1990) 18–36
3. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Proceedings of TACAS. Volume 1055., Springer Verlag (1996) 147–166
4. Mauw, S., Wiersma, W., Willemse, T.: Language-driven system design. International Journal of Software Engineering and Knowledge Engineering (2004) To appear.
5. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **IT-29** (1983) 198–208
6. Engels, A.G., Mauw, S., Reniers, M.: A hierarchy of communication models for Message Sequence Charts. Science of Computer Programming **44** (2002) 253–292
7. Plotkin, G.: A structural approach to operational semantics. Technical Report DIAMI FN-19, Computer Science Department, Aarhus University (1981)
8. Cremers, C., Mauw, S., de Vink, E.: Defining authentication in a trace model. In Dimitrakos, T., Martinelli, F., eds.: FAST 2003. Proceedings of the first international Workshop on Formal Aspects in Security and Trust, Pisa, IITT-CNR technical report (2003) 131–145
9. Roscoe, A.W.: Intensional Specifications of Security Protocols. In: Proc. 9th Computer Security Foundations Workshop, IEEE (1996) 28–38
10. Lowe, G.: Casper: A compiler for the analysis of security protocols. In: Proc. 10th Computer Security Foundations Workshop, IEEE (1997) 18–30
11. Lowe, G.: A hierarchy of authentication specifications. In: Proc. 10th Computer Security Foundations Workshop, IEEE (1997) 31–44
12. Thayer Fábrega, F., Herzog, J., Guttman, J.: Strand spaces: Why is a security protocol correct? In: Proc. 1998 IEEE Symposium on Security and Privacy, Oakland, California (1998) 66–77
13. Abadi, M., Gordon, A.: A calculus for cryptographic protocols: The spi calculus. Inf. Comput. **148** (1999) 1–70
14. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i. Inf. Comput. **100** (1992) 1–40
15. Bodei, C., Degano, P., Focardi, R., Priami, C.: Primitives for authentication in process algebras. Theor. Comput. Sci. **283** (2002) 271–304
16. Martinelli, F.: Analysis of security protocols as open systems. Theor. Comput. Sci. **290** (2003) 1057–1106
17. Cremers, C., Mauw, S.: Checking secrecy by means of partial order reduction. In Amyot, D., Williams, A., eds.: SAM 2004: Security Analysis and Modelling. Volume LNCS 3319 of Proceedings of the fourth SDL and MSC Workshop., Ottawa, Canada, Springer-Verlag, Berlin (2004) 177–194