$\mathcal{A}$SPASyA: an Automated tool for Security Protocol Analysis based on a Symbolic Approach

Giacomo Baldi, Andrea Bracciali, Gianluigi Ferrari, and Emilio Tuosto

Dipartimento di Informatica
Università di Pisa, Italy
{baldig,braccia,giangi,etuosto}@di.unipi.it

Abstract. The quest for the formal certification of properties of security protocols is one of the most challenging research issues in the field of formal methods. It requires the development of formal models together with effective verification techniques, methods of detecting malicious behaviour, and so on. In this paper, we describes a formal methodology for verifying cryptographic protocols based on a symbolic state space exploration technique. We also present Aspasya, a semi-automatic verification tool based on our formal framework.

1 Introduction

Protocols based on cryptographic mechanisms play a fundamental role to achieve security of distributed applications. They are used to provide secure communication services (e.g. *Secure Socket Layer*), authentication services for handling and distributing passwords (e.g. *Kerberos*), and so on. The design of cryptographic protocols is highly error-prone. Many formal approaches have been developed and applied to the certification (specification and verification) of cryptographic protocols. Some of these techniques are based on state space exploration (e.g. finite-state model checking [12, 15, 7].). The main advantage of state-space exploration with respect to other approaches is that it provides counterexamples when a protocol does not satisfy some properties. A major issues is therefore the *dimension* of the state space. Due to the rich structure of the messages exchanged inside a protocol session and to the power of the adversary (the intruder), the state-space may explode thus making state-space exploration unfeasible.

The standard way of addressing this problem is to assume an upper bound on the size of messages synthesized by the intruder [12, 15, 7]. Recently, symbolic approaches have been developed to keep finite the state-space [2, 3, 9]. The basic idea of symbolic approaches is to adopt constraint systems (e.g. variation of unification) to express the values variables may assume. Moreover, properties are specified by inserting logical assertions in the specification of the protocol. TRUST [2] and STA [3, 4] are verification environments which model-check security properties by a symbolic state-space exploration.

Our research is aimed at further developing verification methodologies based on the notion of symbolic state-space exploration. Indeed, current verification methodologies are not satisfactory for several reasons. First, they do not clearly separate the specification of protocol behaviour from the specification of its security properties. Usually, the specification of the properties is hard-written into the code of the protocol. We argue that the clear distinction among the two specifications will avoid mis-interpretation of the protocol. Moreover, it will allow one to analyze a cryptographic protocol by looking at different security properties. Second, the assumptions underlying the formal notion of correctness are often not sufficiently formalized. This may lead into difficulties in the verification phase. For instance, it can allow us to prove correctness of a flawed protocol. Third, the life-cycle of a cryptographic protocol depends on the adversary: The precise specification of the adversary knowledge becomes a powerful verification device. It allows one to check correctness of a cryptographic protocol under different assumptions about the power of the adversary.

In this paper we present a verification methodology that tackles the three issues discussed above. Our methodology allows the verification of cryptographic protocols by controlling three different coordinates: The intruder's knowledge, the property specification and the specification of implicit assumptions. Users can opportunely mix those three ingredients for testing the correctness of the protocol without modifying the protocol specification. Moreover, also the definition of the desired property let the verifier abstract away specification details; for example,thenumber of interleaved execution of the protocol for which the property has to be checked. The separation of concerns is the added value of our approach with respect to other approaches based on symbolic state-space exploration.

The proposed methodology is supported by a verification environment called $\mathcal{A}$SPASyA (Automated tool for Security Protocol Analysis based on a Symbolic Approach). The front-end of $\mathcal{A}$SPASyA permits specifying cryptographic protocols by a name passing process calculus (called cIP). Properties are given with a suitable fragment of first order logic. In this paper we do not focus on the theoretical development of cIP (we refer to [5, 17] for details). Instead, we address the issue of the usage of the methodology as a basis for the design and formal certification of cryptographic protocols. To illustrate the effectiveness and usability of our approach, we consider some case studies which allows us to deal with some verification patterns for cryptographic protocols.

Related Work

The cIP calculus and its symbolic model have some similarities with the calculi introduced in [3, 4, 18]. However, our verification methodology clearly distinguishes specification and verification concerns. For instance, in TRUST [18], security properties are "embedded" in the protocol specification and the verification phase just consists of invoking the symbolic trace generator. Moreover, even though the intruder's knowledge and the state space can be controlled, it is necessary to modify the protocol when different verification sessions have to be activated. There are several analogies between our approach and the one adopted in the STA [3, 4] verification environment. However, the STA logic is a special purpose logic specifically designed to deal with certain kinds of security properties. Our logic is more expressive, and moreover our methodology does not require to rewrite the security property when different sessions of the same protocol are considered (as it happens in STA).

2 Cryptographic Protocols: An Overview

Cryptographic protocols have been used to avoid lacking/modifying "sensitive" information in a scenario where two (or more) partners (sometimes called *principals*) communicate through a "public channel". Usually, the word "sensible" means information that should be kept secret or non-modifiable (e.g. a credit card number, an encryption key, etc.). Certification (specification and verification) of cryptographic protocols requires a careful definition of the underlying assumptions adopted both in the algorithms used to encrypt/decrypt messages and in the hypothesis on the capabilities of malicious participants to communications (usually called *intruder* or *attacker*).

This section is devoted to a brief review of some elementary notions on cryptography, protocols specification, security properties and the attacker model. We refer to [16, 13] for a comprehensive introduction to this field.

Cryptography An *intelligible* message m, is referred to as *plaintext* (or *datagram*). By 'intelligible' we mean that the representation of the information denoted by m is public domain knowledge. An *unintelligible* form of m is referred to as *ciphertext* (or *cryptogram*).

The process of assigning a ciphertext to a plaintext is called *encryption*; encryption is parameterised with respect to an *encryption key*. Hereafter, $\{m\}_k$ denotes the cryptogram obtained by encrypting message m with key k, while m, n denotes the pair made of messages

m and n. We assume that keys have no structure, i.e. they are simply names. Given k, *decryption* extracts m from $\{m\}_k$.

Cryptography can be based on either *public* or *private* keys: Indeed two principals, say A and B, can encrypt/decrypt data if they share a key k. It is usually assumed that k is known only by A and B and other principals may acquire k only if A or B explicitly send it. Public key cryptography is characterized by the fact that encryption and decryption keys differs from each other. Each principal A has a *private* key and a *public* key, respectively denoted by A^- and A^+. For each principal A, its public key A^+ is publicly available and may be used by any other principal to encrypt messages intended for A. Such cryptograms may be decrypted only using the private key A^- that only A owns.

The robustness of a system relies on the security of various levels of its architecture and their relationships. At the cryptographic level, the standard working assumption is the so-called *perfect encryption hypothesis* stating that a cryptogram can be decrypted only using its decryption key and that secrets cannot be guessed, no matter how much information is possessed[1].

Protocol specification A *security protocol* may be naively thought of as a finite sequence of messages between two or more participants. There is a great variety of specifications mechanisms of protocols and their properties. Some protocols are informally specified mixing natural language and ad hoc notation (for instance, SSL [10], SSH [19], IKE [11] are specified in this style). Protocols are usually presented as a list of message exchanges written as in the following example.

Example 1. We give the informal specification of the Wide Mouthed Frog (WMF) protocol [6]. The purpose of WMF protocol is to let A send a fresh session key k^{ab} to B through a trusted server S. Principals A and B each share one private keys with S (k^{as} and k^{bs}, respectively). The WMF protocol is:

$$\begin{array}{l}(1)\ A \to S :\ A, \{T^a, B, k^{ab}\}_{k^{as}} \\ (2)\ S \to B : \{T^s, A, k^{ab}\}_{k^{bs}}.\end{array}$$

First, A encrypts for S the identity of B, the session key k^{ab} and a *fresh* time-stamp T^a intended to be used only for a session of the protocol; such names are called *nonces*. By the perfect encryption hypothesis, T^a and k^{ab} cannot be "guessed" by any other participant of the protocol. Then, S forwards k^{ab} and the identity of the initiator to B; timeliness of k^{ab} is witnesses by T^s, a nonce generated by S. $\diamond$

A sequence of message exchanges is not a complete specification for cryptographic protocols. For instance, the specification in Example 1 does not specify whether or not only B (and S) must know k^{ab} and T^s. Furthermore, tests on messages in a security protocol are important: *Type flaws* are possible if the shape of the message cannot be recognised or if a principal implicitly assumes that the received data have a given form. We adopt two further assumptions: (i) "messages are typed", namely, they contains enough information for a principal to recognise their shape and (ii) quoting [6], it is assumed that a protocol may have multiple simultaneous runs and that principals may play different roles in different runs.

Security properties Many security properties can be stated for a given protocol. We mainly focus on *integrity*, *secrecy* and *authentication*. Intuitively, a protocol guarantees integrity if, once a datum has been provided, it cannot be altered by any intruder. A protocol

[1] Such hypothesis is not completely realistic; indeed, under it, *cryptanalysis attacks* (i.e., those attacks that are performed by collecting a great number of cryptograms and then analyzing them for deducing cryptographic keys) cannot be captured. However, realistic keys cannot be deduced in polynomial time by intruders that have a given computational capacity.

guarantees secrecy over a set of data if it is not possible that an intruder will get such data. A protocol guarantees authentication of a user A to a user B if, after running the protocol, B may *safely* assume that A was involved in the protocol run. Secrecy and authentication are closely related. Indeed, before sending secrets to B, A should be sure that (s)he is effectively "speaking" to B. Authentication is normally achieved through exchange of some data that the protocol ensure to be created by the intended partner and nobody else.

If secret information must be communicated in an untrusted environment, the protocol must ensure at least that possible eavesdroppers cannot understand them (secrecy), that the partner in the communication is really the intended one and that the message are really generated by the intended participant (message authenticity). Secrecy, authentication and integrity are the "elementary" properties that protocols aim at guaranteeing.

Example 2. Let us consider again the WMF protocol specification (Example 1). A possible requirement of the protocol is the secrecy of k^{ab}, namely, in every session, the value of k^{ab} must be known only by the principals playing the roles of A and B and S. Another requirement is that the protocol guarantees the authenticity k^{ab}, i.e., in every session where B receives the message from S, (s)he must be ensured that, *in the same session*, (i) A has created k^{ab} and (ii) that A asked S to forward the key to B. ⋄

Intruder model A formal framework for protocol analysis must declare which assumptions are made on the intruder.

The Dolev-Yao model [8] is a widely accepted model. It describes an active intruder as a "principal" that can

- receive and store any transmitted message;
- hide a message;
- decompose messages into parts;
- forge messages using known data.

The only limitation for intruders imposed by the Dolev-Yao model is the assumption of the perfect encryption hypothesis. The model also assumes that intruders can have some private data, namely, information which have not been generated by regular principals. This amounts to saying that intruders can "remember" data exchanged in previous runs of the protocols. The model characterizes an intruder in terms of its knowledge about the exchanged data. In particular, an intruder can record all exchanged messages and use them later to fake principals or to extract data that must be kept secret.

Since an intruder *á la* Dolev-Yao can intercept any communication, it can be formalised as the execution environment which behaves as the "adversary" of "honest" principals. The environment collects all sent messages and manipulates them when a principal is waiting for some data.

Let $N = N_o \cup N_p$ be a countable set of names, where N_o is the set of nonces and N_p is the set of principal names; we assume that $N_o \cap N_p = \emptyset$. Let K be a set of keys; K contains both symmetric and asymmetric keys and that $K \cap N = \emptyset$.

Definition 1 (Messages). *A* message *is a term derived as follows:*

$$M ::= N \quad | \quad K \quad | \quad M, M \quad | \quad \{M\}_M.$$

We let m, n, ... to range over M, while λ ranges over K; λ^- denotes the inverse key of λ, namely, $\lambda^- = \lambda$ if λ is a symmetric key, while $\lambda^- = A^-$ if $\lambda = A^+$ and $\lambda^- = A^+$ if $\lambda = A^-$.

A message may be a name (i.e. a nonce or a principal name), a key (symmetric or not), the pairing of two messages or the encryption of a message.

As usual, we assume that keys are just names (not complex terms) and that messages encode also their shape; for instance, m_1, m_2, m_3 can be only matched by patterns that are triples whose i-th component matches m_i.

The Dolev-Yao intruder is characterised by a set of messages κ and the actions the intruder can perform on the elements of κ. We call it the intruder knowledge. Basically, an intruder can pair known messages, split pairs and encrypt/decrypt cryptograms if the corresponding keys can be deduced by κ. We write $\kappa \bowtie m$ to denote that datum m can be obtained by a finite sequence of such actions from messages in κ.

Example 3. Consider the knowledge $\kappa = \{\{A^-\}_k,\ \{m\}_{A^+},\ k\}$, we show how $k \bowtie \{m\}_k$ with the following deduction tree:

$$\dfrac{\dfrac{\kappa \bowtie \{A^-\}_k \qquad \kappa \bowtie k}{\kappa \bowtie A^-} \qquad \kappa \bowtie \{m\}_{A^+}}{\kappa \bowtie m}$$

and, since $\kappa \bowtie k$, we can encrypt m with k, obtaining $\{m\}_k$. ⋄

In [7] decidability of $\bowtie$ has been proved for private key cryptography; decidability of $\bowtie$ for public key cryptography has been proved in [5, 17].

3 A Calculus and a Logic for Security

This section presents the main features of the *cryptographic interaction pattern* (cIP) calculus and the *protocol logic* ($\mathcal{PL}$) introduced in [5, 17] to specify and prove properties of security protocols.

The cIP calculus The cIP calculus is a name-passing process calculus in the style of the π-calculus [14] with cryptographic primitives.

A cIP processes $A \triangleq (\tilde{X})[E]$ is a principal whose name is A, having *open variables* $\tilde{X}$ and whose sequence of actions is specified by E. We assume that a cIP process can only perform a finite sequence of $in(d)$ or $out(d)$ actions[2]. Datum d is a message where variables can appear; we denote the binding occurrences of a variable x with $?x$ and assume that, for any variable x,

- output actions $out(d)$ do not contain any occurrence of $?x$ in d;
- input actions $in(d)$ have at most one occurrence of $?x$ in d.

Notions of free and bound occurrence of variables, can be defined in the standard way. Hereafter we only consider processes where variable occurrences in the actions either are bound of open variables. It is worth remarking that names and variables are syntactically distinguished entities. Names should be thought of as being constant terms whereas variables are placeholders and are amenable to be substituted with terms or opportunely renamed.

To illustrate the main features of cIP we consider the WMF protocol.

Example 4. Principals of the WMF protocol are described by the following cIP terms:

$$\begin{aligned}
A &\triangleq (X, x^{as})[out(A, \{T^a, x^{as}, k^{ab}\}_{x^{as}})],\\
S &\triangleq (U, y^a, V, y^b)[in(U, \{?t, V, ?r\}_{y^a}).out(\{T^s, U, r\}_{y^b})],\\
B &\triangleq (z^{bs})[in(\{?s, ?x, ?w\}_{z^{bs}})].
\end{aligned}$$

⋄

[2] We are interested in non-recursive, non-deterministic protocols; in [5, 17] more expressive processes have been considered.

A protocol specification is formalised by defining a cIP term for each protocol principal. A distinguished feature of cIP is given by the notion of *open variables* that (together with the *join* operation) provide an explicit mechanisms for sharing names/keys. In Example 4 the open variable X parameterises the identity of the responder, while x^{as} and z^{bs} are the variables deemed to be identified with one of the variables for symmetric keys of S. The server S has four open variables; U and y^a are reserved for the identity and symmetric key of the initiator, whereas V and y^b are used for the identity and the symmetric key of the responder.

An *instance of principal* $A \stackrel{\triangle}{=} (\tilde{X})[E]$ is a process obtained by indexing all variables (open or not) and all names in E with a natural number $i > 0$. Instances run in *contexts*, i.e. (possibly empty) sets of instances where computation takes place and new instances may be dynamically added. Many instances of the same principal may (non-deterministically) join the context modelling the execution of more sessions of the protocol.

We say that a context $\mathcal{C}$ has cardinality n if it contains n instances of principals. We use $\mathrm{ov}(\mathcal{C})$ to indicate the set of open variable of context $\mathcal{C}$.

Definition 2 (Join). *Let $A_n \stackrel{\triangle}{=} (\tilde{X}_n)[E_n]$ be an instance, $\mathcal{C}$ be a context of cardinality $n-1$ and γ be a partial mapping whose domain is $\mathrm{ov}(\mathcal{C}) \cup \tilde{X}_n$ and such that, for each $x \in \mathrm{ov}(\mathcal{C}) \cup \tilde{X}_n$, $\gamma(x)$ is either in K or in N_p depending whether x is a variable for a symmetric key or for a principal. The* join *operation is defined as:*

$$join(A_n, \gamma, \mathcal{C}) = (\tilde{X}_n - dom(\gamma))[E_n\gamma] \;\cup \bigcup_{(\tilde{Y})[E'] \in \mathcal{C}} (\tilde{Y} - dom(\gamma))[E'\gamma].$$

The join operation defines how a principal instance can enter a (running) context by connecting open variables for asymmetric keys to principal names and open variables for symmetric keys to keys K so that they are appropriately shared. Connected variables are no longer open.

The pattern matching mechanism specifies complementarity of outputs and input actions.

Definition 3 (Matching). *Let m and n be two messages. We say that m and n* match *($m \sim n$) if, and only if,*

- *if $m, n \in N \cup K$ then $m = n$;*
- *if $m = p, q$ then $n = p', q'$ and $p \sim p'$ and $q \sim q'$;*
- *if $m = \{m'\}_\lambda$ and $n = \{n'\}_{\lambda^-}$ then $m' \sim n'$.*

A datum d matches *a message m if, and only if, there exists a substitution γ over the variables occurring in d such that $d\gamma \sim m$.*

The first two clauses of Definition 3 are straightforward. The third clause deals with cryptograms. The intuition is that $\{m'\}_\lambda \sim \{n'\}_{\lambda'^-}$ if $m' \sim n'$ do match and λ' is the inverse of the key of λ[3]. Definition 3 becomes clearer if we consider that, in cIP, encryption and decryption mechanisms are embodied into the *in* and *out* actions. For instance, the *in* action of principal B in Example 4 waits for a triple encrypted with k^{bs}, the symmetric key B shares with S and, when such a message arrives the components are assigned to the corresponding variables.

Binding occurrences of data input actions are patterns that must match data of output actions. If the matching holds then the synchronization can take place and the binding variables are substituted with corresponding values in output data.

The semantic model of cIP aims at formalizing the behaviour of the Dolev-Yao intruder. Hence it records all the exchanged messages and the connections of open variables of principal instances. The semantics of cIP is given by means of the reduction relation $\longmapsto$, the

[3] In the case of symmetric keys this simply reduces to require that the encryption keys are equal. For instance, $\{B, k^{ab}, T^a\}_{k^{as}}$ matches itself. In the case of asymmetric keys Definition 3 states that m matches n when the key of m is the public (private) key of a principal and the key of n is the private (public) key of the same principal (e.g. $\{A\}_{B^-} \sim \{A\}_{B^+}$).

smallest binary relation between *configurations* induced by the inference rules in Table 1. A configuration is a triple $\langle \mathcal{C}, \chi, \kappa \rangle$ where:

- $\mathcal{C}$ is a context,
- χ is a variable binding that keeps track of the associations of the variables due to communications and join executions
- and κ contains the names of instances that joined the context and the data sent along the public channel, i.e. κ represents the intruder knowledge.

$$\frac{\kappa \bowtie m : \ \exists \gamma \text{ ground s.t. } d\gamma \sim m}{\langle (\tilde{X}_i)[in(d).E_i] \cup \mathcal{C}, \chi, \kappa \rangle \ \longmapsto \ \langle (\tilde{X}_i)[E_i\gamma] \cup \mathcal{C}, \chi\gamma, \kappa \rangle} \ (in)$$

$$\frac{—}{\langle (\tilde{X}_i)[out(m).E_i] \cup \mathcal{C}, \chi, \kappa \rangle \ \longmapsto \ \langle (\tilde{X}_i)[E_i'] \cup \mathcal{C}, \chi, \kappa \cup m \rangle} \ (out)$$

$$\frac{\mathcal{C}' = join(A_i, \gamma, \mathcal{C}) \qquad A \triangleq (\tilde{X})[E] \qquad i \text{ new}}{\langle \mathcal{C}, \chi, \kappa \rangle \ \longmapsto \ \langle \mathcal{C}', \chi\gamma, \kappa \cup \{A_i, A_i^+\} \rangle} \ (join)$$

Table 1. Context reduction semantics

Relation $\longmapsto$ models both communications taking place inside contexts and the possible evolutions of a context due to the joining of new instances.
Rule (in) describes the evolution of a context containing an instance waiting for a datum d: If the knowledge of the environment can generate a message that matches d via a ground substitution γ, then the system evolves to a configuration obtained by applying γ to the continuation of the instance and recording bindings determined by γ.
Rule (out) states that a message sent by an instance is stored in κ. Notice that, the hypothesis of using only closed principals guarantees that only ground data, i.e. messages, are sent by principals.
Rule $(join)$ provides a mechanism to express the dynamic composition of components to a running open context by adding a new instance to the context. Moreover, note that the intruder is aware of the entering of new instances: A_i and its public key A_i^+ is added to κ.

Example 5. Let us consider the instances obtained by indexing WMF principals A, S and B in Example 4 respectively with 3, 2 and 1. According to cIP semantics we join them using $\gamma_0 = \begin{cases} z_1^{bs}, x_3^{as}, y_2^a, y_2^b \mapsto k, \\ X_3, V_2 \qquad\quad\ \mapsto B_1, \\ U_2 \qquad\qquad\quad \mapsto A_3. \end{cases}$ in an empty context and obtain the configuration $\langle \{A_3, S_2, B_1\}, \ \emptyset, \gamma_0 \rangle$, where

$$\begin{aligned} A_3 &\triangleq ()[out(A_3, \{T_3^a, B_1, k_3^{ab}\}_k)], \\ S_2 &\triangleq ()[in(A_3, \{?t_2, B_1, ?r_2\}_k).out(\{T_2^s, A_3, r_2\}_k)], \\ B_1 &\triangleq ()[in(\{?s_1, ?x_1, ?w_1\}_k)]. \end{aligned}$$

Notice that A_3, S_2 and B_1 share the same key.

A possible trace of executing the specification of WMF protocol from an empty knowledge is given below.

$$
\begin{array}{l}
\langle\{A_3, S_2, B_1\},\ \emptyset, \gamma_0\rangle \\
\mapsto \langle\{()[], S_2, B_1\}, \{A_3, \{T_3^a, B_1, k_3^{ab}\}_k\}, \gamma_0\rangle \\
\mapsto \langle\{()[], S_2', B_1\}, \{A_3, \{T_3^a, B_1, k_3^{ab}\}_k\}, \gamma_0[{}^{T_3^a, k_3^{ab}}/_{t_2, r_2}]\rangle \\
\mapsto \langle\{()[], ()[], B_1\}, \{A_3, \{T_3^a, B_1, k_3^{ab}\}_k, \{T_2^s, A_3, k_3^{ab}\}_k\}, \gamma_0[{}^{T_3^a, k_3^{ab}}/_{t_2, r_2}]\rangle \\
\mapsto \langle\{()[], ()[], ()[]\}, \kappa_1, \gamma_1\rangle,
\end{array}
$$

where κ_1 is the set $\{A_3, \{T_3^a, B_1, k_3^{ab}\}_k, \{T_2^s, A_3, k_3^{ab}\}_k\}$ and γ_1 is the mapping $\gamma_0[{}^{T_3^a, k_3^{ab}, T_3^a, A_3, k_3^{ab}}/_{t_2, r_2, s_1, x_1, w_1}]$. The final assignment is obtained when the environment sends the cryptogram $\{T_3^a, B_1, k_3^{ab}\}_k$ to B_1 (observe that the match holds). $\diamond$

The cIP calculus offers the possibility of *uniformly* extending a context with new instances of principals of the protocol. What is meant here by "uniformly" is the fact that variables and names occurring in principal expressions are labelled with a unique index when instances join the context. This linguistic mechanism allows us to determine which are the instances that originated the names used through the execution of the protocol as well as to distinguish between different participants playing the same role.

$\mathcal{PL}$ logic In our framework security properties are expressed by the $\mathcal{PL}$ logic (protocol logic) that allows one to express properties concerning values that variables are supposed to assume, membership of messages to the intruder's knowledge κ and relations among the values shared by different instances. In this logic, integrity corresponds to the possibility of fixing some value, generalizing the approach introduced in [1], secrecy is handled by exploiting intruder knowledge and the values it may or may not contain, and authentication through relations among principals' variables. The design of our logic has been driven by the features of the cIP calculus.

Definition 4 **($\mathcal{PL}$ – Syntax).** *A formula of the logic $\mathcal{PL}$ is defined as follows:*

$$
\begin{array}{rl}
\phi, \psi ::= & \delta \in \mathfrak{K} \ \mid\ \alpha = \beta \ \mid\ x@\alpha = \delta \\
& \mid\ \forall \alpha : A.\phi \ \mid\ \neg\phi \ \mid\ \phi \wedge \psi \\
\\
\delta \quad ::= & d \ \mid\ \alpha \ \mid\ x@\alpha \ \mid\ \mathbf{I}
\end{array}
$$

where d is datum that does not contain any binding occurrence.

Operators $\neg$, $\wedge$ and $\vee$ are the usual boolean operators[4]. The symbol $\mathfrak{K}$ is used to represent the knowledge that the intruder acquires during a protocol computation.

Definition 4 introduces a new class of variables which are the *instance variables* α, β, etc., that are subject to equality check and quantification. Instance variables range over (indexed) instances of roles and are "typed" by principal names. For instance, proposition $\forall\alpha : A.\phi$ reads as "for all instances of A, ϕ holds".

Among the possible values that can be expressed in $\mathcal{PL}$ formulae there is the distinguished constant $\mathbf{I}$ that denotes the intruder's identity. This permits expressing propositions where the identity of the partner is not necessarily a "regular" role.

Example 6. A property that the WMF protocol should satisfy is the secrecy of the session key k^{ab}, unless it is really intended for $\mathbf{I}$:

$$\forall\alpha : A.X@\alpha \neq \mathbf{I} \rightarrow k^{ab}@\alpha \in \mathfrak{K},$$

that captures the intuitive secrecy property above. $\diamond$

[4] Derived relations $\neq$ and $\notin$, logical connectors $\rightarrow$ and $\vee$, or existential quantifier $\exists$ are defined as usual and will be used as syntactic sugar.

Formulae are verified with respect to a given (terminating) context of a computation and the instances that have taken an active role in the computation. Notation $\kappa \models_\chi \phi$ indicates that κ, under the variable assignment χ, is a model of the formula ϕ.

Definition 5 (Models of $\mathcal{PL}$). *Let χ be a ground substitution for variables X. A* model *for a closed formula ϕ is a pair $\langle \kappa, \chi \rangle$ such that $\kappa \models_\chi \phi$ can be proved by the following rules:*

$$\frac{i = j}{\kappa \models_\chi A_i = A_j}\,(=_1) \qquad \frac{x_i\chi = \delta\chi}{\kappa \models_\chi x@A_i = \delta}\,(=_2) \qquad \frac{\kappa \bowtie \delta\chi}{\kappa \models_\chi \delta \in \mathfrak{K}}\,(\in) \qquad \frac{\kappa \not\models_\chi \phi}{\kappa \models_\chi \neg\phi}\,(\neg)$$

$$\frac{\kappa \models_\chi \phi \quad \kappa \models_\chi \psi}{\kappa \models_\chi \phi \wedge \psi}\,(\wedge) \qquad \frac{\kappa \models_\chi \phi[^{A_j}/_\alpha] \quad \textit{for all } A_j : \kappa \bowtie A_j}{\kappa \models_\chi \forall\alpha : A.\phi}\,(\forall).$$

Rule $(=_1)$ says that $\langle \kappa, \chi \rangle$ is a model of equality $A_i = A_j$ whether the instances are exactly the same instance. Rule $(=_2)$ says that $\langle \kappa, \chi \rangle$ is a model of $x@A_i = \delta$ whether the value associate by χ to variable x of instance A_i, i.e. $x_i\chi$ equals the valued $\delta\chi$. Rule $(\in)$ establishes that $\kappa \models_\chi \delta \in \mathfrak{K}$ whenever $\delta\chi$ can be constructed from the decomposition set of κ. Rules $(\neg)$ and $(\wedge)$ are straightforward. In $\forall\alpha : A.\phi$ the universal quantifier ranges over the finite set of instances of role A. Quantifiers are solved by relating variables to actual instances. In order to prove that $\langle \kappa, \chi \rangle$ is a model for a formula $\forall\alpha : A.\phi$, it is necessary to show that $\langle \kappa, \chi \rangle$ is a model for any formula obtained by substituting A_j for α in ϕ, where A_j is any instance of A deducible from κ.

Since relations $\bowtie$ and $=$ are decidable then $\models$ and $\not\models$ are decidable too.

Notice that if χ and χ' differ only on variables not appearing in ϕ, then $\kappa \models_\chi \phi \Leftrightarrow \kappa \models_{\chi'} \phi$. Hence, we can only consider finite assignments over the variables of ϕ.

Example 7. Let us consider the following authentication property.

$$\begin{aligned}&\phi = \forall\beta : B.\exists\sigma : S.\exists\alpha : A.\\ &\quad (V@\sigma = \beta \wedge U@\sigma = \alpha \wedge X@\alpha = \beta) \rightarrow (t@\sigma = T^a@\alpha \wedge s@\beta = T^s@\sigma \wedge r@\sigma = k^{ab}@\alpha).\end{aligned}$$

The formula states that, whenever B terminates, a server S and an initiator A (that aimed at interacting with B through S) have also took part to the session. In this case, the nonce received by S is the one generated by A, while B receives the nonce generated by S. Finally, the session key received by S must be the key associated to T^a by A.

We show that the final configuration $\langle \{()[], ()[], ()[]\}, \kappa_1, \gamma_1 \rangle$ in Example 5 does not yield a model of ϕ. We have already pointed out that instance variables are quantified over principals that have been joined the context, hence ϕ is equivalent to:

$$\begin{aligned}&(V@S_2 = B_1 \wedge U@S_2 = A_3 \wedge X@A_3 = B_1)\\ &\rightarrow (t@S_2 = T^a@A_3 \wedge s@B_1 = T^s@S_2 \wedge r@S_2 = k^{ab}@A_3)\end{aligned}$$

where we have substituted instance variables with the corresponding names in κ_1. It is easy to see that under mapping γ_1 the antecedent holds while the consequent is false because $\gamma_1(s_1) = T^a_3$, hence $\kappa_1 \not\models_{\gamma_1} \phi$. $\diamond$

3.1 Connection Formulae

The join mechanism is the basic building block to connect principals properly within protocol sessions. It appears hence natural to augment its expressiveness by equipping it with a mechanism to state invariant properties on protocol principals: *Connection formulae.* For instance, the property may state the hypothesis on the usage of keys among principals.

Example 8. A desirable connection of WMF principals in Example 4 is expressed by the following formula:

$$\phi^j = \forall\zeta : S.(\exists\alpha : A.(U@\zeta = \alpha \rightarrow y^a@\zeta = x^{as}@\alpha) \wedge (V@\zeta = \alpha \rightarrow y^b@\zeta = x^{as}@\alpha))$$
$$\wedge(\exists\beta : B.(U@\zeta = \beta \rightarrow y^a@\zeta = z^{bs}@\beta) \wedge (V@\zeta = \beta \rightarrow y^b@\zeta = z^{bs}@\beta)).$$

The formula states that "correct" joins of instances connect symmetric keys of initiator to y^a and those of the responder to y^b; notice that A and B can play both roles. $\diamond$

By taking advantage of such formulae, we can also formalise implicit conditions of the protocol. For instance, we can add to ϕ^j the condition that $y^a@\zeta \neq y^b@\zeta$ stating that the key shared by A and B with S must be different.

Notice that the operational semantics now generates all the contexts satisfying the property. This significantly reduces the state space.

3.2 Symbolic Semantics

The semantics of cIP prescribes that input actions with binding variables force the intruder to generate a matching message from its knowledge. Since the number of messages derivable from a non-empty knowledge is infinite (e.g. if $\kappa \bowtie m$, then $\kappa \bowtie m, m$), the operational semantics generates an infinite state space. Symbolic techniques provide a powerful mechanism to tackle this problem: We introduce the *symbolic variable* $x(\kappa)$, whenever an infinite set of messages can be generated for a binding variable x. This amounts to say that x can assume any message m such that $\kappa \bowtie m$. The value of $x(\kappa)$ will be possibly set by matching later actions in the trace. Moreover, the evolution of the computation will possibly impose further constraints which could restrict the set of values for x. As observed in [5, 17], output of regular principals can be anticipated without any loss of significant traces. The intruder modelled is eager since he learns as much as possible from messages sent by regular principals. When all output messages have been collected, the intruder tries to generate messages for waiting principals. The symbolic semantics enables us to limit, as much as possible, the dimension of traces. Notice that this is not a trivial task, because we need to avoid cutting off traces that lead to attacks. Therefore, the symbolic semantics generates only those traces where output actions are fired in advance with respect to the input ones. More precisely, whenever we have to generate the successors of a state we check if it contains output actions. In such a case, we have only one successor which is the state obtained performing all output actions at once. Otherwise, we have a finite number of successors for each input action.

The symbolic state space can be further reduced by adding "type" information to symbolic variables. For instance, if we know that x stands for a message of type t, we substitute $x(\kappa)$ with $x^t(\kappa)$, where t ranges on $\{P, O, pb, pr, sy, *\}$ (respectively denoting 'principal names', 'nonce', 'public', 'private', 'symmetric'). For example, let us consider the principal $A \triangleq ()[in(?x).in(\{na\}_x)]$. Whenever the intruder has to forge a message matching with $?x$, a symbolic variable $x(\kappa)$ is sent. Afterwords, the second action imposes a constraint on $x(\kappa)$ because the intruder 'learns' that x is intended to be a key, hence we consider only those transitions where a key in κ is substituted for x.

4 $\mathcal{A}$SPASyA

The previous sections reviewed the features of our symbolic model for verifying cryptographic protocols. We now show how the symbolic model has been exploited as a basis for the design and development of an effective and usable verification toolkit called $\mathcal{A}$SPASyA. $\mathcal{A}$SPASyA is written in ocaml and has a modular architecture, where each module encapsulates a single aspect of the symbolic model.

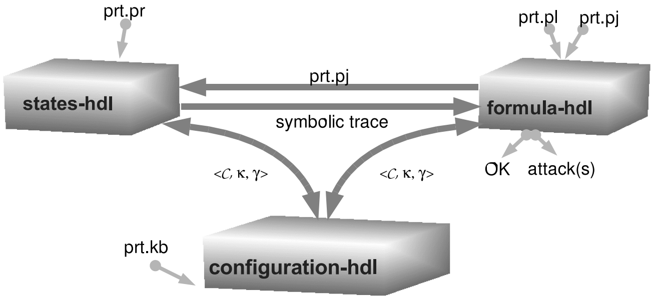

Fig. 1. $\mathcal{A}$SPASyA architecture

4.1 Architecture

The architecture of $\mathcal{A}$SPASyA is displayed in Figure 1 and it is made of three main groups. The group `configuration-hdl` manages cIP configurations (and, therefore, $\mathcal{PL}$ models) and it consists of components:

- `context` for representing instances,
- `knowledge` that manages the set of messages in the configurations and,
- `assignments` handling with variable substitutions.

Group `states-hdl` implements the state space generation and contains components:

- `step` that obtains the next state given the current one, and
- `join` that handles the introduction of new principals inside the context.

Finally, `formula-hdl` implements checking of security properties and resolution of corresponding constraint systems. The modules of `formula-hdl` are:

- `logic` that transforms a $\mathcal{PL}$ formula in an equivalent normalised formula;
- `verifier` that checks whether a formula holds in a given configuration;
- `csolver` that collects constraints on symbolic variables and resolves equality/inequality of symbolic messages.

In a verification session, the user provides the cIP specification of the protocol (file `prt.pr`) equipped with the connection formula (`prt.pj`) and the $\mathcal{PL}$ formula expressing the security property (`prt.pl`). A non-empty initial knowledge (`prt.kb`) and the maximum number of instances that can join a context can be specified, as well.

Modules `step` and `join` ask to `configuration-hdl` for the current state. According to cIP (symbolic) semantics and to `prt.pj`, a new configuration is produced and returned back to `configuration-hdl`. When `configuration-hdl` receives a final configuration, it forwards the corresponding $\mathcal{PL}$ (symbolic) model to `verifier` that together with `csolver` checks for the validity of (the normalisation of) `prt.pl`. The verification process is made on a state that may contain symbolic variables. Resolving symbolic atoms via unification leads to an infinite number of assignments. To cope with this problem $\mathcal{A}$SPASyA uses a symbolic constraint solver. Modules `verifier` and `csolver` return `OK` when the formula holds, otherwise they yield the (possible) attack(s)[5]. The attacks are reported together with the conditions they violate.

4.2 Verifying with $\mathcal{A}$SPASyA

Verifying a protocol with $\mathcal{A}$SPASyA is a four-step procedure: (*i*) Each role is formalised by a cIP principal, (*ii*) the security property is specified with a $\mathcal{PL}$ formula, (*iii*) conditions on connections are specified with a $\mathcal{PL}$ formula and, (*iv*) initial knowledge is tuned.

[5] The user can specify if $\mathcal{A}$SPASyA must stop at the first attack found or explore the whole state space.

cIP specification Principals of the WMF protocol can be translated in a formal cIP specification as in Example 4. The basic idea of this step is to provide a cIP process for each principal corresponding to a role in the protocol. At each step, its behaviour depends on whether the principal is sending or receiving a message.

Generally speaking, the protocol designer must specify open variables of principal. Even though it seems that no algorithm exists to (completely) define cIP principals from their informal specifications, we can give some rule of thumbs:

- Initiator usually needs an open variable which the responder should join;
- if the identity of the partner is acquired in a communication no open variable is required from that partner (unless checks are required);
- an open variable might be necessary when a principal must interact with a server.

Security Properties An interesting authentication property for the WMF protocol states that principal B can safely assume that the session key received from the server S has been generated by an initiator A (and that A intended to use the key with B). This property is represented by the formula:

$$\begin{aligned} \phi = {} & \forall \beta : B.\exists \sigma : S.\exists \alpha : A. \\ & (V@\sigma = \beta \wedge U@\sigma = \alpha \wedge X@\alpha = \beta) \rightarrow (t@\sigma = T^a@\alpha \wedge s@\beta = T^s@\sigma \wedge r@\sigma = k^{ab}@\alpha). \end{aligned}$$

Given the previous cIP specification and the authentication property, $\mathcal{A}$SPASyA generates the state space and tries to satisfy ϕ in any final state. The result is a trace where ϕ does not hold and where principals are connected with

$$\gamma_0 = \begin{cases} z_1^{bs}, x_3^{as}, y_2^a, y_2^b & \mapsto k, \\ X_3 & \mapsto X_3(\kappa_0), \\ V_2 & \mapsto V_2(\kappa_0), \\ U_2 & \mapsto U_2(\kappa_0). \end{cases}$$

Let S_2' be $()[out(\{T_2^s, U_2(\kappa_o), k_3^{ab}\}_k)]$, then the following trace is found:

$$\begin{aligned} & \langle \{A_3, S_2, B_1\},\ \emptyset, \gamma_0 \rangle \\ & \longmapsto \langle \{()[], S_2, B_1\}, \{A_3, \{T_3^a, X_3(\kappa_0), k_3^{ab}\}_k\}, \gamma_0 \rangle \\ & \longmapsto \langle \{()[], S_2', B_1\}, \{A_3, \{T_3^a, X_3(\kappa_0), k_3^{ab}\}_k\}, \gamma_0[{}^{T_3^a, X_3(\kappa_0), k_3^{ab}}/_{t_2, V_2(\kappa_0), r_2}] \rangle \\ & \longmapsto \langle \{()[], ()[], B_1\}, \kappa_1, \gamma_0 \rangle \\ & \longmapsto \langle \{()[], ()[], ()[]\}, \kappa_1, \gamma_1 \rangle, \end{aligned}$$

where κ_1 is the set $\{A_3, \{T_3^a, X_3(\kappa_0), k_3^{ab}\}_k, \{T_2^s, U_2(\kappa_0), k_3^{ab}\}_k\}$ and γ_1 is the mapping $\gamma_0[{}^{T_3^a, X_3(\kappa_0), k_3^{ab}, T_3^a, X_3(\kappa_0), k_3^{ab}}/_{t_2, V_2(\kappa_0), r_2, s_1, x_1, w_1}]$.
When checking $\kappa_1 \models_{\gamma_1} \phi$, `verifier` and `csolver` instantiate the trace above with the "concretizing" substitution $\gamma = [{}^{B_1, B_1, A_3}/_{X_3(\kappa_0), V_2(\kappa_0), U_2(\kappa_0)}]$. Since $\kappa_1\gamma \not\models_{\gamma_1\gamma} \phi$, an attack is found and the above trace is returned together with the violated condition of ϕ, namely $t@\sigma \neq T^a@\alpha$.

Handling Intruder Knowledge Specifying the initial knowledge of the intruder is a powerful device to test a protocol under weaker conditions. It is used mainly for two purposes:

- the intruder knows some secrets (e.g. compromised keys);
- Let the intruder know something about past interactions between principals (cryptograms exchanged in previous sessions).

The latter is especially useful in finding *replay attacks* where the intruder makes a principal accepting a message generated for him in a previous session.

4.3 The Needham-Schroeder Protocol

This section applies the verification methodology of $\mathcal{A}$SPASyA to the Needham-Schroeder protocol. The purpose is to emphasize the flexibility and the expressiveness of $\mathcal{A}$SPASyA. The informal specification together with its corresponding cIP formalisation are given below:

$$\begin{array}{ll} (1)\ A \to B : \{na, A\}_{B+} & A \triangleq (y)[out(\{na, A\}_{y^+}).in(\{na, ?u\}_{A^-}).out(\{u\}_{y^+})] \\ (2)\ B \to A : \{na, nb\}_{A+} & \\ (3)\ A \to B : \{nb\}_{B+}. & B \triangleq ()[in(\{?x, ?z\}_{B^-}).out(\{x, nb\}_{z^+}).in(\{nb\}_{B^-})]. \end{array}$$

The protocol has been designed to achieve reciprocal authentication of A and B by exchanging secret nonces na and nb. At step (3), B receives the last message and assumes that A has been its partner in the protocol session where nonce nb was generated: A desirable property can be formalised with the following $\mathcal{PL}$ formula:

$$\begin{array}{l} \forall\alpha : A.(na@\alpha \in \mathfrak{K} \to x@\alpha = \mathbf{I}) \\ \wedge \\ \forall\beta : B.((nb@\beta \in \mathfrak{K} \to z@\beta = \mathbf{I}) \vee (\exists\alpha : A.z@\beta = \alpha \to x@\alpha = \beta)). \end{array}$$

The first condition states that nonces na generated by instances of A must remain secret, unless they were intended for $\mathbf{I}$. The second condition states that, for each instance β of B, either the intruder obtains $nb@\beta$ because he initiated the protocol session with β as responder ($z@\beta = \mathbf{I}$) or, if β assumes that he communicated with an instance of A, α, ($z@B = \alpha$) then instance α wanted to communicate with β ($z@\alpha = \beta$). (For simplicity, we do not impose any constraint on connections and assume the empty initial knowledge.)

$\mathcal{A}$SPASyA returns two attacks that violate the security property above: One is the well-known Lowe attack [12], the other is a subtle type flaw. We report only the informal description of the type flaw attack[6]. Since no constraints are imposed to connections, the context made only of two instances of B (B_1 and B_2) is generated by `join`. A possible trace of this context is:

$$\begin{array}{l} (1)\ I \to B_1 : \{I, B_2\}_{B_1^+} \\ (2)\ B_1 \to I : \{I, nb_1\}_{B_2^+} \\ (3)\ I \to B_2 : \{I, nb_1\}_{B_2^+} \\ (4)\ B_2 \to I : \{nb_1, nb_2\}_{I+} \\ (5)\ I \to B_2 : \{nb_2\}_{B_2^+} \\ (6)\ I \to B_1 : \{nb_1\}_{B_1^+} \end{array}$$

where the intruder interleaves two sessions with B_1 and B_2 where he plays as initiator. Step (1) contains a type flaw, since the I exploits its own identity as a nonce so that B_1 "thinks" B_2 started a protocol session as initiator. Thence, B_1 replies the nonce challenge with message (2), that I forwards to B_2 to make him/her generate message (4). The intruder can now decrypt message (4), therefore secrecy of nb_1 is violated ($nb_1 \notin \mathfrak{K}$) even though $z@B_1 \neq \mathbf{I}$. Notice that STA and TRUST would have required *two* different verification sessions to discover the two attacks, changing protocol specification.

5 Discussion

The attack of the WMF protocol is based on the fact that A and B share the same key with S. Of course, in symmetric cryptography it is assumed that this cannot happen. Our aim, however, is to show that this assumption can be expressed in our framework *without* changing neither cIP principals nor security formulae. Indeed, we can simply add to the

[6] We refer the reader to `http://www.di.unipi.it/~etuosto/aspasya.html` for the details.

connection invariant the condition $y^a@\sigma \neq y^b@\sigma$. This shows that the $\mathcal{PL}$ formula that drives conditions can also be used for stating implicit assumptions on protocols.

The type flaw attack to the Needham-Schroeder protocol is more subtle; however, we want just remark that $\mathcal{A}$SPASyA can find different attacks by checking only one security property. Hence, it is also useful that indications on the conditions that are violated by the reported attacks are pinpointed.

The most computationally expensive tasks in $\mathcal{A}$SPASyA are state space generation and checking of formulae. We adopted some programming tricks to tune our algorithms with respect to the time spent in computation and memory usage. The main problems related to the handling of (very large) transition systems are memory consumption and trace pruning. These problems have been considered by other authors [18, 3, 4] as well. Our framework extends these approaches by taking advantages of the join mechanism and the implementation of symbolic variable to reduce the state space. Notice that since we are interested in checking a formula when we reach a terminal trace, it is not necessary to maintain in memory all the state space but only the current trace. This has been accomplished using a depth-first search strategy that coupled with the efficient ocaml garbage collector enables us to use a little amount of memory (this also has impact on time efficiency since we gain speed for the lack of page swapping)

References

1. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.
2. Roberto Amadio, Denis Lugiez, and Vincent Vancackére. On the symbolic reduction of processes with cryptographic functions. Technical report, INRIA-Sophia, 2001.
3. Michele Boreale. Symbolic trace analysis of cryptographic protocols. In *28th Colloquium on Automata, Languages and Programming (ICALP)*, LNCS. Springer, July 2001.
4. Michele Boreale and Marzia Buscemi. A framework for the analysis of security protocols. In *CONCUR: 13th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2002.
5. Andrea Bracciali. *Behavioural Patterns and Software Composition*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2003.
6. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
7. Somesh Clarke, Edmund M. Jha and Wilfredo R. Marrero. Using state space exploration and a nautural deduction style message derivation engine to verify security protocols. In *In Proc.* IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998*, 1998.
8. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEETIT: IEEE Transactions on Information Theory*, 29, 1983.
9. P. Marcelo Fiore and Martín Abadi. Computing symbolic models for verifying cryptographic protocols. In *Computer Security Foundations Workshop*, 14th IEEE Computer Security Foundations Workshop, pages 160–173, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.
10. Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol — version 3.0. Available at `http://home.netscape.com/eng/ssl3/ssl-toc.html`, March 1996.
11. Dan Harkins and Dave Carrel. RFC 2409: The Internet Key Exchange (IKE), November 1998. Status: PROPOSED STANDARD.
12. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
13. Alfred J. Menzies, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
14. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.
15. John C. Mitchell, Mark Mitchell, and Ulrich Ster. Automated analysis of cryptographic protocols using murϕ. In *10th IEEE Computer Security Foundations Workshop, IEEE Press*, pages 141–151, 1997.

16. Douglas R. Stinson. *Cryptography: Theory and practice.* CRC Press, 1995.
17. Emilio Tuosto. *Non-Functional Aspects of Wide Area Network Programming.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Via Buonarroti 2, 56125 Pisa - Italy, May 2003. TD-8/03, SEU-Servizio Editoriale Universitario di Pisa. Available at `http://www.di.unipi.it/phd/tesi/tesi_2003/PhDthesis_Tuosto.ps.gz`.
18. Vincent Vanackére. The trust protocol analyser. automatic and efficient verification of cryptographic protocols. In *VERIFY02*, 2002.
19. Tatu Ylonen. SSH — secure login connections over the Internet. In USENIX Association, editor, *6th USENIX Security Symposium, July 22–25, 1996. San Jose, CA*, pages 37–42, Berkeley, CA, USA, July 1996. USENIX.