# Automated Refinement of Security Protocols*

Anders M. Hagalisletto
University of Oslo, Department of Computer Science,
Postbox 1080 Blindern, 0316 Oslo, Norway
Email:andersmo@ifi.uio.no

## Abstract

*The design of security protocols is usually performed manually by pen and paper, by experts in security. Assumptions are rarely specified explicitly. We present a new way to approach security specification: The protocol is refined fully automated into a specification that contains assumptions sufficient to execute the protocol. As a result, the protocol designer using our method does not have to be a security expert to design a protocol, and can learn immediately how the protocol should work in practice.*

**Keywords:** Security protocols, formal specification, automated refinement, testing, agent theory.

## 1 Introduction

Security protocols belong to the core of computer security, combining classical disciplines like communication, cryptography and software engineering. Although the number of security protocols increases (the still highly relevant and thorough survey [2] gives more than 41 authentication protocols), surprisingly little attention has been devoted to the process of designing and testing security protocols. Even though the security community has contributed over the last decade with secrecy proofs within several calculi and attacks discovered through model checking, new protocols are still designed using pen and paper[1].

Recent advances in distributed computer science suggest a need for more support in the designing and testing of protocols. The end-user of the specification, in most cases a programmer, is supposed to understand and take for granted the specification as presented by the expert. However, even the experts make serious mistakes. Errors in protocol specifications can be classified into three main categories, errors with respect to *syntax*, *assumptions*, and *security*.

Therefore it is timely to automate the process of prototyping security protocols. However, taking security protocols into automated software engineering, involves some particular challenges that distinguishes it from standard software engineering: There is no common paradigm of programming languages that can be applied directly. Protocols are language-independent in a strong sense. Therefore the choice of specification language and semantics is controversial. Concurrency plays a crucial role, the protocols might run in environments with unreliable networks. Unlike computer science in general, computer security in general and security protocols in particular, involves intentional concepts, concepts of beliefs, attitudes of agents, and relations between agents, as the relation of trust. All these elements are part of our language, which consequently builds on temporal epistemic logic. The benefits of our methods can be summarized as follows;

- we provide a uniform way to specify protocols,
- assumptions may be specified explicitly,
- assumptions may be constructed fully automated,

The work presented in this paper is part of a large project, involving a specification language for security properties [4] and a simulator for executing protocols implemented in Maude [9]. Protocol analysis using specifications in our language is possible since the simulator is implemented in Maude. However, neither protocol analysis (by hand-proof or semi-automated) nor formal semantics (denotational or operational) is the focus of this paper. This paper is devoted to the specification and analysis of protocol syntax. We shall

*The author would like to thank Boriss Mejias, Olaf Owe, Bjarte M. Östvold, Habtamu Abie, Atle Refsdal, Arild B. Torjussen and Thor Kristoffersen for comments on earlier drafts of this paper.

[1] Evolutionary development of protocols from a given specification of security goals have been proposed (see [11] and [5]). These protocols are derived fully automated based on weight criterias.

explain the language informally and show how protocol specifications can be used for rapid testing of the intended behaviour of protocols. This paper can be summarized in one single question:

> *What do we know about a protocol by considering only its specification?*

Fortunately, the answer is: *We know a lot!* Protocol specifications as given in the literature contain much implicit information that can be exploited in a formal language.

The paper is organized as follows: In section 2, we present an example of an authentication protocol, the Wide Mouthed Frog. The *textbook* version of the protocol can be given a direct translation in our language. A formal language for security specifying security properties is presented and we motivate how this language can be extended to a language for security protocols. Section 3 shows how temporal epistemic logic can be used to precisely define the set of valid textbook protocols. In order both to analyze and design protocol specifications, two additional notions are introduced, *sub-protocols* and *protocol composition*. It turns out that *orderings* and *algebra* of protocols are exactly what we need in order to automate the refinement of textbooks into executable assumption protocols. In section 4 we show how this formal specification can be refined into an *assumption* version of the protocol that contains the sufficient assumptions about required beliefs and necessary actions performed by the agents. Then finally in section 6 we compare our approach with related work, evaluate our approach, and point to some future research directions.

## 2 A language for protocols

Mike Burrows proposed the following authentication protocol, called the Wide Mouthed Frog, which we shall use to illustrate our language and methods throughout the paper. It consists of two messages, involving three parties, $A$ is trusted to generate the session key $K_{AB}$, $S$ forwards the session key to $B$. Timestamps are added by both $A$ and $S$ to ensure freshness of the session. In standard notations for security protocol specification it reads:

$$\begin{array}{llll} (P_1) & A \longrightarrow S & : & A, E(K_{AS} : T_A, B, K_{AB}) \\ (P_2) & S \longrightarrow B & : & E(K_{BS} : T_S, A, K_{AB}) \end{array}$$

The notation $A \longrightarrow S \ : \ A, E(K_{AS} : T_A, B, K_{AB})$ means that agent $A$ sends to $S$ the message consisting of the agent name $A$, followed by the message containing a time-stamp $T_A$, the agent name $B$, and the session key $K_{AB}$ encrypted using the symmetric key $K_{AS}$.

A language of security $\mathscr{L}_S$ can be defined as follows: The terms in $\mathscr{L}_S$ are of two basic sorts, Agents and Messages. The agent names are typically written $a$, $b$, $c$, …, while messages are written $m$, $m_1$, $m_2$, …. *Agent variables* are written $x, y, x_1, x_2, \ldots$, and *agent-terms* $t^A$, $t^A_1$, $t^A_2$, … In $\mathscr{L}_S$ there are two distinguished atomic sentences; $\mathsf{Transmit}(a,b,m), \mathsf{Agent}(a) \in \mathsf{Atomic}$. The sentence $\mathsf{Transmit}(a,b,m)$ reads; "$a$ sends the message $m$ to $b$" while $\mathsf{Agent}(a)$ reads "$a$ is an agent". Second order variables $X, Y$ are place-holders for arbitrary sentences.

**Definition 1** *If $a, b,$ are agent terms, $x$ is an agent variable, and $m$ is a message, then $\mathscr{L}_S$ is the least language such that:*

| | |
|---|---|
| $(i)$ | $\mathsf{Atomic} \subset \mathscr{L}_S$. |
| $(ii)$ | *If* $\varphi,\ \psi \in \mathscr{L}_S$, *then* $\neg\varphi,\ \varphi \to \psi \in \mathscr{L}_S$. |
| $(iii)$ | *If* $\varphi \in \mathscr{L}_S$, *then* $\mathsf{Bel}_a(\varphi),\ \varphi\,\mathcal{U}\,\psi \in \mathscr{L}_S$. |
| $(iv)$ | *If* $\varphi \in \mathscr{L}_S$ *then* $\forall x\, \varphi \in \mathscr{L}_S$. |
| $(v)$ | *If* $\Phi \in \mathscr{L}_S$ *then* $\forall X\ \Phi \in \mathscr{L}_S$. |

As usual $\wedge$, $\vee$ and $\leftrightarrow$ are definable using $\neg$ and $\to$ and $\top = \varphi \to \varphi$. The operator $\mathsf{Bel}_a(\varphi)$ reads "the agent $a$ believes $\varphi$ holds". The *until* operator $\varphi\,\mathcal{U}\,\psi$ means that $\varphi$ holds until $\psi$ holds. The operator *before* $\mathscr{B}$ is definable from $\mathcal{U}$ by $\varphi\,\mathscr{B}\,\psi = \neg(\neg\varphi\,\mathcal{U}\,\psi)$.

The language $\mathscr{L}_S$ can be used to define both *single agent properties* and *security properties* [4]. A single agent property (SAP) is a characterization of the capabilities and behaviour of a single agent. Below we have defined some SAP's explicitly:

$$\begin{array}{l} \mathsf{Honest}(a) \iff \forall X\, \forall x\, (\mathsf{Transmit}(a,x,X) \to \mathsf{Bel}_a(X)) \\ \mathsf{Conscious}(a) \iff \\ \forall X\, \forall x\, (\mathsf{Transmit}(a,x,X) \to \mathsf{Bel}_a(\mathsf{Transmit}(a,x,X))) \\ \wedge(\mathsf{Transmit}(x,a,X) \to \mathsf{Bel}_a(\mathsf{Transmit}(x,a,X))) \end{array}$$

The good agent is both honest and conscious, as can be observed from the refined protocol specification in section 4. A security property is a relation between two or more agents, as for instance the concept *trust*:

$$\begin{array}{lll} \mathrm{Trust}(a,b,\varphi) & \iff & \mathsf{Transmit}(b,a,\varphi) \to \mathsf{Bel}_a(\varphi) \\ \mathrm{Trust}(a,b) & \iff & \forall X\ \mathrm{Trust}(a,b,X) \end{array}$$

We shall see later that trust plays a crucial role in the execution of security protocols.

The proper extension of $\mathscr{L}_S$ to the language for protocols $\mathscr{L}_P$, is obtained by first extending the terms with notions for protocols and primitives for encryption. The terms of $\mathscr{L}_P$ contain the terms of $\mathscr{L}_S$ and *natural numbers* $N$, $N_1$, $N_2$, …, and three additional sorts, nonces, timestamps, and keys. Variables for the new sorts are labeled with $x^N$, $x^T$, and $x^K$ respectively, when their sorts are emphasized. Constants include *protocol names* $\mu$, $\mu_1$, $\mu_2$, *encryption methods* for cryptography $\mathsf{s}$ (symmetric) and $\mathsf{a}$ (asymmetric), in addition to the indicators $\mathsf{i}$ (private) and $\mathsf{u}$ (public). Then finally there are function symbols for nonces $\mathsf{n}(N, a)$, time-stamps $\mathsf{stamp}(t^T, t^A)$, keys $\mathsf{key}(\mathsf{s}, t_1^A, t_2^A)$, $\mathsf{key}(\mathsf{a}, \mathsf{i}, t^A)$, $\mathsf{key}(\mathsf{a}, \mathsf{u}, t^A)$, and concatenation of protocol names can be formed.

**Definition 2** *Let $a$ denote some agent-term, $x$ an agent-variable, $k$ is a key, and $\mu$ is protocol name, and $N$ is a natural number respectively. Then the language of security protocols $\mathscr{L}_P$ is the least language such that:*

- $(i)$ $\mathscr{L}_S \subseteq \mathscr{L}_P$ *and* $\varepsilon \in \mathscr{L}_P$.
- $(ii)$ $\mathsf{isKey}(k), \mathsf{isNonce}(\mathsf{n}(N, a)), \mathsf{Time}(\mathsf{stamp}(N, a)) \in \mathscr{L}_P$
- $(iii)$ $\mathsf{playRole}(a, x, \mu)$, $\mathsf{role}(a) \in \mathscr{L}_P$.
- $(iv)$ *If* $\varphi \in \mathscr{L}_P$, *then both* $E[k : \varphi]$, $D[k : \varphi] \in \mathscr{L}_P$.
- $(v)$ *If* $\varphi$, $\xi^{\mathcal{T}}$, $\xi^{\mathcal{A}}$, $\xi^{\mathcal{S}} \in \mathscr{L}_P$, *then* $\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,] \in \mathscr{L}_P$.
- $(vi)$ *If* $\varphi \in \mathscr{L}_P$, *then* $\mathsf{Enforce}_a(\varphi) \in \mathscr{L}_P$.

Clause $(ii)$ is self-explanatory. The predicate $\mathsf{role}(t)$ says that agent $t$ plays a specific role inside a protocol session. $\mathsf{role}(t)$ only have meaning in the context of a specific protocol. The ternary predicate $\mathsf{playRole}(t, x, \mu)$ states explicitly that "agent $t$ plays the role $x$ in the protocol named $\mu$". The sentences $E[k : \varphi]$ and $D[k : \varphi]$ denote the basic cryptographic primitives encryption and decryption respectively. The sentence $\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \varphi\,]$ reads "protocol named $\mu$ with session number $N$ with the total roles $\xi^{\mathcal{T}}$, the agent specific roles $\xi^{\mathcal{A}}$, the start-roles $\xi^{\mathcal{S}}$ and the protocol body $\Phi$". The final sentence $\mathsf{Enforce}_a(\varphi)$ reads "enforce agent $a$ to do the sentence $\varphi$" or "agent $a$ does $\varphi$". Enforcement is the only imperative construct in the language, and it can be used by the system specifier or protocol to extend the local belief of the agent.

The syntactical complexity of sentences in $\mathscr{L}_P$, $\deg(\varphi)$, and the free variables $\mathrm{free}(\varphi)$ are defined in the standard way by recursion on the structure of $\varphi$. The events in a protocol are usually positive sentences, like "send message" or "encrypt message". The subset of the language having this property is called the set of *$\mathscr{P}$-positive* sentences. They include the atomic sentences, and composite sentences where a modal operator is the outermost connective; hence each of $E[k : \varphi]$, $D[k : \varphi]$, $\mathsf{Bel}_a(\varphi)$, $\mathsf{Enforce}_a(\varphi)$ and $\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \varphi\,]$ are *$\mathscr{P}$-positive.* Hence sentences where the outermost connective is negation, implication or a temporal connective, are not *$\mathscr{P}$-positive.*

$\mathsf{protocol}$ $[\mathrm{WMF},\ \ 0,\ \ \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z),$
$\mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z),\ \ \mathsf{role}(x),$

$\mathsf{Transmit}(x, z, \mathsf{Agent}(x) \wedge E[\mathsf{key}(\mathsf{s}, x, z) :$
$\quad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y))])$
$\mathscr{B}$
$\mathsf{Transmit}(z, y, E[\mathsf{key}(\mathsf{s}, y, z) : \mathsf{Time}(\mathsf{stamp}(u^T, z))$
$\quad \wedge\ \mathsf{Agent}\ (x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y))])]$

**Figure 1. The Wide Mouthed Frog.**

## 3 Specification of protocols

An event is the minimal unit in a protocol specification, for instance the transmission event or the empty event $\varepsilon$ also called *skip.* A protocol is a chain of events between agents. A *chain of events* is a sentence of the form;
$\varphi_1 \mathscr{B} \varphi_2 \wedge \varphi_2 \mathscr{B} \varphi_3 \wedge \ldots \wedge \varphi_{n-1} \mathscr{B} \varphi_n$, where each $\varphi_i$ is *$\mathscr{P}$-positive.* If the last event is the empty event, $\varphi_n = \varepsilon$, then the chain is called *final.* Skip can only occur as the tail in a chain of events or alone as the empty chain $\varepsilon$. Let $\Phi = \varphi \mathscr{B} [\Phi']$ denote a chain of events written by recursion, hence $\varphi$ is a single event and $\psi$ chain of events. The length of a chain of events is given by $lth(\varepsilon) = 0$ and $lth(\varphi \mathscr{B} [\Phi']) = 1 + lth(\Phi')$.

Let $\varphi$ and $\psi$ be conjunctions of *$\mathscr{P}$-positive* sentences. Then $\varphi$ is *included* in $\psi$, denoted $\varphi \sqsubseteq \psi$, iff either $(i)$ $\varphi = \top$, or

- $(ii)$ $\varphi = (\phi \wedge \Psi)$, $\psi = (\phi \wedge \Psi')$ and $\Psi \sqsubseteq \Psi'$
- $(iii)$ $\varphi = (\phi \wedge \Psi)$, $\psi = (\sigma \wedge \Psi')$ and $(\phi \wedge \Psi) \sqsubseteq \Psi'$

for $\sigma \neq \phi$. We first define the class of valid finitary protocols, and then the class of textbook protocols.

**Definition 3** *A* valid protocol *is a sentence in $\mathscr{L}_P$ on the form;* $\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]$, *such that*

- $(i)$ $\xi^{\mathcal{T}}$, $\xi^{\mathcal{A}}$ *and* $\xi^{\mathcal{S}}$ *are finite conjunctions of roles; such that* $\xi^{\mathcal{A}} \sqsubseteq \xi^{\mathcal{T}}$ *and* $\xi^{\mathcal{S}} \sqsubseteq \xi^{\mathcal{T}}$,
- $(ii)$ $\Phi$ *is a final chain of events and*
- $(iii)$ $\mathrm{free}(\xi^{\mathcal{T}}) = \mathrm{free}(\Phi)$.

The specification of the Wide Mouthed Frog in figure 1 is called a *textbook protocol*, since it is close to the way protocols are specified in research articles and textbooks. Note that variables are introduced, since an agent may play several roles in a single protocol.

**Definition 4** *A* textbook protocol*, is a valid protocol* $\mathsf{P}$*, where each single event is a* $\mathsf{Transmit}(x_j, x_k, \xi)$*, where $x_j$, $x_k$ are agent-variables and $\xi \in \mathscr{L}_P$. (For short we say* $\mathsf{P}$ *is* textbook.*)*

### 3.1 Algebra and orderings of protocols

This section contains the sufficient fragment of definitions and results regarding ordering and concatenation of protocol syntax. In a protocol, the agents participating in the execution have different views on the protocol depending on the role they play. They all run what we call *executable sub-protocol instances*.

Each role that an agent might play in a protocol determines a specific sub-protocol. Below we shall define the concept of *sub-protocol*, through the concept of transitive embedding:

**Definition 5** *Let* $\Phi$ *and* $\Psi$ *be final chains of events. Then* $\Phi$ *can be* embedded transitively *into* $\Psi$, *written* $\Phi \sqsubseteq_{\mathrm{E}} \Psi$, *if the following holds:*

$(i)$ $\quad \varepsilon \sqsubseteq_{\mathrm{E}} \Psi$ *and* $\varphi \mathscr{B} [\Phi] \not\sqsubseteq_{\mathrm{E}} \varepsilon$

$(ii)$ $\quad \varphi \mathscr{B} [\Phi] \sqsubseteq_{\mathrm{E}} \varphi \mathscr{B} [\Psi']$ *iff* $\Phi \sqsubseteq_{\mathrm{E}} \Psi'$

$(iii)$ $\quad \varphi \mathscr{B} [\Phi] \sqsubseteq_{\mathrm{E}} \psi \mathscr{B} [\Psi]$ *iff* $\varphi \mathscr{B} [\Phi] \sqsubseteq_{\mathrm{E}} \Psi$, *if* $\varphi \neq \psi$.

Thus for instance $e_2 \mathscr{B} e_4 \mathscr{B} \varepsilon \sqsubseteq_{\mathrm{E}} e_1 \mathscr{B} e_2 \mathscr{B} e_3 \mathscr{B} e_4 \mathscr{B} e_5 \varepsilon$. If $\theta$ is a set of protocol names with $\mu_1, \mu_2 \in \theta$, then $\mu_1 \subseteq_{\mathrm{N}} \mu_2$ means that $\mu_1$ is a *subname* of $\mu_2$. The empty name is denoted $\epsilon$. $\subseteq_{\mathrm{N}}$ is a partial order.

**Definition 6** $\mathsf{P}_1 = \mathsf{protocol}[\mu_1, N_1, \xi_1^{\mathcal{T}}, \xi_1^{\mathcal{A}}, \xi_1^{\mathcal{S}}, \Phi_1]$ *is a* subprotocol *of* $\mathsf{P}_2 = \mathsf{protocol}[\mu_2, N_2, \xi_2^{\mathcal{T}}, \xi_2^{\mathcal{A}}, \xi_2^{\mathcal{S}}, \Phi_2]$, *written* $\mathsf{P}_1 \sqsubseteq_{\mathrm{P}} \mathsf{P}_2$, *if and only if* $(i)$ $\mu_1 \subseteq_{\mathrm{N}} \mu_2$ *and* $N_1 \leqslant N_2$, $(ii)$, $\xi_1^{\mathcal{T}} \sqsubseteq \xi_2^{\mathcal{T}}$, $\xi_1^{\mathcal{A}} \sqsubseteq \xi_2^{\mathcal{A}}$, *and* $\xi_1^{\mathcal{S}} \sqsubseteq \xi_2^{\mathcal{S}}$ *and* $(iii)$ $\Phi_1 \sqsubseteq_{\mathrm{E}} \Phi_2$.

**Theorem 1** $\sqsubseteq$, $\sqsubseteq_{\mathrm{E}}$, *and* $\sqsubseteq_{\mathrm{P}}$ *are partial orders.*

A strong concept of equality may be defined based on the protocol syntax:

**Definition 7** $\mathsf{P}_1 = \mathsf{protocol}[\mu_1, N_1, \xi_1^{\mathcal{T}}, \xi_1^{\mathcal{A}}, \xi_1^{\mathcal{S}}, \Phi_1]$ *is* equal *to* $\mathsf{P}_2 = \mathsf{protocol}[\mu_2, N_2, \xi_2^{\mathcal{T}}, \xi_2^{\mathcal{A}}, \xi_2^{\mathcal{S}}, \Phi_2]$, *written* $\mathsf{P}_1 = \mathsf{P}_2$, *if and only if* $(i)$ $\mu_1 = \mu_2$ *and* $N_1 = N_2$, $(ii)$, $\xi_1^{\mathcal{T}} = \xi_2^{\mathcal{T}}$, $\xi_1^{\mathcal{A}} = \xi_2^{\mathcal{A}}$, *and* $\xi_1^{\mathcal{S}} = \xi_2^{\mathcal{S}}$ *and* $(iii)$ $\Phi_1 = \Phi_2$.

**Lemma 1** $\mathsf{P}_1 = \mathsf{P}_2$ *iff* $\mathsf{P}_1 \sqsubseteq_{\mathrm{P}} \mathsf{P}_2$ *and* $\mathsf{P}_2 \sqsubseteq_{\mathrm{P}} \mathsf{P}_1$.

If $\mu_1$ and $\mu_2$ are protocol names, then $\mu_1\mu_2$ denotes their concatenation. For any set of protocol names $\Theta$, concatenation is a monoid over $\Theta$ with unit $\epsilon$.

**Definition 8** *If* $\Phi$ *and* $\Psi$ *be final chains of events, then their* concatenation, *denoted* $\Phi \frown \Psi$, *is given by:*

$(i)$ $\quad \Phi \frown \varepsilon = \Phi = \varepsilon \frown \Phi$

$(ii)$ $\quad (\varphi \mathscr{B} [\Phi']) \frown \Psi = \varphi \mathscr{B} [\Phi' \frown \Psi]$

**Lemma 2** *If* $\Phi = \varphi_1 \mathscr{B} \ldots \mathscr{B} \varphi_n \mathscr{B} \varepsilon$ *and* $\Psi = \psi_1 \mathscr{B} \ldots \mathscr{B} \psi_m \mathscr{B} \varepsilon$ *are chain of events, then* $\Phi \frown \Psi = \varphi_1 \mathscr{B} \ldots \mathscr{B} \varphi_n \mathscr{B} \psi_1 \mathscr{B} \ldots \mathscr{B} \psi_m \mathscr{B} \varepsilon$.

**Definition 9** *Let* $\mathsf{P}_1$ *and* $\mathsf{P}_2$ *be two arbitrary valid protocols in* $\mathscr{L}_S$. *Hence* $\mathsf{P}_1 = \mathsf{protocol}[\mu_1, N_1, \xi_1^{\mathcal{T}}, \xi_1^{\mathcal{A}}, \xi_1^{\mathcal{S}}, \Phi_1]$ *and* $\mathsf{P}_2 = \mathsf{protocol}[\mu_2, N_2, \xi_2^{\mathcal{T}}, \xi_2^{\mathcal{A}}, \xi_2^{\mathcal{S}}, \Phi_2]$. *The* composition *of* $\mathsf{P}_1$ *with* $\mathsf{P}_2$, *denoted* $\mathsf{P}_1 \boxplus \mathsf{P}_2$, *is defined by:*

$$\mathsf{protocol}[\mu_1\mu_2, N_1 + N_2, \xi_1^{\mathcal{T}} \wedge \xi_2^{\mathcal{T}}, \xi_1^{\mathcal{A}} \wedge \xi_2^{\mathcal{A}}, \xi_1^{\mathcal{S}} \wedge \xi_2^{\mathcal{S}}, \Phi_1 \frown \Phi_2].$$

Let $\mathcal{E} = \mathsf{protocol}[\epsilon, 0, \top, \top, \top, \varepsilon]$ denote the *empty protocol*, and let $\mathscr{P}$ denote the set of valid protocols. Then:

**Theorem 2** $\langle \mathscr{P}, \boxplus \rangle$ *is a monoid with unit* $\mathcal{E}$.

That $\langle \mathscr{P}, \boxplus \rangle$ is a *monoid* means the following:

| | |
|---|---|
| If $\mathsf{P}_1, \mathsf{P}_2 \in \mathscr{P}$ then $\mathsf{P}_1 \boxplus \mathsf{P}_2 \in \mathscr{P}$ | $\mathscr{P}$ CL |
| $\mathsf{P}_1 \boxplus (\mathsf{P}_2 \boxplus \mathsf{P}_3) = (\mathsf{P}_1 \boxplus \mathsf{P}_2) \boxplus \mathsf{P}_3$ | $\mathscr{P}$ AS |
| $\mathcal{E} \boxplus \mathsf{P} = \mathsf{P} = \mathsf{P} \boxplus \mathcal{E}$ | $\mathscr{P}$ ID |

**Lemma 3** $\mathsf{P}_1 \sqsubseteq_{\mathrm{P}} \mathsf{P}_2 \wedge \mathsf{P}_3 \sqsubseteq_{\mathrm{P}} \mathsf{P}_4 \rightarrow \mathsf{P}_1 \boxplus \mathsf{P}_3 \sqsubseteq_{\mathrm{P}} \mathsf{P}_2 \boxplus \mathsf{P}_4$

The sub-protocol notion $\sqsubseteq_{\mathrm{P}}$, is rather general, an arbitrary sample of events form a protocol may form a sub-protocol. An important class of sub-protocols, is the *intrinsically connected* sub-protocols. An intrinsically connected sub-protocol $\mathsf{P}_s$ in a protocol $\mathsf{P}$ is a sub-protocol such that $\mathsf{P}_s$ is connected in $\mathsf{P}$. For example, each of $\mathsf{P}_1$, $\mathsf{P}_2$ and $\mathsf{P}_3$ are intrinsically connected in $\mathsf{P}_1 \boxplus \mathsf{P}_2 \boxplus \mathsf{P}_3$. Intuitively one can understand the intrinsically connected sub-protocols as regions within protocols that share a common local concern.

**Definition 10** *A protocol* $\mathsf{P}_1$ *is* intrinsically connected *in a protocol* $\mathsf{P}_2$, *written* $\mathsf{P}_1 \sqsubseteq_{\mathrm{PI}} \mathsf{P}_2$, *iff there exists protocols* $\mathsf{P}'$ *and* $\mathsf{P}''$, *such that* $\mathsf{P}_2 = \mathsf{P}' \boxplus \mathsf{P}_1 \boxplus \mathsf{P}''$.

Fortunately, protocol composition is *functional*:

**Lemma 4** $\mathsf{P}_1 = \mathsf{P}_2 \implies \mathsf{P}' \boxplus \mathsf{P}_1 \boxplus \mathsf{P}'' = \mathsf{P}' \boxplus \mathsf{P}_2 \boxplus \mathsf{P}''$.

## 4 Explicit specification of assumptions

Many assumptions about the underlying implementation are not explicitly stated in the textbook protocol. Since there is no *prima facie* relations of trust between the agents, the agents can not add message content to their beliefs when they receive something from their environment. Authentication protocols are ways of establishing trust. Therefore we require that the agents are honest and that they do not unconditionally trust other agents. Assumptions about the state of a given agent is represented by the belief operator. If it is required that agent $a$ possess a key $k$, the specification yields $\mathsf{Bel}_a(\mathsf{isKey}(k))$. If a specification is on the form, "agent $a$ enforce that $a$ believes" $\mathsf{Enforce}_a(\mathsf{Bel}_a(\varphi))$. Hence we define:

**Definition 11** *Let $x_i$ and $x_j$ be agent-variables and $\psi \in \mathscr{L}_P$. An* assumption protocol *is a valid protocol where each single event is either* $\mathsf{Transmit}(x_j, x_k, \psi)$, $\mathsf{Bel}_{x_i}(\psi)$, *or* $\mathsf{Enforce}_{x_i}(\mathsf{Bel}_{x_i}(\psi))$.

Agents may have the capability of producing fresh nonces and keys and set timestamps. Thus freshness involves yet another extension of the language of security, the *freshness extension* denoted $\mathscr{L}_P^{\text{ext}}$ gives the following $\mathscr{L}_S \subseteq \mathscr{L}_P \subseteq \mathscr{L}_P^{\text{ext}}$:

| | |
|---|---|
| $\mathsf{newNonce}(\mathsf{n}(t^N, t^A))$ | create new nonce |
| $\mathsf{newKey}(\mathsf{key}(\mathsf{s}, t_1^A, t_2^A, M))$ | create new key |
| $\mathsf{Current}(\mathsf{stamp}(N, t^A))$ | current local time |

The use of the double operator $\mathsf{Enforce}_x(\mathsf{Bel}_x(\psi))$ shall be rather restrictive. In this paper five kinds of patterns are used;

$(i)$ $\mathsf{Enforce}_x(\mathsf{Bel}_x(E[k : \psi]))$
$(ii)$ $\mathsf{Enforce}_x(\mathsf{Bel}_x(D[k : \psi]))$
$(iii)$ $\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathrm{Trust}(x, y, \psi)))$
$(iv)$ $\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{newKey}(k)))$
$(v)$ $\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{Current}(t)))$

The sentence $(i)$ states that $x$ tries to perform an encryption of sentence $\psi$ and adds this new sentence to its beliefs. Sentence $(ii)$ expresses that $x$ tries to perform a decryption of sentence $\psi$ and adds this new sentence to its beliefs. Clause $(iii)$ says that $x$ is enforced to trust $y$ with respect to the particular sentence $\psi$. Finally, the agent might be enforced to create a fresh key $(iv)$ and set the current time-stamp derived from its local clock.

### 4.1 Automated refinement

The process of refining a textbook protocol by hand into a specification containing all the assumptions, is both time consuming and error prone. Typically we used from 2-4 days of hard and boring work to specify the assumption version of the classical authentication protocols, yet several errors occurred during the process of specification.

A surprising discovery made during this investigation was that the refinement of textbook protocols can be fully automated. There are two advantages: First, the process of refining a textbook protocol is speeded up dramatically. Using our method, it is a practically feasible task to build a large library of authentication protocols including assumptions. Second, the specifier does not have to be an expert in cryptography in order to specify and test protocols. Principles of both symmetric and asymmetric cryptography are built into the refinement algorithm. A consequence of this is that the automated refinement also gives an automated explanation of the underlying cryptographic mechanisms in the protocol! The core idea is thus that we take a textbook protocol as input, and return an executable refined assumption protocol. For every transmission, preconditions are generated for the sender of the message and receiver's knowledge is maximized.

**Definition 12** *If* $\mathsf{P}$ *is a textbook protocol, then* $\mathsf{P}$ *can be refined (automated) into an assumption protocol by the function* $\Re$ *as follows:*

*(AR0)* $\Re(\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]) = \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Re(\Phi)\,]$

*(AR1)* $\Re(\varepsilon) = \varepsilon$

*(AR2)* $\Re(\mathsf{Transmit}(x, y, F)\,\mathscr{B}\,[\Phi]) =$

$\mathrm{pre}(x, F) \frown (\mathsf{Transmit}(x, y, F)$ $(i)$

$\mathscr{B}$ $\mathsf{Enforce}_y\,(\mathsf{Bel}_y(\mathrm{Trust}(y, x, F)))\,\mathscr{B}\,\varepsilon)$ $(ii)$

$\frown \mathrm{post}(y, F) \frown \Re(\Phi)$ $(iii)$

The clauses AR0 and AR1 take care of the start and end of the automated refinement, respectively. The recursion is carried out through in the final clause AR2, and splits into four successive parts: $(i)$ the sender's assumptions, $(ii)$ trusted transmission, $(iii)$ the receiver's information extraction, and $(iv)$ recursion on the remainder. The function $\mathrm{pre}(x, F)$ thus takes an agent-term $x$, the sender, and message content $F$, what is sent, as arguments and returns a chain of assumptions that is required to be true for the agent $x$ in order for $x$ to be able to transmit the message. The function $\mathrm{post}(y, F)$ returns the sequence of local events that the receiver $y$ should perform in order to be cryptographic competent. Note that our security protocols run in an environment where the agents are supposed to trust the particular content of transmissions in protocol sessions.

**Definition 13** *The assumption function* pre *is defined by recursion on the complexity of the message content:*

*(AA)* $\mathrm{pre}(x, \mathsf{Agent}(t)) = \mathsf{Bel}_x(\mathsf{Agent}(t))\,\mathscr{B}\,\varepsilon$

*(AK)* $\mathrm{pre}(x, \mathsf{isKey}(k)) = \mathsf{Bel}_x(\mathsf{isKey}(k))\,\mathscr{B}\,\varepsilon$

*(AN)* $\mathrm{pre}(x, \mathsf{isNonce}(n)) = \mathsf{Bel}_x(\mathsf{isNonce}(n))\,\mathscr{B}\,\varepsilon$

*(AT)* $\mathrm{pre}(x, \mathsf{Time}(r)) = \mathsf{Bel}_x(\mathsf{Time}(r))\,\mathscr{B}\,\varepsilon$

*(AC)* $\mathrm{pre}(x, F \wedge G) = \mathrm{pre}(x, F) \frown \mathrm{pre}(x, G)$

*(AE1)* $\mathrm{pre}(x, E[\mathsf{key}(\mathsf{s}, x, y) : F]) =$
$\mathrm{pre}(x, F \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y))) \frown$
$\mathsf{Enforce}_x(\mathsf{Bel}_x(E[\mathsf{key}(\mathsf{s}, x, y) : F]))\,\mathscr{B}\,\varepsilon$

*(AE2)* $\mathrm{pre}(x, E[\mathsf{key}(\mathsf{s}, y, z) : F]) =$
$\mathsf{Bel}_x(E[\mathsf{key}(\mathsf{s}, y, z) : F])\,\mathscr{B}\,\varepsilon$ *if* $x \neq y \wedge x \neq z$

The function generating assumptions thus makes *explicit* the content of $F$ in a message $\mathsf{Transmit}(x, y, F)$, with respect to what the sender $x$ should know in order to be able to send the message to $y$.

Similarly, the receiver may extract information from a message, by enforcing as many decryptions as possible. The information extracted is then implicitly possessed by the receiver $y$.

**Definition 14** *The extraction function* post *is defined by recursion on the complexity of the message content:*

$$
\begin{array}{ll}
(PA) & \mathrm{post}(y, \mathsf{Agent}(t)) = \mathrm{post}(x, \mathsf{isKey}(k)) = \\
 & \mathrm{post}(y, \mathsf{isNonce}(n)) = \mathrm{post}(y, \mathsf{Time}(r)) = \varepsilon \\
(PE1) & \mathrm{post}(y, E[\mathsf{key}(\mathsf{s}, y, x) : F]) = \\
 & \quad \mathrm{post}(y, \mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, x))) \\
 & \qquad \frown (\mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{s}, y, x) : \\
 & \qquad\qquad E[\mathsf{key}(\mathsf{s}, y, x) : F]])) \mathscr{B} \varepsilon) \frown \mathrm{post}(y, F) \\
(PE2) & \mathrm{post}(y, E[\mathsf{key}(\mathsf{s}, x, z) : F]) = \varepsilon \ \textit{if}\ y \neq x \wedge y \neq z
\end{array}
$$

**Lemma 5** $\mathit{lth}(\mathsf{P}') = 1 \implies \Re(\mathsf{P}') \boxplus \Re(\mathsf{P}) = \Re(\mathsf{P}' \boxplus \mathsf{P})$.

The next lemma shows that automated refinement is a homomorphism, and is proven using that composition $\boxplus$ is a monoid and functional:

**Theorem 3** $\Re(\mathsf{P}_1) \boxplus \Re(\mathsf{P}_2) = \Re(\mathsf{P}_1 \boxplus \mathsf{P}_2)$

*Proof:* By induction on $\mathit{lth}(\mathsf{P}_1)$. Ind. basis yields:

$$\Re(\mathcal{E}) \boxplus \Re(\mathsf{P}_2) =^{(a)} \mathcal{E} \boxplus \Re(\mathsf{P}_2) =^{(b)} \Re(\mathsf{P}_2) =^{(c)} \Re(\mathcal{E} \boxplus \mathsf{P}_2)$$

$(a)$ follows by definition of $\Re$ AR1, $(b)$ and $(c)$ by $\mathscr{P}$ID. Consider the ind. step. Let $\mathsf{P}_1 = \mathsf{P}' \boxplus \mathsf{P}$, where $\mathit{lth}(\mathsf{P}') = 1$:

$$
\begin{array}{ll}
\Re(\mathsf{P}' \boxplus \mathsf{P}) \boxplus \Re(\mathsf{P}_2) & \text{lemma 4 and 5} \\
= (\Re(\mathsf{P}') \boxplus \Re(\mathsf{P})) \boxplus \Re(\mathsf{P}_2) & \text{theorem 2, } \mathscr{P}\text{AS} \\
= \Re(\mathsf{P}') \boxplus (\Re(\mathsf{P}) \boxplus \Re(\mathsf{P}_2)) & \text{ind. hyp. and lemma 4} \\
= \Re(\mathsf{P}') \boxplus \Re(\mathsf{P} \boxplus \mathsf{P}_2) & \text{lemma 5} \\
= \Re(\mathsf{P}' \boxplus (\mathsf{P} \boxplus \mathsf{P}_2)) & \text{theorem 2, } \mathscr{P}\text{AS} \\
= \Re((\mathsf{P}' \boxplus \mathsf{P}) \boxplus \mathsf{P}_2), & \text{which is what we wanted.}
\end{array}
$$

■

**Lemma 6** *If* $\mathsf{P}$ *is a textbook protocol containing only one message transmission, then* $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re(\mathsf{P})$*.*

**Theorem 4** *If* $\mathsf{P}$ *is textbook, then* $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re(\mathsf{P})$*.*

*Proof:* By induction over $\mathit{lth}(\mathsf{P})$. The basis is obvious using (AR-1). Consider the induction step: Suppose that $\mathit{lth}(\mathsf{P}) = k$ and that $\xi^{\mathcal{A}} = \mathsf{role}(x_1) \wedge \ldots \wedge \mathsf{role}(x_n)$. By induction hypothesis $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re(\mathsf{P})$. Suppose without loss of generality that $\mathsf{P}$ is extended with one clause at the end. Since we consider arbitrary extensions of the protocol, it is convenient to consider the extension as a protocol addition: $\mathsf{P} \boxplus \mathsf{P}'$, where $\mathit{lth}(\mathsf{P}') = 1$. Since $\mathsf{P}' \sqsubseteq_{\mathrm{P}} \Re(\mathsf{P}')$, then

$$\mathsf{P} \boxplus \mathsf{P}' \sqsubseteq_{\mathrm{P}} (\Re(\mathsf{P}) \boxplus \Re(\mathsf{P}')) = \Re(\mathsf{P} \boxplus \mathsf{P}')$$

which follows by the monotonicity of automated refinement over the sub-protocol relation (lemma 3) and since $\Re$ is a homomorphism (theorem 3). ■

The previous algorithm can handle neither freshness nor duplication of protocol statements. Superfluous information appears normally in the refinement process by duplicated belief statements, since $\Re$ typically traverses the same elementary facts several time.

**Theorem 5** *Any textbook protocol* $\mathsf{P}$*, can be refined fully automated into an assumption protocol* $\mathsf{P}^\star$ *with explicit generation of fresh timestamps and nonces.*

*Proof:* Let $\mathsf{P}' = \Re(\mathsf{P})$ be an automated refined protocol. The function $\mathrm{eq}(\mathsf{P}')$ recursively traverses the protocol body and only keeps the first occurrence of every belief statement and removes the remaining. $\mathsf{P}'' = \mathrm{eq}(\mathsf{P}')$. Since belief statements of the kinds

$$\mathsf{Bel}_x(\mathsf{isNonce}(\mathsf{n}(z^N, x))) \text{ and } \mathsf{Bel}_x\,(\mathsf{Time}(\mathsf{stamp}(w^T, x)))$$

occur as early as possible in the assumption protocol, each occurrence might be replaced by statements creating the fresh local values. The function $\star$ thus takes $\mathsf{P}''$ as argument and returns an explicit freshness protocol: Each occurrence of $\mathsf{Bel}_x(\mathsf{isNonce}(\mathsf{n}(z^N, x)))$ is replaced by $\mathsf{newNonce}(\mathsf{n}(z^N, x))$ and each occurrence of $\mathsf{Bel}_x(\mathsf{Time}(\mathsf{stamp}(w^T, x)))$ is replaced by $\mathsf{Current}(\mathsf{stamp}(w^T, x))$. Then $\mathsf{P}^\star = \star(\mathrm{eq}(\Re(\mathsf{P})))$. ■
By theorem 5 we assure that the agent's nonces and timestamps are explicitly constructed in the specification. If $\mathsf{P}$ is a textbook protocol then let $\Re^\star$ denote the function constructed in the proof of theorem 5, that is: $\Re^\star(\mathsf{P}) = \star(\mathrm{eq}(\Re(\mathsf{P})))$. Figure 2 shows how the algorithm refines the textbook specification. Fortunately, the previous results about straightforward automated refinement can be transfered to automated refinement including fresh nonces and timestamps, although the proof is more delicate. The reason is that neither $\star$ nor eq is a homomorphism over $\boxplus$ ; in other words $\mathrm{eq}(\mathsf{P}_1) \boxplus \mathrm{eq}(\mathsf{P}_2) \neq \mathrm{eq}(\mathsf{P}_1 \boxplus \mathsf{P}_2)$ and $\star(\mathsf{P}_1) \boxplus \star(\mathsf{P}_2) \neq \star(\mathsf{P}_1 \boxplus \mathsf{P}_2)$. Hence we can not reuse the proof of theorem 4, since theorem 3 does not generalize to freshness refinement $\Re^\star$. Fortunately, we can prove a couple of useful properties

**Lemma 7** *Let* $\mathsf{P}_1$ *and* $\mathsf{P}_2$ *be textbook protocols, then*
$(i)$ $\mathrm{eq}(\Re^\star(\mathsf{P}_1) \boxplus \Re^\star(\mathsf{P}_2)) = \mathrm{eq}(\Re^\star(\mathsf{P}_1 \boxplus \mathsf{P}_2))$
$(ii)$ *Both* $\mathrm{eq}(\mathsf{P}_1) = \mathsf{P}_1$ *and* $\mathrm{eq}(\Re^\star(\mathsf{P}_1)) = \Re^\star(\mathsf{P}_1)$

**Lemma 8** *If* $\mathsf{P}$ *is a textbook protocol, with* $\mathit{lth}(\mathsf{P}) = 1$*, then we have that* $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P})$*.*

**Theorem 6** *If* $\mathsf{P}$ *is textbook, then* $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P})$*.*

*Proof:* By induction on $\mathit{lth}(\mathsf{P})$. Ind. basis is verified by $\Re^\star(\varepsilon) = \star(\mathrm{eq}(\Re(\varepsilon))) = \varepsilon$. Consider the induction step: Analogous to theorem 4 we consider one-step

$$\begin{array}{l}
\mathsf{protocol}\quad [\mathrm{WMF\text{-}AUTO\text{-}REFINE},\quad 0,\\
\quad \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z), \qquad \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z), \qquad \mathsf{role}(x)
\end{array}$$

$$\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{newKey}(\mathsf{key}(\mathsf{s}, x, y, v^K)))) \tag{1}$$

$\mathscr{B}$

$$\mathsf{Bel}_x(\mathsf{Agent}(x)) \tag{2}$$

$\mathscr{B}$

$$\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{Current}(\mathsf{stamp}(w^T, x)))) \tag{3}$$

$\mathscr{B}$

$$\mathsf{Bel}_x(\mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K)) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z))) \tag{4}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Enforce}_x(\mathsf{Bel}_x(E[\mathsf{key}(\mathsf{s}, x, z) :\\ \quad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]))\end{array} \tag{5}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Transmit}(x, z, \mathsf{Agent}(x) \wedge E[\mathsf{key}(\mathsf{s}, x, z) :\\ \quad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))])\end{array} \tag{6}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Enforce}_z(\mathsf{Bel}_z(\mathrm{Trust}(z, x, E[\mathsf{key}(\mathsf{s}, x, z) :\\ \quad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))])))\end{array} \tag{7}$$

$\mathscr{B}$

$$\mathsf{Bel}_z(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z))) \tag{8}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Enforce}_z(\mathsf{Bel}_z(D[\mathsf{key}(\mathsf{s}, x, z) : E[\mathsf{key}(\mathsf{s}, x, z) :\\ \qquad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]]))\end{array} \tag{9}$$

$\mathscr{B}$

$$\mathsf{Enforce}_z(\mathsf{Bel}_z(\mathsf{Current}(\mathsf{stamp}(u^T, z)))) \tag{10}$$

$\mathscr{B}$

$$\mathsf{Bel}_z(\mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K)) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) \tag{11}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Enforce}_z(\mathsf{Bel}_z(E[\mathsf{key}(\mathsf{s}, y, z) :\\ \quad \mathsf{Time}(\mathsf{stamp}(u^T, z)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]))\end{array} \tag{12}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Transmit}(z, y, E[\mathsf{key}(\mathsf{s}, y, z) :\\ \quad \mathsf{Time}(\mathsf{stamp}(u^T, z)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))])\end{array} \tag{13}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Enforce}_y(\mathsf{Bel}_y(\mathrm{Trust}(y, z, E[\mathsf{key}(\mathsf{s}, y, z) :\\ \quad \mathsf{Time}(\mathsf{stamp}(u^T, x)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y))]))))]\end{array} \tag{14}$$

$\mathscr{B}$

$$\mathsf{Bel}_y(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) \tag{15}$$

$\mathscr{B}$

$$\begin{array}{l}\mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{s}, y, z) : E[\mathsf{key}(\mathsf{s}, y, z) :\\ \qquad \mathsf{Time}(\mathsf{stamp}(u^T, x)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]]))\quad ]\end{array} \tag{16}$$

**Figure 2. Automated refinement of WMF.**

extensions of the protocol: $\mathsf{P} = \mathsf{P}' \boxplus \mathsf{P}''$, where $lth(\mathsf{P}') \geqslant 0$ and $lth(\mathsf{P}'') = 1$. By induction hypothesis $\mathsf{P}' \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P}')$, and by lemma 8, $\mathsf{P}'' \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P}'')$. Then by lemma 3, $\mathsf{P}' \boxplus \mathsf{P}'' \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P}') \boxplus \Re^\star(\mathsf{P}'')$. Now comes the delicate part: Since eq is monotone over $\sqsubseteq_{\mathrm{P}}$, we have that $\mathrm{eq}(\mathsf{P}' \boxplus \mathsf{P}'') \sqsubseteq_{\mathrm{P}} \mathrm{eq}(\Re^\star(\mathsf{P}') \boxplus \Re^\star(\mathsf{P}''))$ (1). By lemma 7 $(i)$, $\mathrm{eq}(\mathsf{P}' \boxplus \mathsf{P}'') = \mathsf{P}' \boxplus \mathsf{P}''$ (2), since $\mathsf{P}$ is a textbook protocol. By lemma 7 $(ii)$ and $(iii)$, $\mathrm{eq}(\Re^\star(\mathsf{P}') \boxplus \Re^\star(\mathsf{P}'')) = \mathrm{eq}(\Re^\star(\mathsf{P}' \boxplus \mathsf{P}''))$ (3), and $\mathrm{eq}(\Re^\star(\mathsf{P})) = \Re^\star(\mathsf{P})$ (4). Then the equations;

$$\begin{array}{l}\mathsf{P}' \boxplus \mathsf{P}'' =^{(1)} \mathrm{eq}(\mathsf{P}' \boxplus \mathsf{P}'') \sqsubseteq_{\mathrm{P}} \mathrm{eq}(\Re^\star(\mathsf{P}') \boxplus \Re^\star(\mathsf{P}''))\\ =^{(2)} \mathrm{eq}(\Re^\star(\mathsf{P}') \boxplus \Re^\star(\mathsf{P}'')) =^{(3)} \mathrm{eq}(\Re^\star(\mathsf{P}' \boxplus \mathsf{P}''))\\ =^{(4)} \Re^\star(\mathsf{P}' \boxplus \mathsf{P}'') \text{ proves the theorem.}\end{array}$$

■

## 4.2 Public key cryptography extension

Let us consider asymmetric cryptography. In public key cryptography the following two axioms hold for any agent $x$, relating to decryption and encryption:

$$\begin{array}{ll}\mathrm{PKI1} & D[\mathsf{key}(\mathsf{a}, \mathsf{i}, x) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, x) : F]] \leftrightarrow F\\ \mathrm{PKI2} & D[\mathsf{key}(\mathsf{a}, \mathsf{u}, x) : E[\mathsf{key}(\mathsf{a}, \mathsf{i}, x) : F]] \leftrightarrow F\end{array}$$

The public key is considered to be a public fact. Every cryptographic competent agent may have access to any public key among the agents participating in the network. The private key is required to be secret, no other agent at the same level of trust is supposed to possess the key.

Both the assumption construction and information extraction functions must be extended. Consider first the assumption construction: There are two cases, either the sender $x$ intends to send a message encrypted with $x$'s public or private key, or the sender $x$ encrypts a message with assumptions for sending messages:

$$\begin{array}{ll}(\mathrm{AE3}) & \mathrm{pre}(x, E[\mathsf{key}(\mathsf{a}, i, x) : F]) =\\ & \quad \mathrm{pre}(x, F \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{a}, i, x)))\frown\\ & \qquad \mathsf{Enforce}_x(\mathsf{Bel}_x(E[\mathsf{key}(\mathsf{a}, i, x) : F]))\,\mathscr{B}\,\varepsilon\\ (\mathrm{AE4}) & \mathrm{pre}(x, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, y) : F]) =\\ & \quad \mathrm{pre}(x, F \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{u}, y)))\frown\\ & \qquad \mathsf{Enforce}_x(\mathsf{Bel}_x(E[\mathsf{key}(\mathsf{a}, \mathsf{u}, y) : F]))\,\mathscr{B}\,\varepsilon\end{array}$$

In case of asymmetric cryptography, the receiving agent is supposed to follow the principles of public key infrastructure:

$$\begin{array}{ll}(\mathrm{PE3}) & \mathrm{post}(y, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, y) : F]) =\\ & \quad (\mathsf{Bel}_y(\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{u}, y)) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{i}, y)))\\ & \qquad \mathscr{B}\ \mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{a}, i, x) :\\ & \qquad\quad E[\mathsf{key}(\mathsf{a}, u, x) : F]]))\,\mathscr{B}\,\varepsilon)\frown\mathrm{post}(y, F)\\ (\mathrm{PE4}) & \mathrm{post}(y, E[\mathsf{key}(\mathsf{a}, \mathsf{i}, z) : F]) =\\ & \quad (\mathsf{Bel}_y(\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{u}, y)))\,\mathscr{B}\\ & \qquad \mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{a}, \mathsf{u}, y) :\\ & \qquad\quad E[\mathsf{key}(\mathsf{a}, \mathsf{i}, y) : F]]))\,\mathscr{B}\,\varepsilon)\frown\mathrm{post}(y, F)\end{array}$$

Thus, whenever $y$ receives a message encrypted with $y$'s public key, $y$ should possess both its private and public key and therefore be able to decrypt the message according to the axiom.

**Observation 1** *All the previous results are maintained in case the functions* pre *and* post *are extended with the equations for asymmetric cryptography.*

## 4.3 Experimental results

A protocol-simulator written in Maude is the underlying basis of the project. The implementation includes totally 2600 lines of code. It includes rules for execution, and the algebra for protocol composition. We have refined several classical authentication protocols as given in Clark and Jacobs [2]. The table below shows three protocols that use symmetric cryptography, and one public key protocol, the Needham Schroeder Public Key Protocol. The symmetric key protocols include the Wide Mouthed Frog, Needham Schroeder Symmetric Key, and the Otway-Rees authentication protocol. The refinement functions can be extended with rules for asymmetric cryptography, the final row in the table show the results for Needham Schroeder Public Key.

| Protocol | $T^+$ | Automated refinement | | | | Simul ation |
|---|---|---|---|---|---|---|
| | | $\Re$ | | $\Re^\star$ | | |
| | *lh* | *lh* | rew | *lh* | rew | rew |
| WMF | 3 | 20 | 100 | 16 | 531 | 5693 |
| Need. sym. | 6 | 39 | 250 | 31 | 1594 | 11136 |
| OtwayRees | 5 | 47 | 295 | 29 | 1733 | 13656 |
| Need. pub. | 7 | 50 | 259 | 38 | 2188 | 10240 |

We let $T^+$ denote the extension of the textbook protocol to include explicit specification of generation of fresh keys. The results of the automated refinements are divided into core refinement $\Re$ and freshness refinement $\Re^\star$, where both the length (*lh*) and the number of rewrites in Maude (rew) are reported. The rightmost column gives the number of rewrites in a successful execution of the refined $\Re^\star$-protocols in the simulator. Each of the scenarious involved three agents "Alice", "Bob", and "Server", where each agent possesed the protocol in advance. Each agent in the scenario was able to construct fresh nonces and timestamps, and "Server" was in addition able to produce fresh keys.

The first three protocols are manually extended to enforce creation of fresh symmetric keys, therefore the length of these protocols ($T^+$) is extended with one compared to their conterparts in [2]. As expected, the core automated refinement $\Re$ gives significantly longer protocols, bounded by the maximal complexity of the message content in the original protocol. Since $\Re^\star$ removes superflous information, we always have $lth(\Re^\star(\mathsf{P})) \leqslant lth(\Re(\mathsf{P}))$.

## 5 Related work

State based techniques and model-checking ([6], [8]) have been used extensively the last decade as a paradigm for discovering possible attacks of protocols, since state machines closely model system behaviour. Epistemic logic and theorem proving techniques ([1], [10]) have been used both in attempts to verify security protocols and to precisely describe security properties, because the proof techniques are advanced and the languages involved are high level. Some tools like the protocol analyzer NRL [8], have been successful in representing many protocols and discovering several new attacks, while others like CAPSL [3] and Casper [7] have been used to specify protocols in a uniform way. The model that most closely resembles our approach is the strand-space apprach [12]. Some authors have investigated techniques to generate security protocols automatically by evolutionary methods ([11] and [5]).

## 6 Conclusion

We have shown how a straightforward sorted language for security can be used to assist in explaining authentication protocol semantics through automatic refinement of the specification syntax. This certainly speeds up the time it takes to understand and test both classical and new protocols. The automatized refinement proved to work gently for any protocol fed into the algorithm.

## References


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

[2] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature, 1997. Version 1.0.

[3] M. J. Denker G. and R. H. The CAPSL integrated protocol environment. Technical Report SRI-CLS-2000-02, SRI, 2000.

[4] A. M. Hagalisletto and J. Haugsand. A formal language for specifying security properties. In *Proceedings for the Workshop on Specification and Automated Processing of Security Requirements - SAPS'04*. Austrian Computer Society, 2004. book@acs.at.

[5] J. C. Hao Chen and J. Jacob. Automated Design of Security Protocols. *Computational Intelligence*, 20(3):503 – 516, 2004. Special Issue on Evolutionary Computing in Cryptography and Security.

[6] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147 – 166. Springer-Verlag, 1996.

[7] G. Lowe. Casper - a compiler for the analysis of security protocols. `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/`, 1998. SRI-CLS-2000-02.

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

[9] J. Meseguer and G. Rosu. Rewriting logic semantics: From language specifications to formal analysis tools. In D. A. Basin and M. Rusinowitch, editors, *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 1–44. Springer, 2004.

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

[11] A. Perrig and D. Song. A first step towards the automatic generation of security protocols. In *Network and Distributed System Security Symposium, NDSS '00*, pages 73–84, February 2000.

[12] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3), 1999.