A Logic for Constraint-based Security Protocol Analysis

Ricardo Corin
University of Twente, The Netherlands

Ari Saptawijaya*
University of Indonesia, Indonesia

Sandro Etalle
University of Twente, The Netherlands

Abstract

We propose $\mathcal{PS}$-LTL, a pure-past security linear temporal logic that allows the specification of a variety of authentication, secrecy and data freshness properties. Furthermore, we present a sound and complete decision procedure to establish the validity of security properties for symbolic execution traces, and show the integration with constraint-based analysis techniques.

1 Introduction

The communication via a shared medium, like the Internet, is inherently insecure: anyone has access to en route messages and can potentially eavesdrop or even manipulate the ongoing communication. Security protocols are distributed programs specifically designed to achieve secure communication over such media, typically exchanging messages constructed using cryptographic operations.

Security protocols are difficult to design correctly, hence their analysis is critical. A successful analysis model is the Dolev Yao model [17], in which the attacker has complete network control and ideal cryptography is assumed. The model is attractive because it has an appropriate level of abstraction, as many attacks are independent of the underlying details of the cryptographic operations. Moreover, the model can be easily formalized using languages and tools based on formal methods (e.g. [5, 26, 24, 23, 15]). For the case of analysis with a bounded number of protocol instances (and also allowing the attacker to use composed keys), the security problem is known to be decidable [29]. In this setting, constraint solving approaches (originally presented by Millen and Shmatikov [28] and later extended and improved [10, 1, 13, 21]) are efficient and effective, since instead of considering all possible traces (i.e. protocol executions) only evaluate a finite number of symbolic traces in a lazy fashion.

Unfortunately, constraint-based verification systems are relatively weak when it comes to specifying the properties one wants to check. Typically, for each security property an ad-hoc recipe is given, which amounts to modifying the protocol instance with particular tests that capture violations of the property under analysis. For example, to check authentication [28], one has to craft a protocol instance in which some participant is present but not its corresponding party, and observe whether the participant can still finish its run. This is coarse-grained and cumbersome to implement. (Later, Millen [27] implemented a fixed, built-in notion of authentication in his Prolog verifier.) Checking secrecy is also done ad-hoc and in a restrictive manner, by adding an artificial protocol role which expects a secret no other participant would send.

In this paper we remedy this situation by proposing a dedicated language to specify security properties. The language is expressive and flexible, and allows the specification of complex properties in a clean and intuitive manner, fairly separated from the protocol being analyzed. More specifically, our contribution is twofold:

- First, we develop $\mathcal{PS}$-LTL, a language to specify security properties based on linear temporal logic (LTL) with pure-past operators. As we shall see, our language provides adequate flexibility, allowing one to specify several security properties like authentication ([23, 12]) (including *aliveness*, *weak agreement* and *non-injective agreement*), secrecy (*standard secrecy* [2] and *perfect forward secrecy* [16]) and also data freshness [9]. We also present a preliminary study of denial of service (DoS) [25] within our language.

- While the semantics of $\mathcal{PS}$-LTL is defined as usual on concrete (variable-free) traces, constraint-based protocol verifiers generate symbolic traces which contain *constrained* variables (i.e. variables which may be instantiated only with values the attacker can compute). We present a decision procedure which allows to check

*Part of this work was done while the author was affiliated to University of Twente.

a relevant subset (that covers all the properties of interest) on the symbolic traces produced by constraint solving systems. Moreover, we show the soundness and completeness of our decision procedure. Finally, we implement the procedure using Prolog into our protocol verification tool thereby providing a full verification system (an online demo is available[11]) where we verify several protocols (at present, more than twenty protocols from the Clark and Jacob library [6]).

To the best of our knowledge, in the context of constraint-based methods our proposal is the first language for the specification of security properties which is equipped with a sound and complete procedure for evaluating formulas against a symbolic trace. Moreover, besides the theoretical value of our results we believe that our contribution has also practical significance, as protocol designers can use our approach to systematically engineer and debug protocols during the early design phases.

Paper Structure We review the protocol model and the constraint solving technique in Section 2. $\mathcal{PS}$-LTL is presented in Section 3, along several example properties. Our decision procedure is presented in Section 4. Section 5 discusses related work and finally Section 6 concludes the paper. Example scenarios and proofs may be found in [8].

2 Protocol Model and Constraint Solving

We introduce some notation for the rest of the paper: first the term algebra and intruder rules, then the protocol model and finally constraint solving as introduced in [28].

2.1 Preliminaries

Term algebra and substitutions Messages are represented as terms in a free algebra $\mathcal{T}$ generated by the operators in Table 1 (left), from an enumerable set of variables $\mathcal{V}$ (denoted by uppercase letters $A, B, Na, K, \dots$), and an enumerable set of constants $\mathcal{C}$ (denoted by lowercase $a, b, na, k, \dots$), representing the principal identities, nonces and keys. (We do not consider different *types*, although it would be straightforward to do so.) A special constant $e \in \mathcal{C}$ is distinguished to denote the intruder's identity. We have constructors for representing public keys, pairing, message hashing, symmetric and asymmetric encryption, and signature. Private keys in asymmetric encryption are not modelled in the term algebra since we assume that these keys are never part of messages in protocols. (This assumption is realistic, as every protocol we consider does not send its private key in messages.) For readability, we simply write $(t_1, t_2, \dots, t_{n-1}, t_n)$ to denote multiple pairing $(t_1, (t_2, \dots, (t_{n-1}, t_n) \dots))$.

For $i \leq j$, the integer interval $\{i, i+1, \dots, j-1, j\}$ is denoted as $[i \dots j]$. The set of *ground* terms, denoted by $\mathcal{T}^+$, is generated like $\mathcal{T}$ above but only from constants $\mathcal{C}$ and excluding $\mathcal{V}$. When $t \in \mathcal{T}^+$, we say that t is *ground*, otherwise it is *non-ground*. The variables of a term t are denoted as $var(t)$.

Substitutions (denoted by $\sigma, \rho, \gamma, \dots$) are finite mappings from $\mathcal{V}$ to $\mathcal{T}$. Ground substitutions map $\mathcal{V}$ to $\mathcal{T}^+$. The domain of σ is denoted as $dom(\sigma)$. The empty substitution is denoted as ε. Given $v \in \mathcal{V}$ and $t \in \mathcal{T}$, $[{}^t/v]$ denotes the singleton substitution mapping of v to t. Given a term $t \in \mathcal{T}$ and a substitution σ, $t\sigma$ denotes the term resulting from substituting each occurrence of $v \in dom(\sigma)$ in t by $\sigma(v)$. A term t' is an *instance* of another term t if there exists σ s.t. $t' = t\sigma$. The same terminology is used for the (later introduced) events, protocol roles and traces.

Intruder model Rules are used to represent the abilities of the intruder. Let A be a set of terms and t a term, and let ℓ be a rule label, stating the name of the rule. A *rule* is denoted by $A \rightarrow_\ell t$. We work with the set of rules given in Table 1 (right), where t_1 and t_2 are terms in $\mathcal{T}$. As usual, the attacker is allowed to pair and split terms, hash, symmetrically encrypt terms with any (possibly non-atomic) key and decrypt symmetrically if the key is known. Public-key encryption ($penc$) is modelled by allowing to encrypt with any key. However, rule *pdec* only allows the asymmetric decryption of a term encrypted with the attacker's public key. The attacker cannot decrypt any term encrypted with a different public key than his own, since we assume that private keys are never leaked (as they do not take part of any message). Moreover, the attacker can only sign terms using his private key, represented in rule *sig* (here $pk(e)$ is the public key that is needed to verify the signature).

We now define $\mathcal{F}(T)$ (the *fake* operation), representing the terms the intruder can generate from the set of ground terms T:

Definition 1. *Let T be a set of ground terms, i.e. $T \subseteq \mathcal{T}^+$. Then, $\mathcal{F}(T)$ is defined as $\cup_{n \geq 0} \mathcal{F}^n(T)$, where $\mathcal{F}^n(T)$ is the set defined inductively as follows:*

$$\begin{array}{lcl} \mathcal{F}^0(T) & = & T \\ \mathcal{F}^n(T) & = & \mathcal{F}^{n-1}(T) \cup \{t \mid A \rightarrow_\ell t \textit{ is a DY rule and} \\ & & A \subseteq \mathcal{F}^{n-1}(T)\} \end{array}$$

Intuitively, when the attacker knows (e.g. has eavesdropped) a set of messages T, then he can compute the set of terms $\mathcal{F}(T)$.

2.2 Protocol Model

Our protocol model is related to the strand-space formalism [31], although we sometimes use a different terminology, e.g. we call *system scenario* what in strand spaces is

$t_1, t_2 ::=$					
	c	constant in $\mathcal{C}$	$\{t_1, t_2\}$	$\rightarrow_{pair}$	(t_1, t_2)
	v	variable in $\mathcal{V}$	$\{(t_1, t_2)\}$	$\rightarrow_{first}$	t_1
	$pk(t_1)$	public key	$\{(t_1, t_2)\}$	$\rightarrow_{second}$	t_2
	(t_1, t_2)	pair	$\{t\}$	$\rightarrow_{hash}$	$h(t)$
	$h(t_1)$	hash	$\{t_1, t_2\}$	$\rightarrow_{senc}$	$\{t_1\}_{t_2}$
	$\{t_1\}_{t_2}$	symmetric encryption	$\{\{t_1\}_{t_2}, t_2\}$	$\rightarrow_{sdec}$	t_1
	$\{t_1\}_{\overrightarrow{t_2}}$	asymmetric encryption	$\{t_1, t_2\}$	$\rightarrow_{penc}$	$\{t_1\}_{\overrightarrow{t_2}}$
	$sig_{t_1}(t_2)$	signature	$\{\{t_1\}_{\overrightarrow{pk(e)}}\}$	$\rightarrow_{pdec}$	t_1
			$\{t_1\}$	$\rightarrow_{sig}$	$sig_{pk(e)}(t_1)$

Table 1. Grammar for terms (left) and DY rules (right)

called a *semibundle*. In the following, we introduce events, traces, protocol roles, and system scenarios.

Definition 2. *An* event *is one of the following:*

- *A* communication *event is a pair $\langle a : m \diamond b\rangle$ where a, b are variables or principal constants, $\diamond \in \{\triangleleft, \triangleright\}$ and m is a term. a is called the* active *party, and b is the* passive *party.*

  Since we will let the attacker intercept and forge communication messages, the event $\langle a : m \triangleright b\rangle$ reads as "principal a sends message m with intended destination *b". Symmetrically, $\langle a : m \triangleleft b\rangle$ stands for "principal a receives message m* apparently from *b".*

- *A* status *event: $p(d_1, \cdots, d_n)$, with d_i a term for $i \in [1 \ldots n]$ and p is a function symbol.*

  We consider three different, self-explanatory status events: start, run *and* end *(see Example 4).*

Definition 3. *A* protocol role *is a finite sequence of events in which all events share the same active principal.*

Given a protocol written in standard '$a \rightarrow b : m$' notation, it is straightforward to obtain its *parametric* protocol roles[1], as shown in the next example.

Example 4. *Consider the BAN Concrete Andrew Secure RPC protocol [5], that aims to achieve mutual authentication of two agents a and b which share a long term key k_{lt}. (The last message is stripped out, since it is not necessary for security.)*

$$\begin{aligned} 1.\ a \rightarrow b &: (a, n_a) \\ 2.\ b \rightarrow a &: \{(n_a, k_{st})\}_{k_{lt}} \\ 3.\ a \rightarrow b &: \{n_a\}_{k_{st}} \end{aligned}$$

First a sends a message with her identity and a fresh nonce n_a. Upon receipt, b generates a short term session key k_{st}, encrypts it along with a's nonce n_a using the long term key k_{lt}, shared previously with a. Finally, a replies with her nonce n_a encrypted with the newly established key k_{st}. In the following, we name protocol roles such as init, *and* resp, *denoting an initiator and a responder respectively. The parametric protocol roles are as follows (We show the status events in bold typeface).*

$$\begin{aligned} &\text{init}(A, B, N_A, K_{lt}, K_{st}) = \\ &\quad\langle \quad \mathbf{start(A, B, initiator)} \\ &\qquad \langle A : (A, N_A) \triangleright B\rangle \\ &\qquad \langle A : \{(N_A, K_{st})\}_{K_{lt}} \triangleleft B\rangle \\ &\qquad \mathbf{run(A, B, initiator, N_A, K_{lt}, K_{st})} \\ &\qquad \langle A : \{N_A\}_{K_{st}} \triangleright B\rangle \\ &\qquad \mathbf{end(A, B, initiator, N_A, K_{lt}, K_{st})} \\ &\quad\rangle \end{aligned}$$

$$\begin{aligned} &\text{resp}(A, B, N_A, K_{lt}, K_{st}) = \\ &\quad\langle \quad \mathbf{start(B, A, responder)} \\ &\qquad \langle B : (A, N_A) \triangleleft A\rangle \\ &\qquad \mathbf{run(B, A, responder, N_A, K_{lt}, K_{st})} \\ &\qquad \langle B : \{(N_A, K_{st})\}_{K_{lt}} \triangleright A\rangle \\ &\qquad \langle B : \{N_A\}_{K_{st}} \triangleleft A\rangle \\ &\qquad \mathbf{end(B, A, responder, N_A, K_{lt}, K_{st})} \\ &\quad\rangle \end{aligned}$$

While start and end status events are located in the obvious places, the position where run status events are located is more subtle. We locate these events as soon as the protocol role has received every piece of data relevant to the protocol run (this becomes relevant in Section 3.2).

The next step consists of (partially) instantiating the parametric roles, i.e. gathering several protocol roles together providing a particular system instance to be analysed.

Definition 5. *A* system scenario *is a multiset of protocol roles.*

A system scenario determines which sessions are present, and which agents play which roles.

Example 6. *Consider the following simple system scenario,*

[1]A parametric role is the most general role; it can be later instantiated in particular scenarios.

where init *and* resp *are the roles of Example 4:*

$$Sc_0 = \{\text{init}(a, B, n_a, k_{lt}, K_{st}), \text{resp}(A, b, N_A, k_{lt}, k_{st})\}$$

The initiator is played by a, using fresh nonce na and shared key k_{lt}, while the responder is b, using the shared key k_{lt} and the (freshly created) session key k_{st}.

Intuitively, this scenario is crafted so that, if the protocol is secure, then the unknown participants A and B can only be played by a and b, respectively, as they are the only participants that know their shared key k_{lt}.

2.3 Trace Validity

A *trace* is a sequence of events. Traces can be obtained from system scenarios by interleaving events from the protocol roles, and possibly instantiating variables. Formally, ν is an *interleaving* of Sc if $\nu \in ||Sc$, for $||Sc$ defined as: $||Sc = \cup_{\langle a\ r\rangle \in Sc}\ a.(||(Sc \setminus \{\langle a\ r\rangle\} \cup \{r\})$, with $s.Sc = \{\langle s\ r\rangle \mid r \in Sc\}$. A *prefix* interleaving is a prefix of a complete interleaving.

Definition 7. *We say that a trace tr* derives *from scenario Sc if there exists an instance Sc' of Sc s.t. tr is a prefix interleaving of Sc'.*

Appending an event ev to trace tr is written $\langle tr\ ev\rangle$. Functions $last$ and $length$ have the usual meaning: $last(\langle tr\ ev\rangle) = ev$ ($last$ is undefined for the empty trace), $length(\langle\rangle) = 0$ and $length(\langle tr\ ev\rangle) = length(tr) + 1$. The prefix trace consisting of the first i events is denoted as tr_i, with $tr_0 = \langle\rangle$ and $tr_m = tr$ for $m \geq length(tr)$.

The *initial intruder knowledge* set, denoted *IK*, is a ground set of terms representing what the intruder knows before starting the analysis of a specific scenario. This set includes the intruder identity e, and may include other public information, like principal identities or public keys.

When the protocol is executed in presence of the intruder, we apply the Dolev Yao model: (a) every message sent by an honest principal is added to the intruder's knowledge, and (b) every message received by an honest principal is produced by the intruder using the knowledge accumulated until that point. Formally, after the events in tr have taken place, the knowledge of the intruder is equal to $IK \cup K(tr)$, with $K(tr)$ defined as follows.

Definition 8. *The (intruder gathered)* knowledge *of a trace tr is given by $K(tr) = \{m \mid last(tr_i) = \langle a : m \triangleright b\rangle, i \in [1 \ldots length(tr)]\}$.*

Suppose we have a ground trace $tr = \langle tr'\ ev\rangle$, with $ev = \langle a : m \triangleleft b\rangle$. We say that the event ev in tr is *valid* if the intruder could produce m using $IK \cup K(tr')$. A whole trace is valid when all its receive communication events are valid, as shown in the next definition.

Definition 9. *A ground trace tr is* valid *w.r.t. IK if for each $i \in [0 \ldots length(tr) - 1]$, $last(tr_{i+1}) = \langle a : m \triangleleft b\rangle$ implies that $m \in \mathcal{F}(K(tr_i) \cup IK)$.*

2.4 Constraint Solving

Central to this paper is the notion of *constraint* and *constraint set*, as defined below.

Definition 10. *A* constraint *is a pair $m : K$, of a term m and a term set K (standing for* knowledge*). $m : K$ is* simple *if m is a variable. A* constraint set *CS is a finite set of constraints; CS is simple if each constraint in the set is simple.*

A *solution* of a constraint set is a substitution that makes every constraint to be *solvable*. We formalize this notion in the next definition, using the fake operator $\mathcal{F}(\cdot)$ introduced in Definition 1.

Definition 11. *We say that σ is a* solution *of the constraint $m : K$ if $m\sigma$ and $K\sigma$ are ground and $m\sigma \in \mathcal{F}(K\sigma)$. $m : K$ is* solvable *iff it has a solution. Also, γ is a* partial solution *of $m : K$ iff $m\gamma : K\gamma$ is solvable. Finally, a constraint set is solvable iff each of its constraints is solvable.*

Millen and Shmatikov's reduction algorithm (called $\mathbf{P}$ in the following) [28] maps a constraint set CS to (a possibly empty) set of pairs of simple constraints CS' and substitutions γ. We do not explicitly define $\mathbf{P}$ here but rather use it as a black box (since our extensions do not concern its details), relying on $\mathbf{P}$'s properties:

Theorem 12 ([28][2]). *(a) $\mathbf{P}$ always* terminates. *(b)* Soundness: *If $\mathbf{P}$ applied to CS outputs (CS', γ), then γ is a partial solution of CS, and every solution of CS' is also a solution of $CS\gamma$. (c)* Completeness: *If CS is solvable with solution σ, then applying $\mathbf{P}$ to CS returns some (CS', γ) such that, for some solution σ' of CS', $\sigma = \gamma\sigma'$.*

We now describe an algorithm (sketched in [10]) which given a system scenario Sc_0 and an initial intruder knowledge *IK* non-deterministically produces a set of traces. This procedure differs from the original given in [28] in that the scenario is directly executed by incrementally adding events during an execution and checking that the constraint set remains solvable. This ensures that unsolvable interleavings are never considered and thus results in a significant efficiency gain.

Procedure 13. *A* state *is a 4-tuple $\langle Sc, IK, CS, tr\rangle$, where Sc is a system scenario, IK is the initial intruder knowledge, CS is a simple constraint set and tr is a (possibly non-ground) trace. An* execution step *from state $\langle Sc, IK, CS, tr\rangle$ to $\langle Sc', IK, CS', tr'\rangle$ is obtained by performing:*

[2] We actually reformulated the result [28] using the terminology of partial solutions as given in Definition 11.

1. *Choose non-deterministically a non-empty role $r \in Sc$. Let $r = \langle ev\ r' \rangle$. Consider the following cases for ev:*

   (a) *If ev is a* send *communication event or a status event, let γ be the empty substitution and CS'' be CS.*

   (b) *If ev is a* receive *communication event, i.e. $ev = \langle a : m \lhd b \rangle$, check that the intruder can generate m using the knowledge $K(tr) \cup IK$, by applying procedure $\mathbf{P}$ to $CS \cup \{m : (K(tr) \cup IK)\}$, obtaining a new simple constraint set CS'' and a partial solution γ (Note that there may be many possible CS'' and γ).*

2. *Let $Sc' := (Sc \setminus \{r\} \cup \{r'\})\gamma$, $CS' := CS''$ and $tr' := \langle\ tr\gamma\ ev\gamma\ \rangle$.*

A run for Sc_0 with *IK is a sequence of execution steps starting from state $\langle Sc_0, IK, \emptyset, \langle\rangle \rangle$.*

Every (and only!) valid traces (as defined above in Definition 9) should be output in states of Procedure 13. The formal statement of this result, along with its proof, appears in [8, Theorem 2.3.6].

3 $\mathcal{PS}$-LTL

We first introduce the syntax and semantics of $\mathcal{PS}$-LTL and then specify security properties with the language.

3.1 Syntax and Semantics

The syntax of $\mathcal{PS}$-LTL is defined as follows.

Definition 14. *$\mathcal{PS}$-LTL formulas are generated by the following grammar:*

$$\begin{aligned}\phi ::= \quad & \texttt{true} \mid \texttt{false} \mid p(d_1, \ldots, d_n) \mid \texttt{learn}(m) \\ & \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \texttt{Y}\phi \mid \phi\texttt{S}\phi \mid \exists v.\phi \mid \forall v.\phi\end{aligned}$$

where each d_i ($i \in [1 \ldots n]$) and m is either a variable in $\mathcal{V}$ or a ground term in $\mathcal{T}^+$.

Standard formulas $\texttt{true}$, $\texttt{false}$, $\neg\phi$, $\phi \wedge \phi$, $\phi \vee \phi$ carry the usual meaning. The formula $p(d_1, \ldots, d_n)$ is a status event. (For simplicity and w.l.o.g. we allow d_is to be only constant terms or variables.) $\texttt{learn}(m)$ is a predicate stating that the intruder knows term m (we borrow the name from NPATRL [30]). $\texttt{Y}\phi$ means '*yesterday* ϕ held', while $\phi_1\texttt{S}\phi_2$ means that 'ϕ_1 held *ever since* a moment in which ϕ_2 held'. When $v \in \mathcal{V}$, we write $\exists v.\phi$ and $\forall v.\phi$ to bind v in ϕ, with the quantifiers carrying the usual meaning with v ranging over terms. Other operators can be represented using the above defined operators: $\phi_1 \rightarrow \phi_2$ is defined as $\neg\phi_1 \vee \phi_2$; $\texttt{O}\phi$ (*once* ϕ) is a shorthand for $\texttt{true}\ \texttt{S}\ \phi$ and finally $\texttt{H}\phi$ (*historically* ϕ) is a shorthand for $\neg\texttt{O}\neg\phi$. For clarity, we impose a precedence hierarchy for operators, where unary operators bind stronger than binary operators. Operators $\texttt{Y}$, $\texttt{O}$, and $\texttt{H}$ bind equally strong and bind stronger than $\neg$. The precedence hierarchy for binary operators is $\texttt{S} > \wedge > \vee > \rightarrow$, where $op_1 > op_2$ means "op_1 binds stronger than op_2". In the sequel, we assume that $\mathcal{PS}$-LTL formulas are *closed* (i.e. they contain no free variables), and that each variable is quantified at most once. Also, we assume that the variables occurring in a formula ϕ are disjoint from the variables occurring in execution traces tr from the considered system scenario (this can always be achieved by alpha conversion).

Our semantics $\langle tr, IK \rangle \models \phi$ is defined for two different cases: First, we define it when tr is a ground trace, which we call *concrete* validity. Later (in Definition 17), we extend the semantics to the general case, in which tr contains variables. This establishes *symbolic* validity. Given a trace tr, we recall that tr_i denotes its prefix trace consisting of the first i events.

Definition 15 (Concrete validity)**.** *Let ϕ be a closed $\mathcal{PS}$-LTL formula, tr be a ground trace and IK be an initial intruder knowledge. Then, $\langle tr, IK \rangle \models \phi$ is defined in Table 2.*

It is easy to see that the semantics of $\texttt{O}$ and $\texttt{H}$ coincide with the intuitive ones, i.e. that $\langle tr, \mathrm{IK} \rangle \models \texttt{O}\phi$ iff $\exists i \in [0, length(tr)] : \langle tr_i, \mathrm{IK} \rangle \models \phi$, and $\langle tr, \mathrm{IK} \rangle \models \texttt{H}\phi$ iff $\forall i \in [0, length(tr)] : \langle tr_i, \mathrm{IK} \rangle \models \phi$. In fact, we can also state and prove other standard results for LTL and infinite traces, like the tautology $\texttt{H}\phi \rightarrow \texttt{O}\phi$. Furthermore, we can state some particular relations of $\mathcal{PS}$-LTL, like the following proposition which intuitively shows that the intruder never forgets information.

Proposition 16. *For every ground trace tr, initial intruder knowledge IK and message m:*

(i) $\langle tr, IK \rangle \models \texttt{learn}(m)$ *iff* $\langle tr, IK \rangle \models \texttt{O}\ \texttt{learn}(m)$ *iff* $\langle tr, IK \rangle \models \texttt{learn}(m)\ \texttt{S}\ \texttt{learn}(m)$*; and*

(ii) $\langle tr, IK \rangle \models \texttt{Y}\ \texttt{learn}(m)$ *implies* $\langle tr, IK \rangle \models \texttt{learn}(m)$.

Proof. Straightforward from unfolding Definition 15 and the monotonicity of $\mathcal{F}(\cdot)$, i.e. $\mathcal{F}(K(tr_i) \cup IK) \subseteq \mathcal{F}(K(tr_j) \cup IK)$ for each $i \leq j$. $\square$

3.2 Writing Security Properties with $\mathcal{PS}$-LTL

In this section we show how to specify several security properties in $\mathcal{PS}$-LTL for the BAN Concrete Andrew Secure RPC protocol [5], shown in Example 4. In addition to this protocol we also have successfully used our

$\langle tr, \mathrm{IK}\rangle \models \mathtt{true}$		
$\langle tr, \mathrm{IK}\rangle \not\models \mathtt{false}$		
$\langle tr, \mathrm{IK}\rangle \models p(d_1, \ldots, d_n)$	iff	$tr = \langle tr'\ q(e_1, \ldots, e_m)\rangle$ *and* $p(d_1, \ldots, d_n) = q(e_1, \ldots, e_m)$
$\langle tr, \mathrm{IK}\rangle \models \mathtt{learn}(m)$	iff	$m \in \mathcal{F}(K(tr) \cup \mathrm{IK})$
$\langle tr, \mathrm{IK}\rangle \models \neg\varphi$	iff	$\langle tr, \mathrm{IK}\rangle \not\models \varphi$
$\langle tr, \mathrm{IK}\rangle \models \exists v.\varphi$	iff	$\exists t \in \mathcal{T}^+ : \langle tr, \mathrm{IK}\rangle \models \varphi[^t/_v]$
$\langle tr, \mathrm{IK}\rangle \models \forall v.\varphi$	iff	$\forall t \in \mathcal{T}^+ : \langle tr, \mathrm{IK}\rangle \models \varphi[^t/_v]$
$\langle tr, \mathrm{IK}\rangle \models \varphi_1 \wedge \varphi_2$	iff	$\langle tr, \mathrm{IK}\rangle \models \varphi_1$ and $\langle tr, \mathrm{IK}\rangle \models \varphi_2$
$\langle tr, \mathrm{IK}\rangle \models \varphi_1 \vee \varphi_2$	iff	$\langle tr, \mathrm{IK}\rangle \models \varphi_1$ or $\langle tr, \mathrm{IK}\rangle \models \varphi_2$
$\langle tr, \mathrm{IK}\rangle \models Y\varphi$	iff	$tr = \langle tr'\ ev\rangle$ and $\langle tr', \mathrm{IK}\rangle \models \varphi$
$\langle tr, \mathrm{IK}\rangle \models \varphi_1 S \varphi_2$	iff	$\exists i \in [0, length(tr)] : (\langle tr_i, \mathrm{IK}\rangle \models \varphi_2 \wedge\ \forall j \in [i+1, length(tr)] : \langle tr_j, \mathrm{IK}\rangle \models \varphi_1)$

Table 2. Concrete validity $\langle tr, IK\rangle \models \phi$

tool properties for several other protocols, see [11] (we already analysed over twenty protocols from the Clark Jacob library [7]).

3.2.1 Authentication

First we specify various forms of authentication as defined in [23]. We cover all the variants except injective agreement, which would require counting events in a trace. (In principle, we could extend our system to cover injective agreement, which would result on the ability to detect some replay attacks on which injective agreement is violated but non-injective agreement is satisfied.)

We detail the case of authentication of the initiator to a responder, and do not show here the similar converse case.

Aliveness The aliveness property is the weakest form of authentication in Lowe's hierarchy:

A protocol guarantees to a responder B aliveness *of another principal A if, whenever B (acting as responder) completes a run of the protocol, apparently with initiator A, then A has previously been running the protocol.*

Notice that A may have run the protocol with a principal other than B. The aliveness of principal A to responder B is shown in $\mathcal{PS}$-LTL in Table 3 (1).

Caveat. As usual, we prevent the attacker e from recording events in a protocol execution. However, as the attacker may be involved in legitimate executions, there may be "fake" attacks in which an honest participant talks to the attacker e, who behaves honestly. For example, it could be that in some scenario an initiator a talks to e and issues the event $end(a, e, ...)$, but there is no corresponding start event (hence violating aliveness). We currently ignore such "fake" attacks, as we also do with unrealistic type-flaw attacks [20]. However, it would be easy to extend our language to prevent such fake attacks from arising, by adding an atomic predicate $honest(X)$, which holds when X is not e. Then, for example, we would write the aliveness property as $\forall A, B, \ldots .\ \exists B', R'.\ (end(B, A, responder, \ldots) \wedge honest(A)) \rightarrow \circ\ start(A, B', R')$

Aliveness is violated for our protocol of Example 4 on a scenario containing at least two protocol sessions (i.e. two initiators and two responders). We run our tool on a suitable scenario and found a similar attack to the one found by Lowe [22]. Furthermore, we also checked the aliveness property for Lowe's fixed version of BAN concrete Andrew Secure RPC protocol [22], and found no attacks, confirming the validity of Lowe's fix.

Weak Agreement Weak agreement is slightly stronger than aliveness:

A protocol guarantees to a responder B weak agreement *with another principal A if, whenever B (acting as responder) completes a run of the protocol, apparently with initiator A, then A has previously been running the protocol,* ***apparently with*** *B.*

For this property, A may not necessarily have been acting as initiator. We show this property expressed in $\mathcal{PS}$-LTL in Table 3 (2). Since weak agreement is stronger than aliveness, the attack mentioned above also applies.

Non-injective Agreement Non-injective agreement is slightly stronger than weak agreement:

A protocol guarantees to a responder B non-injective agreement *with another principal A on a set of data items* $\mathbf{D}$ *if, whenever B (acting as responder) completes a run of the protocol, apparently with initiator A, then A has previously been running the protocol, apparently with B, A was acting as initiator in his run, and the two principals agreed on the data values corresponding to all the variables in* $\mathbf{D}$.

The property is formalized in $\mathcal{PS}$-LTL in Table 3 (3) (Our tool also discovers the attack against this property).

1. Aliveness: $\forall A, B, D1, D2, D3.\ \exists B', R'.\ end(B, A, responder, D1, D2, D3) \rightarrow \mathtt{O}\ start(A, B', R')$
2. Weak agreement: $\forall A, B, D1, D2, D3.\ \exists R'.\ end(B, A, responder, D1, D2, D3) \rightarrow \mathtt{O}\ start(A, B, R')$
3. Non-injective agreement:
   $\forall A, B, D1, D2, D3.\ end(B, A, responder, D1, D2, D3) \rightarrow \mathtt{O}\ run(A, B, initiator, D1, D2, D3)$
4. Perfect forward secrecy:
   $\forall A, B, N, K_{lt}.\ learn(K_{lt}) \wedge\ \mathtt{Y}(\mathtt{O}\ (end(B, A, responder, N, K_{lt}, k_{st}) \wedge \mathtt{H}\ \neg learn(k_{st}))) \rightarrow \mathtt{H}\ \neg learn(k_{st})$
5. Perfect forward secrecy (more efficient form):
   $\forall A, B, N, K_{lt}.\ learn(K_{lt}) \wedge\ \mathtt{Y}(\mathtt{O}\ (end(B, A, responder, N, K_{lt}, k_{st}) \wedge\ \neg learn(k_{st}))) \rightarrow\ \neg learn(k_{st})$
6. Freshness:
   $\forall A, B_1, R_1, N_1, K_1, K, B_2, R_2, N_2, K_2.\ \mathtt{Y}(\mathtt{O}\ end(A, B_1, R_1, N_1, K_1, K)) \rightarrow \neg end(A, B_2, R_2, N_2, K_2, K)$
7. Denial of service: $\forall A, B, D1, D2, D3.\ run(B, A, responder, D1, D2, D3) \rightarrow \mathtt{O}\ start(A, B, initiator).$

Table 3. $\mathcal{PS}$-LTL properties

3.2.2 Secrecy

We focus on standard secrecy and perfect forward secrecy.

Standard secrecy We define first the simple case of standard secrecy, which is the inability of an attacker to obtain the value of the secret [2]. Recall scenario Sc_0 of Example 6. The secrecy of the session key k_{st} can be checked by the $\mathcal{PS}$-LTL formula $\neg learn(k_{st})$, which does not find any secrecy attack on Sc_0.

Perfect Forward Secrecy We now consider perfect forward secrecy (PFS), as defined by Diffie, et al. [16]:

An (authenticated key exchange) protocol provides perfect forward secrecy if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs.

In Diffie et.al [16], the proposed Authenticated Diffie-Hellman key exchange protocol is shown to preserve PFS, since long term keys are only used to sign messages and are never related to the session key derivation. This is not the case for the RPC Andrew protocol and its variants, since the short term session key is directly encrypted by the long term key (see below for an example of a secure protocol).

The disclosure of long term secret keying material, e.g. k_{lt}, can be realized by providing an additional protocol role, which contains only one send event that leaks k_{lt} to the intruder [3]. The specification of PFS in $\mathcal{PS}$-LTL needs to express that: (i) the leaking of the long term key k_{lt} happens *after* a protocol run has been completed, and at that time (ii) the short term session key (k_{st} in our example) was secret but (iii) after the leaking of the long term key, the short term key is learnt by the attacker. Our treatment of this property is similar to Delicata and Schneider [14]. In Table 3 (4) we express PFS; thanks to Proposition 16, we can rewrite the property in a more efficient form, as shown in Table 3 (5). Our tool finds the straightforward attack quickly in an appropriate scenario.

Discussion. In the definition of Diffie et al. there is a division in time between the "present" (i.e. when the long-term key is disclosed) and "past" (i.e. when the earlier runs took place). In our specification, that moment in time is given by the moment in which a session finishes succesfully, and an end event is recorded in the trace. However, as an anonymous reviewer points out, it could be that a session did not end succesfully, so it remains stuck without an end event being recorded (due to some interaction with the attacker). Then, the disclosure of the long-term key allows the learning of a session key that would not have been possible in the normal run. It would certainly be interesting to extend in the future our formulation to cover this case.

A secure protocol w.r.t. PFS Consider the protocol:

$$\begin{array}{lcl} 1.\ a \rightarrow b & : & \{(pk(r_a), b)\}_{k_{lt}} \\ 2.\ b \rightarrow a & : & (\{k_{st}\}^{\rightarrow}_{pk(r_a)}, \{(h(k_{st}), a)\}_{k_{lt}}) \end{array}$$

This protocol is a modified version of the protocol due to Boyd and Mathuria [4] that aims to meet perfect forward secrecy.[4] Agents a and b share a long term key k_{lt}. Agent a generates a *fresh* asymmetric key pair in every protocol

[3]We could easily extend the learn predicate to give extra information to the attacker, so we do not need to modify a scenario. Thus we would have $\mathtt{learn}(m, K)$ with semantics $\langle tr, IK\rangle \models \mathtt{learn}(m, K)$ iff $m \in \mathcal{F}(IK \cup K(tr) \cup K)$.

[4]Note that the protocol provides only one-way authentication, that is authentication of b to a.

run (indicated by a *fresh* r_a) and *discards* it after the run is completed (thus the key pair is *ephemeral*). In the first message, a encrypts the public part $pk(r_a)$ together with b's identity with k_{lt} and sends it to b. Upon receipt, b obtains $pk(r_a)$ and then replies by encrypting the freshly generated short term session key k_{st} with $pk(r_a)$ and encrypting the hash of k_{st} and a's identity with k_{lt}.

Although the disclosure of k_{lt} after a completed protocol run allows an attacker to impersonate a or b in the subsequent runs, it does not provide the attacker with the ability to recover the session key k_{st} from the previous run. This session key can only be recovered using a's private key from the completed run (which has been discarded as soon as the protocol run completes).

Assume that when the initiator ends it emits the event $end(A, B, initiator, R_A, K_{lt}, K_{st})$. We verified a scenario in which the short term key chosen by the responder is k_{st}, and our tool found no attack, thus confirming the analysis of a similar protocol by Delicata and Schneider [14].

3.2.3 Beyond Authentication and Secrecy

We now consider the novel specification of some further properties. First we develop the property of data freshness, that states that particular session data (e.g. a session key) should be always fresh in a session. Then we consider the study of a primitive notion of denial of service.

Data Freshness We state the data freshness property as follows:

Data D is fresh whenever a principal A never completes a run with another principal agreeing on D, if once in the past A has already completed a protocol run with another principal agreeing on the same data D.

The freshness $\mathcal{PS}$-LTL property of a short term session key K for the protocol of Example 4 is shown in Table 3 (6).

We run our tool to check the freshness of the session key k_{st} and obtained an attack similar to the previous aliveness attack. In this attack, the session key k_{st} is used twice, i.e. when a was acting as an initiator in one session and as a responder in the other session. Thus, it violates the freshness of k_{st}. For Lowe's fixed version of BAN concrete Andrew Secure RPC protocol [22], no attack was found as expected.

Towards analysing Denial of Service attacks We now sketch finally the specification of a property to analyse potential vulnerabilities regarding denial of service (DoS) attacks [25] (We plan to study this notion further as future work.)

In the protocol of Example 4, the first message (a, n_a) can be generated cheaply by anyone (not necessarily by a). Moreover, upon receiving this message b commits to perform several expensive operations (e.g. generating a session key and allocating state for the running session).

To mount a DoS attack against b, an attacker needs to start several sessions in which b reaches its *run* event (since at that point b commits to finishing the protocol, see Example 4), but there is no corresponding effort done by a (so no one is committed to finish any session with b). Note that the attacker does not need to achieve that b finishes his execution (and emitting the *end* status event); the attacker needs only that b emits its *run* event. Thus this property differs from the authentication ones given above which always require b to emit its *end* event.

We model this indication of DoS attack in $\mathcal{PS}$-LTL, by specifying that if a responder b runs a session apparently with a, then the honest initiator a has once started a session with b, as shown in Table 3 (7).

We successfully use our tool to check this formula on a single session and obtain a trace that indicates a potential DoS attack. When we modify the protocol by encrypting the first message using the long term key k_{lt} (i.e. message 1 is $\{(a, n_a)\}_{k_{lt}}$) we found no attack. Intuitively, this encryption allows only *honest* participants, who share k_{lt}, to generate the first message (still, the attacker could replay message 1 and continue mounting the DoS attack; this would be detected by an injective analysis, as discussed in the beginning of Section 3.2.1).

4 Deciding $\mathcal{PS}$-LTL in Constraint Solving

The constraint-based Procedure 13 presented in Section 2.4 outputs *symbolic* traces containing constrained variables. In this section we show how to decide validity of a $\mathcal{PS}$-LTL formula against such a symbolic trace. Since in the previous section we defined validity (called concrete) only w.r.t. ground traces, the first thing we need to do is to extend the notion of validity for symbolic traces.

Definition 17 (Symbolic validity)**.** *Given a trace tr derived from a system scenario Sc and the initial intruder knowledge IK, we say that $\langle tr, IK\rangle \models \phi$ when for every valid instance tr' of tr, $\langle tr', IK\rangle \models \phi$.*

Let φ be a closed $\mathcal{PS}$-LTL formula representing a security property. We let $A_\varphi = \neg\varphi$ be its corresponding *attack* property. Given a symbolic trace tr and the initial intruder knowledge IK, we now define a procedure $\mathbf{D}$ that tries to find a valid ground instance tr' of tr s.t. $\langle tr', IK\rangle \models A_\varphi$. If $\mathbf{D}$ succeeds, tr' represents a violation of φ (hence an attack), since $\langle tr', IK\rangle \models A_\varphi$ iff $\langle tr', IK\rangle \not\models \varphi$, and thus $\langle tr, IK\rangle \not\models \varphi$. On the other hand, if $\mathbf{D}$ fails, then we know that there is no tr' s.t. $\langle tr', IK\rangle \models A_\varphi$. In other words, for every ground instance tr' of tr, $\langle tr', IK\rangle \models \varphi$, i.e. $\langle tr, IK\rangle \models \varphi$. Thus $\mathbf{D}$ decides symbolic validity.

Our approach consists of two stages. We first translate a closed $\mathcal{PS}$-LTL formula ϕ into a (shown equivalent) *elementary formula* EF, using the transformation $\mathbf{T}$ described next in Section 4.2. Then, we input the translated formula to the decision procedure $\mathbf{D}$ presented in Section 4.3. Procedure $\mathbf{D}$ uses the ability to solve *negated constraints*, so we start by defining these negated constraints and then presenting a strategy to solve them.

4.1 Solving Negated Constraints

In Section 2.4 we consider "positive" constraints $m : T$, since its solution decides whether there exists a substitution σ s.t. $m\sigma$ can be built from $T\sigma$. In the following we consider *negated* constraints, whose solution decides whether there exists a substitution σ s.t. $m\sigma$ is *not* derivable from $T\sigma$.

Definition 18. *A* negated *constraint is denoted by $\neg(m : T)$, where m is a term and T is a set of terms. σ is a* solution *of $\neg(m : T)$ if $m\sigma \notin \mathcal{F}(T\sigma)$, in which case we say that $\neg(m : T)$ is solvable.*

If both m and T are ground, then procedure $\mathbf{P}$ (see Theorem 12) can be used to solve $\neg(m : T)$:

Corollary 19. *Let m be a ground term and let T be a set of ground terms. Then $\neg(m : T)$ is solvable iff $\mathbf{P}$ applied to $m : T$ fails.*

Proof. By Theorem 12, $\mathbf{P}$ fails iff for all σ, $m\sigma \notin \mathcal{F}(T\sigma)$. Since $m : T$ is ground, we obtain that $\mathbf{P}$ fails iff $m \notin \mathcal{F}(T)$, establishing the property. $\square$

When $\mathbf{P}$ succeeds, we know that there exists a substitution σ s.t. $m\sigma \in \mathcal{F}(T\sigma)$. So if $\mathbf{P}$ fails, we have that *for all* substitutions σ, $m\sigma \notin \mathcal{F}(T\sigma)$; However, what we are trying to establish is whether there *exists* a substitution σ s.t. $m\sigma \notin \mathcal{F}(T\sigma)$. In the case that $m : T$ is ground, then the two cases collapse and hence we can use Corollary 19. However, when m or T is non ground, we cannot use $\mathbf{P}$ straightforwardly.

Example 20. *Consider the negated constraint $\neg(\{X\}_Y : \{\{secret_1\}_{secret_2}, e\})$. Applying procedure $\mathbf{P}$ to $\{X\}_Y : \{\{secret_1\}_{secret_2}, e\}$ succeeds, assigning $secret_1$ to X and $secret_2$ to Y. However, the negated constraint is solvable, e.g. by assigning e to X and $secret_2$ to Y.*

Given a state $\langle Sc, IK, CS, tr\rangle$ from Procedure 13 for a run of input scenario Sc_0 and *IK* and given a negated constraint $\neg(m : K(tr') \cup IK)$[5] for some term m and some tr' prefix trace of tr, we are interested on finding a solution σ of both CS and $\neg(m : K(tr') \cup IK)$.

[5]For readability we only consider one negated constraint; the extension to the general case is straightforward.

We now present a simple strategy to solve $CS \cup \neg(m : K(tr') \cup IK)$ when $K(tr')$ is possibly non ground, although m has to be ground. This solution is enough for our current purposes, as all our security properties are covered; a solution for the general case is still a matter of current research.

Initially, we include a set of fresh constants to the attacker knowledge, one for each variable occurring in the input scenario Sc_0. More formally, we assume that the initial intruder knowledge *IK* includes a set of constants $C_V = \{c_X \mid X \in \mathcal{V}$ and X occurs in $Sc_0\}$ (recall that $\mathcal{V}$ is the set of variables) thus $C_V \subseteq IK$. C_V contains *intruder generated constants* which do not occur in the input scenario, and hence are never needed to to solve the positive constraint solving phase of CS (the use of these constants is inspired by the work of Kähler and Küsters [21]).

Let σ_V be the substitution that maps every variable X to the corresponding constant c_X. Our solving strategy consists on checking whether σ_V is a solution of $CS \cup \neg(m : K(tr') \cup IK)$. Intuitively, using a fresh constant for each variable gives the best chances for the negated constraint $\neg(m : (K(tr') \cup IK)\sigma_V)$ to hold; any other arbitrary solution σ could map variables to terms which are more "related" to each other than the (completely unrelated) fresh constants used by σ_V, thus giving more chances that the attacker can derive m from $(K(tr') \cup IK)\sigma$. This result is formalized below, where $T \not\trianglelefteq S$ means that no term $t \in T$ occurs in any term $s \in S$ and $T \not\trianglelefteq s$ means that no term $t \in T$ occurs in s.

Theorem 21. *Let $\langle Sc, IK, CS, tr\rangle$ be a state from Procedure 13 where for each $X : T_X \in CS$, $C_V \subseteq T_X$ and $C_V \not\trianglelefteq T_X \setminus C_V$. Let tr' be a prefix of tr and $\neg(m : K(tr') \cup IK)$ be a negated constraint,* where m is ground and *$C_V \not\trianglelefteq m$. There exists a substitution σ solution of both CS and $\neg(m : K(tr') \cup IK)$ iff σ_V is a solution of both CS and $\neg(m : K(tr') \cup IK)$.*

The result relies on a series of lemmas, as reported in [8, Appendix A.2]. Below we briefly describe the proof (for the non trivial case ($\Rightarrow$)).

It is easy to see that σ_V is a solution of CS. We then focus on establishing that σ_V is a solution of $\neg(m : K(tr') \cup \text{IK})$, that is, $m \notin \mathcal{F}(K(tr')\sigma_V \cup \text{IK})$. Recall that we assume $C_V \subseteq IK$ are intruder generated constants, one per variable V occurring in the input scenario, with the substitution σ_V mapping each V to c_V. We consider a similar set of constants D_V, with similar substitution ρ_V. Let $\delta = \{d_X \to c_X \mid X \in \mathcal{V}\}$.

We first see that if $m \notin \mathcal{F}((K(tr') \cup IK)\sigma)$ then $m \notin \mathcal{F}_\delta((K(tr') \cup IK)\rho_V)\sigma'$, for σ' the mapping that replaces each occurrence of d_X with $X\sigma$ and $\mathcal{F}_\delta$ like $\mathcal{F}$ with the added rule $\{\{t_1\}_{t_2}, t_2'\} \rightarrow_{cdec} t_1$ when $t_2\delta = t_2'\delta$. This follows from the fact that $\mathcal{F}_\delta((K(tr') \cup IK)\rho_V)\sigma'$ (the set that results from replacing each variable X to d_X, clausuring it with $\mathcal{F}_\delta$ and then remapping back each d_X to $X\sigma$) is actually included in the original set $\mathcal{F}((K(tr') \cup \mathit{IK})\sigma)$. Next, we show that $m \notin \mathcal{F}_\delta((K(tr') \cup \mathit{IK})\rho_V)\sigma'$ implies that $m \notin \mathcal{F}_\delta((K(tr') \cup \mathit{IK})\rho_V)\delta$, i.e. if m is not in the former set the m is not in the same set which differs only in that in the last step replaces d_Xs to c_Xs (the δ mapping) instead of applying σ'. Then we establish that $m \notin \mathcal{F}((K(tr') \cup \mathit{IK})\rho_V\delta)$, (that is, if we replace d_Xs to c_Xs inside the closure by δ then we can "substitute" $\mathcal{F}_\delta$ by $\mathcal{F}$). Finally, we get the theorem $m \notin \mathcal{F}((K(tr') \cup \mathit{IK})\sigma_V)$, since applying ρ_V and δ is equivalent to applying σ_V.

Now, the problem of deciding whether a negated constraint $\neg(m : K)$ is solvable (where m is ground) is solved by Theorem 21, which tells us that $\neg(m : K)$ is solvable iff $m\sigma_V : K\sigma_V$ is *not* solvable, something that can be easily checked using $\mathbf{P}$ as described in Corollary 19. As we see in the next, the ability to solve negated constraints is one important ingredient to decide validity of $\mathcal{PS}$-LTL formulas in Section 4.3.

4.2 Translating $\mathcal{PS}$-LTL

The first step in solving $\mathcal{PS}$-LTL consists on applying a transformation to simpler formulas called *elementary formulas*. We first introduce elementary formulas, whose syntax is given by the following definition:

Definition 22. *Elementary formulas EF (ranged over by π) are defined by the grammar:*

$$\pi ::= \texttt{true} \mid \texttt{false} \mid t_1 = t_2 \mid m : K \mid \neg\pi \mid \pi \wedge \pi \mid \pi \vee \pi \mid \exists v.\pi \mid \forall v.\pi$$

Here each t_1, t_2 and m is either a variable or a ground term, K is a set of terms and v is a variable.

Let π be an EF formula. We define the *left* free variables $free_l(\pi)$ and its *right* free variables $free_r(\pi)$, as follows:

$$\begin{array}{lcl}
free_l(\texttt{true}) = free_l(\texttt{false}) & = & \emptyset \\
free_l(t_1 = t_2) & = & var(t_1) \\
free_l(m : K) & = & var(m) \\
free_l(\neg\pi) & = & free_l(\pi) \\
free_l(\pi_1 \wedge \pi_2) = & = & free_l(\pi_1) \cup free_l(\pi_2) \\
free_l(\pi_1 \vee \pi_2) & = & free_l(\pi_1) \cup free_l(\pi_2) \\
free_l(\exists v.\pi) = free_l(\forall v.\pi) & = & free_l(\pi) \setminus \{v\}
\end{array}$$

$free_r(\pi)$ is similar, but defined with: $free_r(t_1 = t_2) = var(t_2)$ and $free_r(m : K) = var(K)$.

We now give semantics of an EF formula π w.r.t. a ground substitution σ.

Definition 23. *Let π be an EF formula and σ be a ground substitution s.t. $free_l(\pi) = \emptyset$ and $free_r(\pi) = dom(\sigma)$. Then $\sigma \models' \pi$ is defined by:*

$$\begin{array}{lll}
\sigma \models' \texttt{true} & & \\
\sigma \not\models' \texttt{false} & & \\
\sigma \models' t_1 = t_2 & \textit{iff} & t_1 = t_2\sigma \\
\sigma \models' m : K & \textit{iff} & m \in \mathcal{F}(K\sigma) \\
\sigma \models' \neg\pi & \textit{iff} & \sigma \not\models' \pi \\
\sigma \models' \pi_1 \wedge \pi_2 & \textit{iff} & \sigma \models' \pi_1 \textit{ and } \sigma \models' \pi_2 \\
\sigma \models' \pi_1 \vee \pi_2 & \textit{iff} & \sigma \models' \pi_1 \textit{ or } \sigma \models' \pi_2 \\
\sigma \models' \exists v.\pi & \textit{iff} & \exists t \in \mathcal{T}^+ : \sigma \models' \pi[^t/_v] \\
\sigma \models' \forall v.\pi & \textit{iff} & \forall t \in \mathcal{T}^+ : \sigma \models' \pi[^t/_v]
\end{array}$$

We define a translation $\mathbf{T}(\phi, tr, \mathit{IK})$ from a $\mathcal{PS}$-LTL formula ϕ, a trace tr and an initial intruder knowledge IK into an EF formula:

Definition 24. *Let ϕ be a $\mathcal{PS}$-LTL formula, tr be an execution trace and IK be an initial intruder knowledge. Then $\mathbf{T}(\phi, tr, \mathit{IK})$ is the EF formula resulting from applying the three steps detailed in Table 4.*

It can be shown that the transformation $\mathbf{T}$ terminates and is confluent given a finite trace, although we do not prove that here. The last step (3) removes cases which are known not to hold. The following lemma states that the translation $\mathbf{T}$ is correct, i.e. it preserves the semantics of $\mathcal{PS}$-LTL w.r.t. semantics of EF.

Lemma 25. *Let ϕ be a closed $\mathcal{PS}$-LTL formula, tr be a trace and IK be an initial intruder knowledge, and let σ be a ground substitution such that $var(tr) \subseteq dom(\sigma)$. Then $\langle tr\sigma, \mathit{IK}\rangle \models \phi$ iff $\sigma \models' \mathbf{T}(\phi, tr, \mathit{IK})$.*

Transformation $\mathbf{T}$ provides the necessary input to the decision algorithm $\mathbf{D}$ of the next section, to decide validity of the original formula.

We call an EF formula *existential* if it is of the form $\exists v_1 \ldots \exists v_n.\varphi$, and φ does not contain any quantifiers ($\forall$ nor $\exists$). In addition, we say that an EF formula ϕ is *negation ground* if every occurrence of a negated constraint $\neg(m : T)$ in ϕ satisfies that m is ground.

We now define the subset Φ of $\mathcal{PS}$-LTL over which we are going to decide symbolic validity:

Definition 26. *Φ is the set of* well-behaving *$\mathcal{PS}$-LTL formulas: $\Phi \triangleq \{ \phi \mid \phi$ closed and $\mathbf{T}(\phi, tr, \mathit{IK})$ is existential and negation ground for all $tr, \mathit{IK} \}$*

Φ is expressive enough to allow the specification of several interesting security properties. In particular, every property φ considered in Section 3.2 satisfies $A_\varphi \in \Phi$.

4.3 Deciding Validity

Let ϕ be a well-behaving $\mathcal{PS}$-LTL formula and $\pi = \exists v_1 ... \exists v_n.\varphi$ be the result of the translation $\mathbf{T}(\phi, tr, \mathit{IK})$ for some tr and IK. We transform φ into its *disjunctive normal form* $\varphi = \bigvee_j \psi_j$, with $\psi_j = \bigwedge_i \pi_{j,i}$. Given a simple

1. First, we repeatedly apply transformation $\lfloor\cdot\rfloor\cdot$, until none of the rules can be applied:

$$\begin{array}{rcl}
\lfloor\exists v.\phi\rfloor tr & \Rightarrow & \exists v.\lfloor\phi\rfloor tr \\
\lfloor\forall v.\phi\rfloor tr & \Rightarrow & \forall v.\lfloor\phi\rfloor tr \\
\lfloor\neg\phi\rfloor tr & \Rightarrow & \neg\lfloor\phi\rfloor tr \\
\lfloor\phi_1 \wedge \phi_2\rfloor tr & \Rightarrow & \lfloor\phi_1\rfloor tr\ \wedge \lfloor\phi_2\rfloor tr \\
\lfloor\phi_1 \vee \phi_2\rfloor tr & \Rightarrow & \lfloor\phi_1\rfloor tr\ \vee \lfloor\phi_2\rfloor tr \\
\lfloor\mathtt{Y}\phi\rfloor\langle\rangle & \Rightarrow & \mathtt{false} \\
\lfloor\mathtt{Y}\phi\rfloor\langle tr\ e\rangle & \Rightarrow & \lfloor\phi\rfloor tr \\
\lfloor\phi_1\mathtt{S}\phi_2\rfloor\langle\rangle & \Rightarrow & \lfloor\phi_2\rfloor\langle\rangle \\
\lfloor\phi_1\mathtt{S}\phi_2\rfloor\langle tr\ e\rangle & \Rightarrow & \lfloor\phi_2\rfloor\langle tr\ e\rangle\ \vee\ (\lfloor\phi_1\rfloor\langle tr\ e\rangle\ \wedge \lfloor\phi_1\mathtt{S}\phi_2\rfloor tr) \\
\lfloor\mathtt{true}\rfloor tr & \Rightarrow & \mathtt{true} \\
\lfloor\mathtt{false}\rfloor tr & \Rightarrow & \mathtt{false} \\
\lfloor\mathtt{learn}(m)\rfloor tr & \Rightarrow & m : (K(tr) \cup \mathrm{IK}) \\
\lfloor p(d_1,\ldots,d_n)\rfloor\langle\rangle & \Rightarrow & \mathtt{false} \\
\lfloor p(d_1,\ldots,d_n)\rfloor\langle tr\ q(e_1,\ldots,e_m)\rangle & \Rightarrow & \mathtt{false}\text{ if } p \neq q \text{ or } n \neq m \\
\lfloor p(d_1,\ldots,d_n)\rfloor\langle tr\ p(e_1,\ldots,e_n)\rangle & \Rightarrow & d_1 = e_1 \wedge \cdots \wedge d_n = e_n
\end{array}$$

(Note how a `learn` translates directly into a constraint. Also, notice that in each equality $d_i = e_i$, $var(d_i) \cap var(e_i) = \emptyset$, as we require that variables from the formula and from the trace do not clash.)

2. Repeatedly rewrite atoms $\neg\neg\phi$ to ϕ, and move $\neg$ inside conjunctions and disjunctions using DeMorgan distributive laws.

3. Move universal quantifiers $\forall$ as far as possible to the right, and simplify universally quantified formulas over (possibly negated) equalities and constraints, according to the following rules:

$$\begin{array}{rcl}
\forall v.(\phi_1 \wedge \phi_2) & \Rightarrow & \forall v.\phi_1 \wedge \forall v.\phi_2 \\
\forall v.(\phi_1 \vee \phi_2) & \Rightarrow & \forall v.\phi_1 \vee \forall v.\phi_2 \text{ if } v \text{ is not free in } \phi_1 \text{ or } v \text{ is not free in } \phi_2 \\
\forall v.\phi & \Rightarrow & \phi \text{ if } v \text{ is not free in } \phi \\
\forall v.(v : K) & \Rightarrow & \mathtt{false} \text{ (where } K \text{ is a term set)} \\
\forall v.\neg(v : K) & \Rightarrow & \mathtt{false} \text{ (where } K \text{ is a term set)} \\
\forall v.\neg(v = t) & \Rightarrow & \mathtt{false} \\
\forall v.(v = t) & \Rightarrow & \mathtt{false}
\end{array}$$

(In the last two rules, t is a term s.t. $t \neq v$ since we require that $var(v) = \{v\} \cap var(t) = \emptyset$.)

Table 4. Transformation $\mathbf{T}(\phi, tr, \mathrm{IK})$

constraint set CS, the procedure $\mathbf{D}(\pi, CS)$ we are about to define either fails and returns `false` or succeeds and returns a substitution σ that makes π true. For simplicity, in the sequel we assume that each ψ_j contains just one *positive equality* $L_j = R_j$ (where L_j and R_j denote the left and right term in the equality, respectively), one *negated equality* $\neg(L_j^\neg = R_j^\neg)$ (where $L_j^\neg$ and $R_j^\neg$ denote the left and right term in the negated equality, respectively), one *positive constraint* $m_j : K_j$ and one *negated constraint* $\neg(m_j^\neg : K_j^\neg)$. The generalization to the case with several (possibly negated) atomic formulas and with `true` and `false` atoms is straightforward.

Procedure 27. *Let $\varphi = \bigvee_j \psi_j$, with $\psi_j = (L_j = R_j) \wedge \neg(L_j^\neg = R_j^\neg) \wedge (m_j : K_j) \wedge \neg(m_j^\neg : K_j^\neg)$. Let CS be a simple constraint set. Procedure $\mathbf{D}$ succeeds if all the following steps succeed, in which case it returns $\sigma = \rho\rho_k\sigma_V$ where ρ is given by Step 2, ρ_k is given by Step 3 and σ_V is the substitution described in Section 4.1.*

1. *Pick a disjunct ψ_j while possible, otherwise exit and return* false.
2. Solve Positive Equality: *Take a relevant most general unifier ρ of L_j and R_j such that $dom(\rho) \subseteq var(L_j) \cup var(R_j)$, i.e. $L_j\rho = R_j\rho$ (If no mgu exists, go back to Step 1).*
3. Solve Positive Constraint: *Apply $\mathbf{P}$ to $(CS \cup \{m_j : K_j\})\rho$. Let $\rho_1, \ldots, \rho_l$ be the partial solutions.*
4. *Pick ρ_k, $k \in [1 \ldots l]$, while possible, otherwise go back to Step 1.*
5. Solve Negated Constraint: *Apply $\mathbf{P}$ to $(CS \cup \{m_j : K_j, m_j^\neg : K_j^\neg\})\rho\rho_k\sigma_V$ (where σ_V is the substitution given in Section 4.1). If it is solvable, go back to Step 4.*
6. Solve Negated Equality: *Check that $L_j^\neg\rho\rho_k\sigma_V$ and $R_j^\neg\rho\rho_k\sigma_V$ differ syntactically, otherwise go back to Step 4.*

Step 2 tries to solve the positive equality, finding a suitable unifier ρ. (We need a unifier and not a matching for the general case of many equalities). In case ρ is not found, then the disjunct does not hold , so we try a different one going back to Step 1. Similarly, Step 3 solves the positive constraint. Step 5 checks that both ρ, ρ_k and σ_V cause the negated constraint to hold (this is based on Theorem 21). Finally, Step 6 checks that the negated equalities hold. Since we consider (i) relevant unifiers for Step 2, of which there are only finitely many, (ii) $\mathbf{P}$ only outputs a finite number of solutions for Step 3 and 5, and (iii) we only need to perform a syntactic check for Step 6, then we can deduce that procedure $\mathbf{D}$ terminates. The correctness of $\mathbf{D}$ is more challenging to establish.

Lemma 28. *Let ϕ be a well-behaving $\mathcal{PS}$-LTL formula, $\langle Sc, IK, CS, tr\rangle$ be a state from Procedure 13, and $\pi = \mathbf{T}(\phi, tr, IK)$. Then:*

1. *$\mathbf{D}(\pi, CS)$ succeeds and returns a substitution σ implies that $\sigma \models' \pi$, with $tr\sigma$ valid w.r.t. IK; and*
2. *$\sigma \models' \pi$, with $tr\sigma$ valid w.r.t. IK implies that there exists a substitution γ s.t. $\mathbf{D}(\pi, CS)$ succeeds and returns γ.*

Now we are ready to formulate the main result of this section, which states that applying the transformation $\mathbf{T}$ of Definition 24 and $\mathbf{D}$ defined in Procedure 27 is both sound and complete.

Theorem 29. *Let Sc_0 be a system scenario, IK be an initial intruder knowledge, ϕ be a closed $\mathcal{PS}$-LTL formula representing a security property with $A_\phi = \neg\phi$ is a well-behaving $\mathcal{PS}$-LTL formula. Let also $\langle Sc, IK, CS, tr\rangle$ be a state from Procedure 13 and $\pi = \mathbf{T}(A_\phi, tr, IK)$. Then $\mathbf{D}(\pi, CS)$ fails iff $\langle tr, IK\rangle \models \phi$.*

Proof. $\mathbf{D}(\pi, CS)$ fails iff, by Lemma 28, $\forall\sigma : \sigma \not\models' \mathbf{T}(A_\phi, tr, IK)$. By Lemma 25, $\langle tr\sigma, IK\rangle \not\models A_\phi$. By definition, this is equivalent to $\langle tr\sigma, IK\rangle \models \phi$. So, we obtained that $\forall\sigma : \langle tr\sigma, IK\rangle \models \phi$, which by Definition 17 of symbolic validity is $\langle tr, IK\rangle \models \phi$. ☐

4.4 Integrating $\mathcal{PS}$-LTL to Constraint Solving

Procedure 13 outputs execution traces which are instances of prefix interleavings representing *partial runs*, in which not every participant necessarily finishes its execution. For example, the empty trace $\langle\rangle$ is always output regardless of the input scenario Sc_0 or *IK* in Procedure 13. Clearly, the empty trace does *not* represent an attack on the protocol in question. We thus need to include a *termination condition* $TC_\phi(s)$ which is evaluated in *every* resulting state s of a run from Procedure 13 against a $\mathcal{PS}$-LTL security property ϕ. When $TC_\phi(s)$ holds, we know that an attack has happened and we can then stop execution and report the attack trace. Otherwise, execution should continue in search for another attack. Given a state $\langle Sc, \mathrm{IK}, CS, tr\rangle$, we define $TC_\phi(\langle Sc, \mathrm{IK}, CS, tr\rangle)$ to hold when $\mathbf{D}(\pi, CS)$ holds, for $\pi = \mathbf{T}(\neg\phi, tr, \mathrm{IK})$ with $\neg\phi$ is a well-behaving $\mathcal{PS}$-LTL formula. The termination condition essentially checks whether *tr* in the current execution state can be instantiated to provide a solution of $\neg\phi$, that is, to *falsify* ϕ: by Theorem 29, we know that $\mathbf{D}(\mathbf{T}(\neg\phi, tr, IK), CS)$ holds iff $\langle tr, IK\rangle \not\models \phi$. In that case, the procedure terminates and outputs the trace *tr* that shows an attack. Otherwise, the procedure proceeds until an attack or no attack is found.

	Without $\mathcal{PS}$-LTL	With $\mathcal{PS}$-LTL
RPC flawed	0.06s	2.48s
RPC fixed	0.08s	11.91s

Table 5. Benchmarks

Benchmarks In Table 5 we show some benchmarks of the implementation w.r.t. both the flawed and fixed version of the Andrew RPC protocol when checking authentication (in particular, aliveness), comparing to our old implementation without $\mathcal{PS}$-LTL. As we can see our prototype is considerably slower (we plan to optimize this, see Future Work in the next section).

5 Related Work

Our logic is related to the trace logic proposed in [9]. $\mathcal{PS}$-LTL provides more powerful temporal operators (e.g. the *yesterday* `Y` and *since* `S` operators), which in turn allows one to write of more expressive security properties, like perfect forward secrecy. $\mathcal{PS}$-LTL is inspired by the successful and elegant NPATRL logic [30], but, as shown in subsequent work [26], NPATRL is strictly less powerful than LTL; also in that paper it is mentioned that the implementation of NPATRL to NRL Protocol Analyzer [24] presents difficulties (e.g. the inability to mention several *learn*'s in the same formula, a restriction we do not impose and that is essential to specify e.g. perfect forward secrecy). Our treatment of pure-past LTL is an adaptation of Havelund and Rosu [19]. We provide a different semantics tailored for security and constraint solving, but also include a different definition for *historically* `H`, which we believe preserves better the faithfulness to standard LTL.

Our decision procedure exploits the ability to solve negated constraints. The idea to solve negated constraints is based on allowing the intruder to generate constants, as done originally in the work of Kähler and Küsters [21] for analysis of contract signing protocols in constraint solving.

Finally, we use events *start*, *run* and *end* to specify authentication properties as correspondence assertions à la Gordon and Jeffrey [18].

6 Conclusions and Future Work

We propose $\mathcal{PS}$-LTL, a language for specifying security properties. Our language is based on linear temporal logic (LTL) with pure-past operators. This language is both simple to use and expressive, as evidenced by the ability to specify several security properties including authentication [23, 12] (*aliveness*, *weak agreement* and *non-injective agreement*), secrecy (*standard secrecy* [2] and *perfect forward secrecy* [16]) and also data freshness [9]. We also study properties to prevent against denial of service (DoS) [25] attacks. Having a dedicated language to specify properties allows the protocol designer to fairly separate the protocol instances under study from the properties one wants to check. This is useful during the verification phase of the protocol, as it allows to change the protocol instances (e.g. to add more sessions) while keeping the property unchanged.

We present a sound and complete decision procedure to check a fragment of $\mathcal{PS}$-LTL against symbolic traces, thus integrating the $\mathcal{PS}$-LTL interpreter into our protocol verification tool,providing a full verification system This has significant practical value to protocol designers, as effective protocol debugging can be applied effectively during the engineering phase of the security protocol design.

Future Work There are many possible directions for future work. As already mentioned, we would like to study further the specification of denial of service properties (see Section 3.2.3).

It is theoretically interesting to study whether a general strategy to solve negated constraints exists (in Section 4.1 we provide a strategy to solve $\neg(m : T)$ in the particular case of m ground; Still, this is enough to cover all our properties of interest). It would also be interesting to relate negated constraints to other approaches in the literature to solve "negative" predicates, e.g. `nonunif` of [3].

Another direction is to implement formula checking more efficiently: for example, such an implementation would not recompute the translation of $\mathcal{PS}$-LTL to elementary formula EF *every* time a property is checked, but maintain an internal data structure which can be optimized as the trace gets expanded, along the lines of [19]. We are also interested on enlarging the subclass Φ of $\mathcal{PS}$-LTL from Section 4.2, thus obtaining a more expressive language (e.g. to cover stronger authentication notions like the ones in [12]).

Acknowledgements We are grateful to Pieter Hartel, Ralf Küsters and the anonymous reviewers (both from this conference and from the FMSE2005 workshop) for their very useful comments.

References

[1] D. Basin, S. Modersheim, and L. Vigano. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *Workshop on Security Protocol Verification. CONCUR 2003*, September 2003.

[2] B. Blanchet. Automatic proof of strong secrecy for security protocols. Research Report MPI-I-2004-NWG1-001, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, July 2004.

[3] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS*

2005), pages 331–340, Chicago, IL, June 2005. IEEE Computer Society.

[4] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

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

[6] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. `http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`, 1997.

[7] J. A. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0. University of York, Department of Computer Science, November 1997.

[8] R. Corin. *Analysis Models for Security Protocols*. PhD thesis, University of Twente, 2006.

[9] R. Corin, A. Durante, S. Etalle, and P. H. Hartel. A trace logic for local security properties. In *Int. Workshop on Software Verification and Validation (SVV)*, volume 118, Mumbai, India, Dec 2003. Elsevier Science in Electronic Notes in Theoretical Computer Science.

[10] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M. V. Hermenegildo and G. Puebla, editors, *9th Int. Static Analysis Symp. (SAS)*, volume LNCS 2477, pages 326–341, Madrid, Spain, Sep 2002.

[11] R. Corin, S. Etalle, and A. Saptawijaya. Online demos. Improved Constraint-based procedure at `http://130.89.144.15/cgi-bin/show.cgi`. $\mathcal{PS}$-LTL demo at `http://130.89.144.15/cgi-bin/psltl/show.cgi`, June 2005.

[12] C. Cremers, S. Mauw, and E. de Vink. Defining authentication in a trace model. In T. Dimitrakos and F. Martinelli, editors, *Fast 2003*, Proceedings of the first international Workshop on Formal Aspects in Security and Trust, pages 131–145, Pisa, September 2003. IITT-CNR technical report.

[13] S. Delaune and F. Jacquemard. A theory of dictionary attacks and its complexity. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 2–15, Asilomar, Pacific Grove, California, USA, June 2004. IEEE Computer Society Press.

[14] R. Delicata and S. Schneider. Temporal rank functions for forward secrecy. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 126–139, 2005.

[15] G. Delzanno and S. Etalle. Proof theory, transformations, and logic programming for debugging security protocols. In A. Pettorossi, editor, *11th Int. Logic Based Program Synthesis and Transformation (LOPSTR)*, volume LNCS 2372, pages 76–90, Paphos, Greece, Nov 2001. Springer-Verlag, Berlin.

[16] W. Diffie, P. C. V. Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107 – 125, June 1992.

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

[18] A. D. Gordon and A. S. A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *J. Computer Security*, 12(3/4):435–484, 2004.

[19] K. Havelund and G. Rosu. Testing linear temporal logic formulae on finite execution traces. Technical Report TR 01-08, RIACS, 2001.

[20] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000.

[21] D. Kähler and R. Küsters. Constraint Solving for Contract-Signing Protocols. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR 2005)*, 2005. To appear.

[22] G. Lowe. Some new attacks upon security protocols. In *PCSFW: Proceedings of The 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[23] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop, 1997*, pages 31–44. IEEE Computer Society Press, 1997.

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

[25] C. Meadows. A formal framework and evaluation method for network denial of service. In *CSFW '99: Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, page 4, Washington, DC, USA, 1999. IEEE Computer Society.

[26] C. Meadows, P. F. Syverson, , and I. Cervesato. Formal specification and analysis of the group domain of interpretation protocol using NPATRL and the NRL protocol analyzer. *Journal of Computer Security*, 12(6):893–931, 2004.

[27] J. Millen. Constraint solver webpage, at `http://www.csl.sri.com/users/millen/capsl/constraints.html`.

[28] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *8th ACM Conference on Computer and Communication Security*, pages 166–175. ACM SIGSAC, November 2001.

[29] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In S. Schneider, editor, *Proc. 14th IEEE Computer Security Foundations Workshop*, 2001.

[30] P. Syverson and C. Meadows. A formal language for cryptographic protocol requirements. *Designs, Codes and Cryptography*, 7:27 – 59, 1996.

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