Automated refinement of security protocols

Anders M. Hagalisletto

***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.

I. 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 [6] 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,
- soundness of manually refined security protocols can be formally verified.

The work presented in this paper is part of a large project, involving a specification language for security properties [9] and a simulator for executing protocols implemented in Maude [14]. The language and the simulator have a formal operational semantics [8]. 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 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 II, 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 III 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 IV 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 VI we compare our approach with related work, evaluate our approach, and point to some future research directions.

II. A language for protocols

Mike Burrows proposed the following authentication protocol, called the Wide Mouthed Frog, which we shall

[1]Evolutionary development of protocols from a given specification of security goals have been proposed (see [16] and [10]). These protocols are derived fully automated based on weight criterias.

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:

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

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_1^A, t_2^A, ... In $\mathscr{L}_S$ there are two distinguished atomic sentences; $\mathsf{Transmit}(a, b, m)$, $\mathsf{Agent}(a) \in$ 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) Atomic $\subset \mathscr{L}_S$.
(ii) If φ, $\psi \in \mathscr{L}_S$, 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 φ holds". The *until* operator $\varphi \mathcal{U} \psi$ means that φ holds until ψ 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* [9]. 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:

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

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

$$\begin{array}{ll}\mathrm{Trust}(a, b, \varphi) & \Longleftrightarrow \mathsf{Transmit}(b, a, \varphi) \to \mathsf{Bel}_a(\varphi) \\ & \qquad\qquad \text{Particular Trust} \\ \mathrm{Trust}(a, b) & \Longleftrightarrow \forall X\ \mathrm{Trust}(a, b, X) \\ & \qquad\qquad \text{Unconditional Trust}\end{array}$$

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* μ, μ_1, μ_2, *encryption methods* for cryptography s (symmetric) and a (asymmetric), in addition to the indicators i (private) and 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 μ 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 φ, $\xi^{\mathcal{T}}$, $\xi^{\mathcal{A}}$, $\xi^{\mathcal{S}} \in \mathscr{L}_P$, 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 μ". The sentences $E[k : \varphi]$ and $D[k : \varphi]$ denote the basic cryptographic primitives encryption and decryption respectively. The sentence $\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]$ reads "protocol named μ 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 Φ". The final sentence $\mathsf{Enforce}_a(\varphi)$ reads "enforce agent a to do the sentence φ" or "agent a does φ". 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 φ. 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*.

III. Specification of protocols

An event is the minimal unit in a protocol specification, for instance the transmission event or the empty event ε also called *skip*. A protocol is a chain of events between agents. A

$$
\begin{aligned}
&\mathsf{protocol}\ \ [\mathrm{WMF}_1,\quad 0,\quad \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z),\\
&\mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z),\quad \mathsf{role}(x),\\
&\\
&\mathsf{Transmit}(x, z, \mathsf{Agent}(x) \wedge E[\mathsf{key}(\mathsf{s}, x, z)\ :\\
&\quad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y))])\\
&\mathscr{B}\\
&\mathsf{Transmit}(z, y, E[\mathsf{key}(\mathsf{s}, y, z)\ :\ \mathsf{Time}(\mathsf{stamp}(u^T, z))\\
&\quad\ \wedge \mathsf{Agent}\ (x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y))])])]
\end{aligned}
$$

Fig. 1. The Wide Mouthed Frog.

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 φ_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 ε. Let $\Phi = \varphi \mathscr{B} [\Phi']$ denote a chain of events written by recursion, hence φ is a single event and ψ 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 φ and ψ be conjunctions of $\mathscr{P}$-*positive* sentences. Then φ is *included* in ψ, denoted $\varphi \sqsubseteq \psi$, iff either $\varphi = \top$, or

$\varphi = (\phi \wedge \Psi)$ and $\psi = (\phi \wedge \Psi')$ and $\Psi \sqsubseteq \Psi'$

$\varphi = (\phi \wedge \Psi)$ and $\psi = (\sigma \wedge \Psi')$ and $(\phi \wedge \Psi) \sqsubseteq \Psi'$

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) Φ 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 rather close to the way protocols are specified in research articles and textbooks. Note that variables are introduced, since an agent may play various roles in a single protocol.

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

A. Algebra and orderings of protocols

This section contains the sufficient fragment of definitions and results regarding ordering and concatenation of protocol syntax. More details and results can be found in the report [8]. 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 Φ and Ψ be final chains of events. Then Φ can be *embedded transitively* into Ψ, written $\Phi \sqsubseteq_{\mathrm{E}} \Psi$, if the following holds:

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

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

(iii) $\varphi \mathscr{B} [\Phi] \sqsubseteq_{\mathrm{E}} \psi \mathscr{B} [\Psi]$ 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 θ is a set of protocol names with $\mu_1, \mu_2 \in \theta$, then $\mu_1 \sqsubseteq_{\mathrm{N}} \mu_2$ means that μ_1 is a *subprotocol* of μ_2. The empty name is denoted ϵ. $\sqsubseteq_{\mathrm{N}}$ 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) $\mu_1 \sqsubseteq_{\mathrm{N}} \mu_2$ 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 μ_1 and μ_2 are protocol names, then $\mu_1\mu_2$ denotes their concatenation. For any set of protocol names Θ, concatenation is a monoid over Θ with unit ϵ.

Definition 8: Let Φ and Ψ be final chains of events. Then their *concatenation*, denoted $\Phi \frown \Psi$, is defined by:

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

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

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 P_1 and 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 $\mathsf{P}_2 = \mathsf{protocol}[\mu_2, N_2, \xi_2^{\mathcal{T}}, \xi_2^{\mathcal{A}}, \xi_2^{\mathcal{S}}, \Phi_2\,]$. Then the composition of P_1 with 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 $\mathcal{E}$.

Proof: To prove that $\langle \mathscr{P}, \boxplus\ \rangle$ is a *monoid* amounts to prove the following, which is left to the reader:

If 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: $\mathsf{P}_1 \sqsubseteq_{\mathrm{P}} \mathsf{P}_2 \wedge \mathsf{P}_3 \sqsubseteq_{\mathrm{P}} \mathsf{P}_4 \rightarrow \mathsf{P}_1 \boxplus \mathsf{P}_3 \sqsubseteq_{\mathrm{P}} \mathsf{P}_2 \boxplus \mathsf{P}_4$

Proof: By extensive applications of lemma 2. ■

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 P_s in a protocol P is a sub-protocol such that P_s is connected in P. For example, each of P_1, P_2 and 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 P_1 is *intrinsically connected* in a protocol P_2, written $\mathsf{P}_1 \sqsubseteq_{\mathrm{PI}} \mathsf{P}_2$, iff there exists protocols P' and P'', such that $\mathsf{P}_2 = \mathsf{P}' \boxplus \mathsf{P}_1 \boxplus \mathsf{P}''$.

Fortunately, protocol composition is *functional*:

Lemma 4: $\mathsf{P}_1 = \mathsf{P}_2 \implies \mathsf{P}' \boxplus \mathsf{P}_1 \boxplus \mathsf{P}'' = \mathsf{P}' \boxplus \mathsf{P}_2 \boxplus \mathsf{P}''$.
Proof: By lemma 1 and lemma 3. ■

IV. 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;

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

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

Many requirements about the execution of the Wide Mouthed Frog protocol are hidden in the textbook version. It is supposed that the initiator can generate fresh keys and timestamps, and is aware of its participants. Refining the protocol in figure 1, we obtain the assumption version of the protocol as seen in figure 2.

Observation 1: The textbook version of Wide Mouthed Frog is a sub-protocol of the assumption version in figure 2.
Observation 1 expresses that the assumption version of Wide Mouthed Frog Protocol is a refinement of its corresponding textbook protocol.

$$
\begin{array}{lr}
\mathsf{protocol}\ \ [\mathrm{WMF}_2,\ \ 0, & (H_1) \\
\quad \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z),\quad \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z),\quad \mathsf{role}(x) & (H_2) \\
\quad \mathsf{Bel}_x(\mathsf{Agent}(x) \wedge \mathsf{Agent}(y) \wedge \mathsf{Agent}(z) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z))) & (1 \circ) \\
\mathscr{B} & \\
\quad \mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{Current}(\mathsf{stamp}(w^T, x)))) & (2 \bullet) \\
\mathscr{B} & \\
\quad \mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{newKey}(\mathsf{key}(\mathsf{s}, x, y, v^K)))) & (3 \bullet) \\
\mathscr{B} & \\
\quad \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))])) & (4 \bullet) \\
\mathscr{B} & \\
\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))]) & (5 \rhd) \\
\mathscr{B} & \\
\quad \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))]))) & (6 \bullet) \\
\mathscr{B} & \\
\quad \mathsf{Enforce}_z(\mathsf{Bel}_z(D[\mathsf{key}(\mathsf{s}, x, z) : E[\mathsf{key}(\mathsf{s}, x, z) : & \\
\qquad\quad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]])) & (7 \bullet) \\
\mathscr{B} & \\
\quad \mathsf{Bel}_z(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z))) & (8 \circ) \\
\mathscr{B} & \\
\quad \mathsf{Enforce}_z(\mathsf{Bel}_z(\mathsf{Current}(\mathsf{stamp}(u^T, z)))) & (9 \bullet) \\
\mathscr{B} & \\
\quad \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))])) & (10 \bullet) \\
\mathscr{B} & \\
\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))]) & (11 \rhd) \\
\mathscr{B} & \\
\quad \mathsf{Enforce}_y(\mathsf{Bel}_y(\mathrm{Trust}(y, 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))])))] & (12 \bullet) \\
\mathscr{B} & \\
\quad \mathsf{Bel}_y(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) & (13 \circ) \\
\mathscr{B} & \\
\quad \mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{s}, y, z) : E[\mathsf{key}(\mathsf{s}, y, z) : & \\
\qquad\quad \mathsf{Time}(\mathsf{stamp}(u^T, z)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]]))\quad] & (14 \bullet)
\end{array}
$$

Fig. 2. Assumption version of WMF.

A. 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.

The message might contain both conjunctions of sentences and nested occurrences of encryption. Hence the refinement function generates a tree of assumptions. Since protocols are chains of events, the algorithm for automated refinement must *linearize* the refined tree. This can be done by the following algorithm:

Definition 12: If P is a textbook protocol, then 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 over 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 over the complexity of the message content:

(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) $\mathrm{post}(y, E[\mathsf{key}(\mathsf{s}, y, x)\ :\ F]) =$
$\mathrm{post}(y, \mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, x)))$
$\frown(\mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{s}, y, x)\ :$
$E[\mathsf{key}(\mathsf{s}, y, x)\ :\ F]]))\,\mathscr{B}\,\varepsilon) \frown \mathrm{post}(y, F)$

(PE2) $\mathrm{post}(y, E[\mathsf{key}(\mathsf{s}, x, z)\ :\ F]) = \varepsilon$ if $y \neq x \wedge y \neq z$

Let P be an arbitrary nonempty protocol with message contents $M = \{F_1, \ldots, F_{lth(\mathsf{P})}\}$. The maximal message content is a sentence $F \in M$ such that for every i with $1 \leqslant i \leqslant lth(\mathsf{P})$ we have that $\deg(F) \geqslant \deg(F_i)$. In other words, the maximal message content measures the most complex logical sentence in the protocol. Observe that conjunction, beliefs and encryption operators are treated equally. Then we can compute a bound for the maximal length of the automated refined protocol as a function of both the length and maximal content of the textbook protocol.

Lemma 5: Let P be a nonempty textbook protocol, where the maximal message content is F. Then we have:
$lth(\Re(\mathsf{P})) \leqslant lth(\mathsf{P}) \times (2^{\deg(F)} + \deg(F) + 2)$.

Proof: By induction over $\deg(F)$ for the textbook protocols P with fixed length $lth(\mathsf{P})$. ∎

Lemma 6: If $lth(\mathsf{P}') = 1$ then $\Re(\mathsf{P}') \boxplus \Re(\mathsf{P}) = \Re(\mathsf{P}' \boxplus \mathsf{P})$.

Proof: Let $\mathsf{P} = \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]$, and let $\mathsf{P}' = \mathsf{protocol}[\mu', N', \xi^{\mathcal{T}}_{\prime}, \xi^{\mathcal{A}}_{\prime}, \xi^{\mathcal{S}}_{\prime}, \mathsf{Transmit}(x, y, F)\,\mathscr{B}\,\varepsilon\,]$, since $lth(\mathsf{P}') = 1$. Then by definition of $\boxplus$:

$$\begin{aligned}&\Re(\mathsf{P}' \boxplus \mathsf{P}) =\\ &\mathsf{protocol}[\mu'\mu, N' + N, \xi^{\mathcal{T}}_{\prime} \wedge \xi^{\mathcal{T}}, \xi^{\mathcal{A}}_{\prime} \wedge \xi^{\mathcal{A}}, \xi^{\mathcal{S}}_{\prime} \wedge \xi^{\mathcal{S}},\\ &\qquad \Re((\mathsf{Transmit}(x, y, F)\,\mathscr{B}\,\varepsilon) \frown \Phi)\,].\end{aligned} \tag{1}$$

From definition 12, and definition 8, we can prove that

$$\begin{aligned}&\Re((\mathsf{Transmit}(x, y, F)\,\mathscr{B}\,\varepsilon) \frown \Phi)\\ &\quad = \Re(\mathsf{Transmit}(x, y, F)\,\mathscr{B}\,\varepsilon) \frown \Re(\Phi).\end{aligned} \tag{2}$$

Substituting (2) into (1) gives

$$\begin{aligned}&\mathsf{protocol}[\mu'\mu, N' + N, \xi^{\mathcal{T}}_{\prime} \wedge \xi^{\mathcal{T}}, \xi^{\mathcal{A}}_{\prime} \wedge \xi^{\mathcal{A}}, \xi^{\mathcal{S}}_{\prime} \wedge \xi^{\mathcal{S}},\\ &\qquad \Re(\mathsf{Transmit}(x, y, F)\,\mathscr{B}\,\varepsilon) \frown \Re(\Phi)\,]\\ &= \Re(\mathsf{P}') \boxplus \Re(\mathsf{P}),\end{aligned}$$
which ends the proof.

∎

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

Lemma 7: $\Re(\mathsf{P}_1) \boxplus \Re(\mathsf{P}_2) = \Re(\mathsf{P}_1 \boxplus \mathsf{P}_2)$

Proof: By induction on $lth(\mathsf{P}_1)$. The basis is okay since;

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

(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 $lth(\mathsf{P}') = 1$:

$$\begin{aligned}&\Re(\mathsf{P}' \boxplus \mathsf{P}) \boxplus \Re(\mathsf{P}_2) && \text{lemma 4 and 6}\\ &= (\Re(\mathsf{P}') \boxplus \Re(\mathsf{P})) \boxplus \Re(\mathsf{P}_2) && \text{theorem 2, } \mathscr{P}\text{AS}\\ &= \Re(\mathsf{P}') \boxplus (\Re(\mathsf{P}) \boxplus \Re(\mathsf{P}_2)) && \text{ind. hyp. and lemma 4}\\ &= \Re(\mathsf{P}') \boxplus \Re(\mathsf{P} \boxplus \mathsf{P}_2) && \text{lemma 6}\\ &= \Re(\mathsf{P}' \boxplus (\mathsf{P} \boxplus \mathsf{P}_2)) && \text{theorem 2, } \mathscr{P}\text{AS}\\ &= \Re((\mathsf{P}' \boxplus \mathsf{P}) \boxplus \mathsf{P}_2), && \text{which is what we wanted.}\end{aligned}$$

∎

Lemma 8: If P is a textbook protocol containing only one message transmission, then $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re(\mathsf{P})$.

Proof: Follows directly from the definition (AR-2) of automated refinement $\Re$. ∎

Theorem 3: If P is a textbook protocol, then $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re(\mathsf{P})$.

Proof: By induction over $lth(\mathsf{P})$. The basis is obvious using (AR-1). Consider the induction step: Suppose that $lth(\mathsf{P}) = k$ 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 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 $lth(\mathsf{P}') = 1$. 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 (lemma 7). ■

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.

Lemma 9: Any textbook protocol 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 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 lemma 9 we assure that the agent's nonces and timestamps are explicitly constructed in the specification. If P is a textbook protocol then let $\Re^\star$ denote the function constructed in the proof of lemma 9, that is: $\Re^\star(\mathsf{P}) = \star(\mathrm{eq}(\Re(\mathsf{P})))$. Figure 3 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 3, since lemma 7 does not generalize to freshness refinement $\Re^\star$. Fortunately, we can prove a couple of useful properties

Lemma 10: Let P_1 and 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 $\mathrm{eq}(\Re^\star(\mathsf{P}_1)) = \Re^\star(\mathsf{P}_1)$

Proof: In case of (i), an event φ might occur both in $\Re^\star(\mathsf{P}_1)$ and $\Re^\star(\mathsf{P}_2)$, hence we have $\Re^\star(\mathsf{P}_1 \boxplus \mathsf{P}_2) \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P}_1) \boxplus \Re^\star(\mathsf{P}_2)$. The only way $\Re^\star(\mathsf{P}_1 \boxplus \mathsf{P}_2)$ and $\Re^\star(\mathsf{P}_1) \boxplus \Re^\star(\mathsf{P}_2)$ might differ concerns multiple occurences of $\mathsf{Enforce}_x(\mathsf{Bel}_x(\psi))$. Hence without violating the previous result, eq might be extended to contract multiple occurences of

$$\begin{array}{l}
\mathsf{protocol}\ \ [\mathrm{WMF}_3,\quad 0,\\
\qquad \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z), \qquad \mathsf{role}(x) \wedge \mathsf{role}(y) \wedge \mathsf{role}(z), \qquad \mathsf{role}(x)
\end{array}$$

$$\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{newKey}(\mathsf{key}(\mathsf{s}, x, y, v^K)))) \tag{1}$$
$$\mathscr{B}$$
$$\mathsf{Bel}_x(\mathsf{Agent}(x)) \tag{2}$$
$$\mathscr{B}$$
$$\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{Current}(\mathsf{stamp}(w^T, x)))) \tag{3}$$
$$\mathscr{B}$$
$$\mathsf{Bel}_x(\mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K)) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z))) \tag{4}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Enforce}_x(\mathsf{Bel}_x(E[\mathsf{key}(\mathsf{s}, x, z) :\\ \ \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))])) \end{array} \tag{5}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Transmit}(x, z, \mathsf{Agent}(x) \wedge E[\mathsf{key}(\mathsf{s}, x, z) :\\ \ \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]) \end{array} \tag{6}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Enforce}_z(\mathsf{Bel}_z(\mathrm{Trust}(z, x, E[\mathsf{key}(\mathsf{s}, x, z) :\\ \ \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]))) \end{array} \tag{7}$$
$$\mathscr{B}$$
$$\mathsf{Bel}_z(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z))) \tag{8}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Enforce}_z(\mathsf{Bel}_z(D[\mathsf{key}(\mathsf{s}, x, z) : E[\mathsf{key}(\mathsf{s}, x, z) :\\ \qquad \mathsf{Time}(\mathsf{stamp}(w^T, x)) \wedge \mathsf{Agent}(y) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]])) \end{array} \tag{9}$$
$$\mathscr{B}$$
$$\mathsf{Enforce}_z(\mathsf{Bel}_z(\mathsf{Current}(\mathsf{stamp}(u^T, z)))) \tag{10}$$
$$\mathscr{B}$$
$$\mathsf{Bel}_z(\mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K)) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) \tag{11}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Enforce}_z(\mathsf{Bel}_z(E[\mathsf{key}(\mathsf{s}, y, z) :\\ \ \mathsf{Time}(\mathsf{stamp}(u^T, z)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))])) \end{array} \tag{12}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Transmit}(z, y, E[\mathsf{key}(\mathsf{s}, y, z) :\\ \ \mathsf{Time}(\mathsf{stamp}(u^T, z)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]) \end{array} \tag{13}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Enforce}_y(\mathsf{Bel}_y(\mathrm{Trust}(y, z, E[\mathsf{key}(\mathsf{s}, y, z) :\\ \ \mathsf{Time}(\mathsf{stamp}(u^T, x)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y))])))] \end{array} \tag{14}$$
$$\mathscr{B}$$
$$\mathsf{Bel}_y(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) \tag{15}$$
$$\mathscr{B}$$
$$\begin{array}{l}\mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{s}, y, z) : E[\mathsf{key}(\mathsf{s}, y, z) :\\ \qquad \mathsf{Time}(\mathsf{stamp}(u^T, x)) \wedge \mathsf{Agent}(x) \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, v^K))]]))\quad] \end{array} \tag{16}$$

Fig. 3. Automated refinement of WMF textbook protocol.

enforcement statements as well. Then $\mathrm{eq}(\Re^\star(\mathsf{P}_1) \boxplus \Re^\star(\mathsf{P}_2))$ removes the extra occurences.

Consider (ii): First $\mathrm{eq}(\mathsf{P}_1) = \mathsf{P}_1$ follows since P_1 is textbook. Second $\mathrm{eq}(\Re^\star(\mathsf{P}_1)) = \mathrm{eq}(\star(\mathrm{eq}(\mathsf{P}_1))) = \star(\mathrm{eq}(\mathsf{P}_1))$, since the latter contains only one occurence of each event. ■

Lemma 11: If P is a textbook protocol, with $lth(\mathsf{P}) = 1$, then we have that $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P})$.

Proof: By lemma 8, $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re(\mathsf{P})$, yet $\mathsf{P} \sqsubseteq_{\mathrm{P}} \mathrm{eq}(\Re(\mathsf{P}))$, and $\mathsf{P} \sqsubseteq_{\mathrm{P}} \star(\mathrm{eq}(\Re(\mathsf{P})))$ since the single transmission in P is preserved by both functions. ■

Theorem 4: If P is a textbook protocol, then $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P})$.

Proof: By induction on $lth(\mathsf{P})$. Ind. basis is verified by $\Re^\star(\varepsilon) = \star(\mathrm{eq}(\Re(\varepsilon))) = \varepsilon$. Consider the induction step: Analogous to theorem 3 we consider one-step 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 11, $\mathsf{P}'' \sqsubseteq_{\mathrm{P}} \Re^\star(\mathsf{P}'')$. 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 10 (i), $\mathrm{eq}(\mathsf{P}' \boxplus \mathsf{P}'') = \mathsf{P}' \boxplus \mathsf{P}''$ (2), since P is a textbook protocol. By lemma 10 (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;

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

■

B. 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:

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

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:

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

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

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

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 2: All the previous results are maintained in case the functions pre and post are extended with the equations for asymmetric cryptography.

C. Automated or manual specifications?

The specification in figure 2 was written by hand. Since we know that fresh keys must be constructed explicitly (hence manually), $\Re$ is extended to handle construction of fresh keys as given in the key-protocol:

$$\begin{array}{l}\Re(\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{newKey}(k))) \mathscr{B}\, [\Phi]) = \\ \quad \mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{newKey}(k))) \mathscr{B}\, [\Re(\Phi)]\end{array}$$

When we applied the automated refinement algorithm discussed previously, $\Re^\star(\mathrm{WMF}_2)$, we get the specification depicted in figure 3. But how do we know that this expresses the same protocol? Compared to figure 2, there are some differences. Yet, the two protocols are equal in a very strong sense, which indicates that there should be a way of identifying "WMF_3" and "WMF_2". The similarity of the two protocol specifications can be established by considering only the specification syntax. Consequently we borrow the concept *structural congruence* from the π calculus, to identify similarity between specifications solely based on syntax. If φ is a subsentence of Φ at position p in Φ, then we write $\Phi_p\{\varphi\}$. The boolean statement " φ is a subsentence of Φ" is written $\varphi \in_{\mathrm{S}} \Phi$.

Definition 15: Structural congruence of expressions in $\mathscr{L}_P$, denoted $\equiv$ is an equivalence relation defined by:

(1) If $\varphi \in_{\mathrm{S}} \Phi$ and $\varphi \equiv \psi$ then $\Phi_p\{\varphi\} \equiv \Phi_p\{\psi\}$
(2) If $\mathsf{P}_1 = \mathsf{protocol}[\mu_1, N_1, \xi_1^{\mathcal{T}}, \xi_1^{\mathcal{A}}, \xi_1^{\mathcal{S}}, \Phi_1]$ and $\mathsf{P}_2 = \mathsf{protocol}[\mu_2, N_2, \xi_2^{\mathcal{T}}, \xi_2^{\mathcal{A}}, \xi_2^{\mathcal{S}}, \Phi_2]$ then $\mathsf{P}_1 \equiv \mathsf{P}_2$ iff either P_1 and P_2 are α-equal (†) or $\xi_1^{\mathcal{T}} \equiv \xi_2^{\mathcal{T}},\ \xi_1^{\mathcal{A}} \equiv \xi_2^{\mathcal{A}},\ \xi_1^{\mathcal{S}} \equiv \xi_2^{\mathcal{S}}$, and $\Phi_1 \equiv \Phi_2$ (‡)
(3) $\varphi \wedge \varphi \equiv \varphi,\ \ \varphi_1 \wedge \varphi_2 \equiv \varphi_2 \wedge \varphi_1,$ $(\varphi_1 \wedge \varphi_2) \wedge \varphi_3 \equiv \varphi_1 \wedge (\varphi_2 \wedge \varphi_3)$
(4) $\mathsf{Bel}_x(\varphi) \mathscr{B}\ \mathsf{Bel}_x(\psi) \equiv \mathsf{Bel}_x(\varphi \wedge \psi)$
(5) $\mathsf{Enforce}_x(\mathsf{Bel}_x(\varphi)) \mathscr{B}\ \mathsf{Bel}_x(\varphi) \equiv \mathsf{Enforce}_x(\mathsf{Bel}_x(\varphi))$
(6) $\mathsf{Bel}_x(\varphi) \mathscr{B}\ \mathsf{Bel}_y(\psi) \equiv \mathsf{Bel}_y(\psi) \mathscr{B}\ \mathsf{Bel}_x(\varphi)$ if $x \neq y$
(7) $\mathsf{Bel}_x(\varphi) \mathscr{B}\ \mathsf{Enforce}_y(\psi) \equiv \mathsf{Enforce}_y(\psi) \mathscr{B}\ \mathsf{Bel}_x(\varphi)$ if $x \neq y$ or $\varphi \notin_{\mathrm{S}} \psi$
(8) $\mathsf{Bel}_x(\varphi) \mathscr{B}\ \mathsf{Transmit}(y, z, \psi) \equiv$ $\mathsf{Transmit}(y, z, \psi) \mathscr{B}\ \mathsf{Bel}_x(\varphi)$ if $x \neq y$ or $\varphi \notin_{\mathrm{S}} \psi$

The rules for structural congruence can be justified as follows: Structural congruence of subexpressions migrate to the context they inhabit (1). Note that structural congruence is independent of the name and the session number. If the roles and the protocol body are structurally congruent, then the protocols are congruent (2†). The concrete variables chosen is not a critical issue for protocols (2‡). In order to ensure that matching work properly, conjunctions like $\phi \wedge \psi$ are interpreted as "ϕ and if ψ", yet (3) ensures that the laws of idempotence, commutativity, and associativity are covered by $\equiv$. A successive chain of beliefs by an agent is congruent with the single belief of their conjunction (4). When the agent has enforced a belief it is superfluous to claim that the agent possess this belief (5). The final congruences (6), (7) and (8) show that beliefs might be moved syntactically under the condition that the movement does not have any impact of local protocol region. The belief can not be moved if the subject of each event is the same agent and the content of the belief is a subsentence of the other event.

Observation 3: $\mathrm{WMF}_2 \equiv \mathrm{WMF}_3$

A useful notion derivable from structural congruence is the notion of *sound assumption protocol*. Let $\mathrm{text}(\mathsf{P})$ denote the textbook filtration of the protocol P. Then we say that an assumption protocol P is a *sound assumption protocol* if $\mathsf{P} \equiv \Re^\star(\mathrm{text}(\mathsf{P}))$. Then we can address the previous question:

Observation 4: WMF_2 is a sound assumption protocol.

D. Experimental results

A protocol-simulator written in Maude is the underlying basis of the project. We have refined several classical authentication protocols as given in Clark and Jacobs[6]. 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 $\Re$		$\Re^\star$		Simul ation
	lh	lh	rew	lh	rew	rew
WMF	3	20	100	16	531	5693
Need. sym.	7	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 generation of fresh keys. The results of the automated refinements are divided into standard refinement $\Re$ and $\Re^\star$, 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.

V. Related work

Although formal specification and analysis of protocols have been investigated by many authors the last two decades, the algebra of protocol syntax and refinement have been overlooked ([2], [1]). State based techniques and model-checking ([11], [13]) 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 ([5], [4] [3]) and theorem proving techniques [5], [15]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 [13], have been successful in representing many protocols and discovering several new attacks, while others like CAPSL [7] and Casper [12] have been used to specify protocols in a uniform way. The conceptual model that most closely resembles our approach is the strand-space apprach [17]. In [18], the authors investigate the distinction of local and global protocols, similar to our concepts. Some authors have investigated techniques to generate security protocols automatically by evolutionary methods ([16] and [10]). An interesting approach using BAN logics [10] proposes techniques to automatically generate protocols. The latter approach is devoted to meta-heuristic techniques for searching for the best protocols given a set of goals, following the BAN rules.

VI. 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 paper explore how a mixture of logics and algebra can broaden our practical and theoretical understanding of security protocols, in order to provide tool support for analysing protocols.

References

[1] *Symposium on Security and Privacy*, Denver, USA, 2001-2005. IEEE Computer Society Press.

[2] *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)The 1st International Workshop on Security in Systems and Networks*. IEEE Computer Society, April 2005.

[3] Pierre Bieber and Frederic Cuppens. Expressions of Confidentiality Policies with Deontic Logic. In *Deontic Logic in Computer Science: Normative System Specification*, pages 103–123. John Wiley and Sons Ltd, 1993.

[4] A. Bleeker and J. van Eijck. The epistemics of encryption. Technical Report Report INS-R0019, 2000. CWI 2000.

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

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

[7] Millen J. Denker G. and Ruess H. The CAPSL integrated protocol environment. Technical Report SRI-CLS-2000-02, SRI, 2000.

[8] Anders Moen Hagalisletto. A framework for simulating and analyzing security protocols. Technical report, University of Oslo, 2005.

[9] Anders Moen Hagalisletto and Jon 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.

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

[11] Gavin 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.

[12] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

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

[14] José Meseguer and Grigore Rosu. Rewriting logic semantics: From language specifications to formal analysis tools. In David A. Basin and Michaël Rusinowitch, editors, *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 1–44. Springer, 2004.

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

[16] Adrian Perrig and Dawn 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.

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

[18] L. Yilmaz and J. A. Hamilton. Modular compositional validation of protocol conflicts for network interoperability. In *Proceedings of the Design, Analysis, and Simulation of Distributed Systems 2004*, pages 149–155, 2004.