A Method for Automatic Cryptographic Protocol Verification

Jean Goubault-Larrecq

G.I.E. Dyade & Projet Coq, Inria, France (`Jean.Goubault@dyade.fr`)

Abstract. We present an automatic, terminating method for verifying confidentiality properties, and to a lesser extent freshness properties of cryptographic protocols. It is based on a safe abstract interpretation of cryptographic protocols using a specific extension of tree automata, $\vee$-parameterized tree automata, which mix automata-theoretic techniques with deductive features. Contrary to most model-checking approaches, this method offers actual security guarantees. It owes much to D. Bolignano's ways of modeling cryptographic protocols and to D. Monniaux' seminal idea of using tree automata to verify cryptographic protocols by abstract interpretation. It extends the latter by adding new deductive abilities, and by offering the possibility of analyzing protocols in the presence of parallel multi-session principals, following some ideas by M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi.

1 Introduction

When secrets are to be preserved, or authenticity of messages is to be established notably, cryptography is often presented as the solution. Once mostly used by the military, cryptography is now being used in more and more application domains, pushed notably by electronic commerce: in banking terminals, payment terminals, smart cards (credit cards, phone cards, subscriber identity modules in GSM mobile phones, etc.), in security servers as used to build corporate extranets, and so on.

By *cryptography* we mean a corpus of algorithmic techniques including encryption and decryption algorithms, of various strengths and efficiencies, but also cryptographically secure hash functions, pseudo-random number generators, key generation algorithms, and so on [30]. However, it has been well-known for some time now [9] that the use of cryptographic techniques, even provably secure ones, is not enough to guarantee any sufficient level of security. To take a silly but common example, imagine you wish to mail a message M (the *plaintext*) to some other person B while keeping M *secret*, that is, undisclosed to third parties. So you encrypt it using a secure algorithm and a symmetric key K, getting the *ciphertext* $\{M\}_K$. The problem is, B needs K to be able to decrypt $\{M\}_K$ and get back M. So you decide to send K to B in a separate mail—in clear. It is then all too easy for an intruder to get K and decipher $\{M\}_K$, so secrecy is broken. The same intruder can also forge new messages $\{M'\}_K$ that B will believe come from you, so authentication fails.

This is why you have to design better, and somewhat more involved exchanges of messages to try and achieve secrecy or authentication. Such exchanges are called *cryptographic protocols*. Nonetheless, and despite the ingenuity of their designers, most cryptographic protocols have been found to be faulty later on, and it is now notorious that designing such protocols is an error prone process. Furthermore, the attacks again faulty protocols are most often of a purely logical nature: that is, attacks can be described independently of the actual cryptographic means used, by scenarios describing how an intruder replays old messages, forges new messages, gets keys and uses them to decrypt yet other messages in order to fool the actors of a protocol (the so-called *principals*). Such attacks are in a sense *silly* design faults, but most are nonetheless difficult to spot, and it is even more difficult to guarantee that no logical (i.e., silly) attacks against a given protocol are possible.

Our goal is to present an automated technique for guaranteeing the absence of logical faults in cryptographic protocols, modulo a set of assumptions about the outside world. In this paper, we concentrate on secrecy, which is both an important property and a requirement for other analyses. Our paper is certainly not the first such attempt, and we survey the relevant literature in Section 2. The main sources of inspiration of this work are Dominique Bolignano's wonderfully simple ideas on modeling cryptographic protocols [4], David Monniaux' brilliant idea to use tree automata to compute upper approximants of the states of knowledge of intruders [24], and Mourad Debbabi *et al*'s idea of modeling honest principals as accomplices of intruders [12]. Most of the techniques described here originate from their works—the novelty is in a series of choices. First, instead of using actual tree automata, we use a slight extension which we call $\vee$-*parameterized tree automata* ($\vee$PTAs), and which will allow us to incorporate some form of deductive reasoning into an otherwise descriptive approach. We shall describe $\vee$PTAs in Section 3. We shall then describe how $\vee$PTAs enable us to represent and compute states of knowledge in Section 4, and apply this to the concrete and abstract semantics for a small formal cryptographic protocol description language in Section 5. The abstract semantics, which we shall prove correct with respect to the concrete semantics, will always provide us an upper approximation of the knowledge state of the intruder. Hence, if our abstract interpreter returns that key K is secret at a given program point, then it must indeed be so. A novelty here, compared to David Monniaux' work, is our way of modeling parallel multi-session principals; although it may seem particularly rough, it is in fact precise enough in practical cases. We report on practical experience with these techniques in Section 6, showing its practical value, and shedding light on its strengths and weaknesses. We discuss future perspectives and possible improvements in Section 7.

2 Related Works

Early work on cryptography and security concentrated on designing strong cryptographic algorithms, and the question of the security of cryptographic *protocols* only appeared in the 1980s. The seminal paper here is [9], which identified a number of problems and proposed a set of recipes formulated as rules for a logical language—now known as the *BAN logic*—for modeling and analyzing cryptographic protocols. Although it was largely criticized, it paved the way for research in assessing the security of cryptographic protocols.

Doing so is a hard task if all interactions between the protocols proper and the cryptographic algorithms are to be taken into account. For example, the RSA public-key encryption algorithm [30] is prone to multiplicative attacks, which exploit the fact that encrypting the product of two messages gives the product of the encryption of each message separately. To guard against such attacks, sophisticated tools from abstract complexity theory are required; this is doable in some cases [2].

The easiest way to formalize cryptographic protocols, however, is to follow [9] and assume that the protocols rest on *perfect cryptographic primitives*. This means that ciphertexts $\{M\}_K$ and $\{M'\}_{K'}$ cannot be confused if $M \neq M'$ or $K \neq K'$, that ciphertexts cannot be confused with non-ciphertexts (e.g., tuples), that you cannot decrypt ciphertexts by repeated encryption ($\{\ldots\{\{M\}_{K_1}\}_{K_2}\ldots\}_{K_n} \neq M$), and so on. This is not only simpler—messages become members of a free algebra generated by encryption and tupling primitives—but is also realistic, in that good engineering practice dictates to combine raw cryptographic primitives like RSA with disambiguating formats like ASN.1 BER coding [17] to ensure that cryptography will be as perfect as possible. Results like [2] also provide perfect cryptographic primitives from imperfect ones—with very high probability and under standard complexity-theoretic assumptions.

Having assumed perfect cryptographic primitives, there are still several ways to assess the security of cryptographic protocols. The pioneering paper [9] proposed to use a modal logic of belief, trust relationships and jurisdiction notions. This was used to formalize former reasoning principles on cryptographic protocols, in a more systematic way. Nonetheless, it was highly criticized and several authors proposed refinements of the approach, culminating in approaches such as [3]. Although this does not seem to be well-known, such approaches can be automated [16, 25].

Model-checking was also proposed as a way to explore security aspects of cryptographic protocols. Several attacks were discovered on existing protocol in this way [18, 19]. Model-checking-based systems include Lowe's FDR [19], Meadows' NRL [21, 22] and others [20]. Model-checkers enable us to find attacks, but in case no attack is possible, they usually cannot give any security guarantee. One can argue that this is not a serious defect, as most systems that resist a thorough test by model-checking are usually correct. However, industrial producers of security products want more than a likelihood of flawlessness; this is particularly obvious in the context of TCSec, ITSec and now Common Criteria certification (for the latter, at levels 5 and up) [15] where the use of formal methods is mandated in the hope of providing sufficient guarantees of quality. Nonetheless, model-checking tools may be used to give actual guarantees under a few assumptions, using formally proven reductions to finite-state cases [31].

Standard formal methods based on mechanical computer-assisted proof development have to date been the most successful techniques to give actual security guarantees for cryptographic protocols [4, 27]. This has been extended to the more complex case of electronic commerce protocols, where not only secrecy and authentication properties matter, but also non-duplication, non-repudiation and various others. It is also helpful to mix classical formal methods with abstract interpretation techniques [6] to reduce the state space to explore, in some cases even to finite state spaces.

It turns out that abstract interpretation (see [11] for a good introduction) works well enough without any deductive formal means and provide for a completely automated solution to the verification of cryptographic protocols. The pioneer here is D. Monniaux [24], who proposes to use tree automata to model the set of messages that intruders are able to construct at each step of any given protocol. The current paper is based on a very similar idea, and extends it to be able to deal with explicit assumptions about the external world, using an extension of tree automata that integrates some deductive abilities ($\vee$PTAs). We stress the fact that these methods, although they seem to be similar to model-checking techniques in that they all compute fixpoints that approximate the behavior of protocols, achieve the opposite goal: if no attack is found by such tools, then there cannot be any (under some reasonable assumptions), however is some possible attack is found, then there might not be any actual attack. So such methods can be of some help on the path to Common Criteria certification.

Tree automata have also been used in a rather different approach [13] to cryptographic protocol verification, where cryptographic protocols are modeled via rewrite systems, and the set of terms to which a given term may rewrite is approximated by tree automata. As in [9], it seems that rewrite systems describing protocols have to be designed by hand in each particular case, through an idealization step that might be error-prone; while the formal method and abstract interpretation approaches above take an actual description, only very slightly idealized, of the protocols to be assessed.

Another markedly different, automated technique—but not necessarily terminating—was proposed by M. Debbabi *et al* [12] based on the observation that honest principals can be seen, from the security viewpoint, as accomplices of intruders (despite their good will); e.g., if some key server expects to read some message M and answers with $\{M\}_K$, then intruders may benefit from this behavior to gain the ability to encrypt messages using K (just send a message to the server), although K might remain unknown to them. This approach is not fine enough to deal with protocols where only one sequential principal possesses K, but it is a great idea to model multi-session principals, and we shall adapt it in this case.

We finish this tour of methods for evaluating the security of cryptographic protocols with perfect cryptographic means by Gordon and Abadi's *spi-calculus* [1]. The nice thing about the latter is its clear semantic foundation, based on a process algebra extending Milner *et al*'s π-calculus [23] to integrate cryptographic term and pattern primitives. (Modeling decryption by pattern-matching against an encryption term is a commonality between [4, 27] and the present work. The other standard way, used in [13, 24] and other works, is to include an explicit decryption primitive inside terms, together with equations stating that decrypting the encrypted message $\{M\}_K$ with the inverse key to K yields M.) Because of the clear semantics of the spi-calculus, confidentiality and authentication properties have well-defined and indisputable meanings, defined through observational equivalences (to be precise, variants of may-testing equivalence). Finding computable criteria for ensuring confidentiality or authentication is still the subject of on-going work. One of the most promising solutions is Abadi's typing system for secrecy, which is not only safe (if a program type-checks, then it does not leak any of its secret inputs, namely the ones of type Any) but also easily implementable. It turns out, as Abadi explains, that the notion of secret in the spi-calculus is much stronger that the one used in [4, 27, 24]. Whereas M is secret in the latter works if no intruder can build the message M, M is secret in the spi-calculus if no intruder can observe any property of M, including the fact that it is equal or not to a previous message M'. In particular, the couple (M_1, M_2) is secret in the former approach if and only if M_1 or M_2 is secret, whereas if it is secret in the latter approach then both M_1 and M_2 are secret. This places secrecy in the spi-calculus at the high end of demands for security; to start with, only protocols that systematically mix messages with *confounders* before encryption can be proved secure. It seems that the less demanding notion of secret used in other approaches is enough in most cases today, and allows one to validate some existing protocols.

3 $\vee$-Parameterized Tree Automata

$\vee$-Parameterized tree automata ($\vee$PTAs) are a way to combine two different styles of representations of sets. The first is *tree automata* [10], which represent sets of first-order *terms*, a.k.a. trees, and can be roughly described as labeled multi-graphs recognizing terms by running them against the structure of the multi-graph. The second is *propositional logic*: conjunction is interpreted non-standardly as intersection, disjunction is union, and so on; this will allow us to do some basic reasoning on unspecified sets (whereas automata only describe fully specified sets). We recall what terms and propositional formulae are, and proceed to define $\vee$PTAs.

3.1 Terms

We consider a typed, or many-sorted term algebra: this is both general enough and interesting in practical implementations. Assume a set $\mathcal{T}$ of *types*; our types will include the type `msg` of *messages*, the type `msglist` of *lists of messages*, and a few others. We denote types by letters of the form τ, possibly subscripted or primed. Let $\mathcal{F}$ a set of so-called *function symbols*. A first-order *signature* Σ over $\mathcal{F}$ is a map from $\mathcal{F}$ to the set of expressions of the form $\tau_1 \times \ldots \times \tau_n \to \tau$, where $n \in \mathbb{N}$ and $\tau_1, \ldots, \tau_n, \tau$ are types.

Let $\mathcal{X}_\tau$, for each type τ, be pairwise disjoint non-empty sets, disjoint from $\mathcal{F}$. ($\mathcal{X}_\tau$ is classically thought as a set of variables of type τ, but we shall understand them as just basic elements of type τ.) Let $\mathcal{X}$ denote the family $(\mathcal{X}_\tau)_{\tau \in \mathcal{T}}$. The set of *terms of type* τ $T_\tau(\Sigma, \mathcal{X})$ is defined simultaneously for all types τ as the smallest set containing $\mathcal{X}_\tau$ and such that for each $f \in \mathcal{F} = \operatorname{dom} \Sigma$, if $\Sigma(f) = \tau_1 \times \ldots \times \tau_n \to \tau$ and $t_1 \in T_{\tau_1}(\Sigma, \mathcal{X})$, ..., $t_n \in T_{\tau_n}(\Sigma, \mathcal{X})$, then the $(n+1)$-tuple $(f, t_1, \ldots, t_n)$ is in $T_\tau(\Sigma, \mathcal{X})$. For legibility, we write the latter $f(t_1, \ldots, t_n)$, or just f if $n = 0$. We shall also sometimes use particular pieces of syntax for particular function symbols. Note that, since $\mathcal{X}_\tau$ is non-empty, $T_\tau(\Sigma, \mathcal{X})$ is non-empty, too.

3.2 Formulae and Sets

We use propositional formulae, up to logical equivalence, to represent (some) sets of terms. Let $\mathcal{A}_\tau$, for each type τ, be a set of so-called *logical variables* of type τ. The intent is that each logical variable α of type τ denotes a set of terms of type τ. *Propositional formulae* F of type τ are defined by the grammar :

$$F ::= A \mid F \wedge F \mid F \vee F \mid \neg F \mid \mathbf{0} \mid \mathbf{1}$$

where A ranges over $\mathcal{A}_\tau$.

The *standard, Tarskian semantics* of formulas of type τ is given as follows. A *standard logical environment* ρ^s_τ is a map from $\mathcal{A}_\tau$ to the set $\mathbb{B} \hat{=} \{\top, \bot\}$. (We use $\hat{=}$ for *equality by definition*. $\top$ denotes truth, $\bot$ denotes falsehood.) Then, we define the predicate $\rho^s_\tau \models F$ ("ρ^s_τ satisfies F") by structural induction on F by: $\rho^s_\tau \models A$ if and only if $\rho^s_\tau(A) = \top$, $\rho^s_\tau \models F_1 \wedge F_2$ if and only if $\rho^s_\tau \models F_1$ and $\rho^s_\tau \models F_2$, $\rho^s_\tau \models F_1 \vee F_2$ if and only if $\rho^s_\tau \models F_1$ or $\rho^s_\tau \models F_2$, $\rho^s_\tau \models \neg F$ if and only it is not the case that $\rho^s_\tau \models F$, it is never the case that $\rho^s_\tau \models \mathbf{0}$, and it is always the case that $\rho^s_\tau \models \mathbf{1}$.

Two formulae F and G of type τ are *equivalent* if and only if, for every ρ^s_τ, $\rho^s_\tau \models F$ if and only if $\rho^s_\tau \models G$. This defines an equivalence relation $\equiv$ on formulae. We shall reason on formulae up to $\equiv$ in the next sections. In practice, this is easily implemented by representing formulae up to $\equiv$ by binary decision diagrams (BDDs) notably (see [8]), by which we mean *reduced, ordered* BDDs, or any improvement.

However, we shall use a non-standard semantics for formulae, where formulae denote sets of terms rather than booleans. Let the *environments* of type τ, ρ_τ, be the functions from $\mathcal{A}_\tau$ to $T_\tau(\Sigma, \mathcal{X})$, and define the semantics $[\![F]\!]\rho_\tau$ of the formula F of type τ in ρ_τ, by structural induction on F, as follows:

$$\begin{array}{rlrl}
[\![A]\!]\rho_\tau & \hat{=}\ \rho_\tau(A) & [\![\neg F]\!]\rho_\tau & \hat{=}\ T_\tau(\Sigma, \mathcal{X}) \setminus [\![F]\!]\rho_\tau \\
[\![F_1 \wedge F_2]\!]\rho_\tau & \hat{=}\ [\![F_1]\!]\rho_\tau \cap [\![F_2]\!]\rho_\tau & [\![\mathbf{0}]\!]\rho_\tau & \hat{=}\ \emptyset \\
[\![F_1 \vee F_2]\!]\rho_\tau & \hat{=}\ [\![F_1]\!]\rho_\tau \cup [\![F_2]\!]\rho_\tau & [\![\mathbf{1}]\!]\rho_\tau & \hat{=}\ T_\tau(\Sigma, \mathcal{X})
\end{array}$$

where $\setminus$ denotes set difference. We call *environment* ρ, any family $(\rho_\tau)_{\tau \in \mathcal{T}}$ of environments of each type τ.

It is sound to use formulae up to logical equivalence $\equiv$ to denote sets:

Lemma 1. *For any two formulae F and G of type τ, $F \equiv G$ if and only if $[\![F]\!]\rho_\tau = [\![G]\!]\rho_\tau$ for every environment ρ_τ of type τ.*

Proof. $\Rightarrow$: Fix $t \in T_\tau(\Sigma, \mathcal{X})$, and define a standard environment ρ^s_τ of type τ by letting $\rho^s_\tau(A) \hat{=} \top$ if and only if $t \in \rho_\tau(A)$. An easy structural induction on the formula H shows that: $(*)$ $t \in [\![H]\!]\rho_\tau$ if and only if $\rho^s_\tau \models H$. Moreover, t is arbitrary. Applying $(*)$ to the cases $H = F$ and $H = G$, it follows that if $F \equiv G$, then $[\![F]\!]\rho_\tau$ and $[\![G]\!]\rho_\tau$ have the same elements t, hence $[\![F]\!]\rho_\tau = [\![G]\!]\rho_\tau$.

$\Leftarrow$: Given a standard environment ρ^s_τ of type τ, define ρ_τ as mapping A to $\emptyset$ if $\rho^s_\tau(A) = \bot$, to $T_\tau(\Sigma, \mathcal{X})$ if $\rho^s_\tau(A) = \top$. An easy structural induction on H shows that: $(**)$ $\rho^s_\tau \models H$ implies $[\![H]\!]\rho_\tau = T_\tau(\Sigma, \mathcal{X})$, and $\rho^s_\tau \not\models H$ implies $[\![H]\!]\rho_\tau = \emptyset$. Since $T_\tau(\Sigma, \mathcal{X})$ is not empty, $(**)$ entails that $\rho^s_\tau \models H$ if and only if $[\![H]\!]\rho_\tau = T_\tau(\Sigma, \mathcal{X})$. If $[\![F]\!]\rho_\tau = [\![G]\!]\rho_\tau$ for every environment ρ_τ of type τ, then it follows that $\rho^s_\tau \models F$ if and only if $\rho^s_\tau \models G$. As ρ^s_τ is arbitrary, $F \equiv G$. $\square$

Of course, this works even if the sets we use are not sets of terms.

3.3 Automata

To deal with term structure, we define the following variant of tree automata. Compared to ordinary tree automata [10], ours integrate propositional formulae at states, and the states are typed (the latter helps in practice limit the size of automata, and does not restrict the generality of the approach). To simplify the following definition, extend Σ to $\mathcal{F} \cup \bigcup_{\tau \in \mathcal{T}} \mathcal{X}_\tau$ by letting $\Sigma(x) \hat{=} \to \tau$ for every $x \in \mathcal{X}_\tau$.

Definition 1 ($\vee$-Parameterized Tree Automata). *Let $\mathcal{Q}$ be a set of so-called* states *q. We assume that each state q has a* type *τ_q, and that $\mathcal{Q}$ contains infinitely many states of each type.*

An $\vee$-parameterized tree automaton, or $\vee$PTA, *of type τ_0, $\mathfrak{A}$, is a 4-tuple $(\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$, where:*

- *$\mathfrak{Q}$ is a finite subset of $\mathcal{Q}$, whose elements are the* states of $\mathfrak{A}$*;*
- *$\mathfrak{F} \subseteq \mathfrak{Q}$ is the set of* final states*, assumed disjoint from $\mathcal{F} \cup \bigcup_{\tau \in \mathcal{T}} \mathcal{X}_\tau$ and such that $\tau_q = \tau_0$ for every $q \in \mathfrak{F}$;*
- *$\mathfrak{B}$ maps each state $q \in \mathfrak{Q}$ to an $\equiv$-equivalence class of formulae of type τ_q (in the sequel, we shall just say formulae instead of $\equiv$-equivalence classes of formulae);*
- *$\mathfrak{R}$ is a set of rewrites rules $f(q_1, \ldots, q_n) \to q$, the* transitions*, where $f \in \mathcal{F} \cup \bigcup_{\tau \in \mathcal{T}} \mathcal{X}_\tau$ is such that $\Sigma(f) \hat{=} \tau_{q_1} \times \ldots \times \tau_{q_n} \to \tau_q$ ("transitions respect types").*

Ordinary tree automata are just $\vee$PTAs without the $\mathfrak{B}$ component. (As the following semantics will show, tree automata can also be equated with $\vee$PTAs where $\mathfrak{B}$ maps each state to the class of $\mathbf{0}$.) Now define the semantics of $\vee$PTAs, given an environment $\rho \hat{=} (\rho_\tau)_{\tau \in \mathcal{T}}$:

Definition 2 (Recognizing). *Let $\mathfrak{A} \hat{=} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$ be an $\vee$PTA of type τ_0. For any $q \in \mathfrak{Q}$ and any environment $\rho \hat{=} (\rho_\tau)_{\tau \in \mathcal{T}}$, for any term t of type τ_q, we say that $\mathfrak{A}$* recognizes t in ρ at q *if and only if:*

- *$t \in [\![\mathfrak{B}(q)]\!]\rho_{\tau_q}$, or:*
- *t is of the form $f(t_1, \ldots, t_n)$, and there is a transition $f(q_1, \ldots, q_n) \to q$ in $\mathfrak{R}$ such that t_j is recognized by $\mathfrak{A}$ in ρ at q_j for each j, $1 \leq j \leq n$.*

We let $[\![\mathfrak{A}]\!]\rho q$ be the set of all terms recognized by $\mathfrak{A}$ in ρ at q, and $[\![\mathfrak{A}]\!]\rho \hat{=} \bigcup_{q \in \mathfrak{F}} [\![\mathfrak{A}]\!]\rho q$ be the set of all terms recognized by $\mathfrak{A}$ in ρ.

This is a well-founded definition by structural induction on t. For any transition $f(q_1, \ldots, q_n) \to q$ where f is a variable x, we have $n = 0$ by assumption, and we take $f(t_1, \ldots, t_n) = f()$ to be x itself, by abuse of notation.

We shall sometimes present $\vee$PTAs by means of tree grammars. For example:

$$
\begin{array}{rll}
\checkmark\, q_0 & ::= \mathtt{k}(q_1) \mid \mathtt{c}(q_0, q_2) \mid \mathtt{c}(q_3, q_4) & : \mathtt{msg} \\
q_1 & ::= \mathtt{symk}(q_{12}) & : \mathtt{key} \\
q_2 & ::= \mathtt{symk}(q_9) & : \mathtt{key} \\
q_3 & ::= \mathtt{k}(q_2) \mid \mathtt{c}(q_3, q_2) & : \mathtt{msg} \\
q_4 & ::= & : \mathtt{key} \\
q_9 & ::= SK_0 \vee AK_0 & : \mathtt{K} \\
q_{12} & ::= SK_0 & : \mathtt{K}
\end{array}
$$

denotes the automaton of type $\mathtt{msg}$ with set of states $\mathfrak{Q} \hat{=} \{q_0, q_1, q_2, q_3, q_4, q_9, q_{12}\}$ (with types given by: $\tau_{q_0} = \tau_{q_3} = \mathtt{msg}$, $\tau_{q_1} = \tau_{q_2} = \tau_{q_4} = \mathtt{key}$, $\tau_{q_9} = \tau_{q_{12}} = \mathtt{K}$), $\mathfrak{F} \hat{=} \{q_0\}$

(as signaled by the check-mark ✓), $\mathfrak{R} \mathrel{\hat{=}} \{\mathtt{k}(q_1) \to q_0, \mathtt{c}(q_0, q_2) \to q_0, \mathtt{c}(q_3, q_4) \to q_0, \mathtt{symk}(q_{12}) \to q_1, \mathtt{symk}(q_9) \to q_2, \mathtt{k}(q_2) \to q_3, \mathtt{c}(q_3, q_2) \to q_3\}$, $\mathfrak{B} \mathrel{\hat{=}} \{q_0 \mapsto \mathbf{0}, q_1 \mapsto \mathbf{0}, q_2 \mapsto \mathbf{0}, q_3 \mapsto \mathbf{0}, q_4 \mapsto \mathbf{0}, q_9 \mapsto SK_0 \vee AK_0, q_{12} \mapsto SK_0\}$ (we do not write $\mathbf{0}$s explicitly in the tree grammar; note that $\mathfrak{B}(q_4) = \mathbf{0}$, so there is actually no term at all recognized at q_4 in any environment).

Assuming $\rho_{\mathtt{K}}(SK_0)$ contains the term k_0 and $\rho_{\mathtt{K}}(AK_0)$ contains k_1, the automaton above recognizes $\mathtt{c}(\mathtt{c}(\mathtt{k}(\mathtt{symk}(k_0)), \mathtt{symk}(k_1)), \mathtt{symk}(k_0))$. Indeed, the latter is recognized at state q_0 because of the entry $q_0 ::= \ldots \mid \mathtt{c}(q_0, q_2) \mid \ldots$, and $\mathtt{c}(\mathtt{k}(\mathtt{symk}(k_0)), \mathtt{symk}(k_1))$ is recognized at q_0, and $\mathtt{symk}(k_0)$ is recognized at q_2. The latter is because k_0 is recognized at q_9, since $q_2 ::= \mathtt{symk}(q_9)$ and $\mathfrak{B}(q_9) = SK_0 \vee AK_0$ has a semantics in environment ρ that contains k_0 by assumption. The former is because of the entry $q_0 ::= \ldots \mid \mathtt{c}(q_0, q_2) \mid \ldots$ again, and $\mathtt{k}(\mathtt{symk}(k_0))$ is recognized at q_0, and $\mathtt{symk}(k_1)$ is recognized at q_2. The latter is because the semantics of $\mathfrak{B}(q_9) = SK_0 \vee AK_0$ also contains k_1 by assumption, and the former is because of the entry $q_0 ::= \mathtt{k}(q_1) \mid \ldots$ and because $\mathtt{symk}(k_0)$ is recognized at q_1: indeed, k_0 is recognized at q_{12}, since the semantics of SK_0 contains k_0.

3.4 Operations on $\vee$PTAs

$\vee$PTAs work mostly like ordinary tree automata. We can compute unions of $\vee$PTAs exactly, and give upper approximants of their intersections, precise enough for the rest of the paper. We may also test them for inclusion or non-emptiness, given ρ. Additionally, we can consider whole classes of environments ρ specified by logical assumptions, i.e., by propositional formulae. We shall then be able to give sufficient conditions to establish that an $\vee$PTA is empty in whatever environment ρ satisfying the assumptions (*must-emptiness* test). This integrates some deductive abilities in the automata-based approach, and is our main reason for considering *parameterized* tree automata.

Lemma 2 (Union). *Let* $\mathfrak{A}_i \mathrel{\hat{=}} (\mathfrak{Q}_i, \mathfrak{F}_i, \mathfrak{R}_i, \mathfrak{B}_i)$*,* $i \in I$*, be any finite family of* $\vee$*PTAs of the same type* τ_0*. Define the* union $\bigsqcup_{i \in I} \mathfrak{A}_i$ *of the latter as the following automaton* $\mathfrak{A} \mathrel{\hat{=}} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$*. For each* $i \in I$*, for each* $q \in \mathfrak{Q}_i$*, let* $\langle i, q\rangle$ *be a state of type* τ_q*, chosen such that* $\langle i, q\rangle = \langle i', q'\rangle$ *implies* $i = i'$ *and* $q = q'$ *(i.e., create fresh copies of each state from each automaton). Let:*

- $\mathfrak{Q} \mathrel{\hat{=}} \{\langle i, q\rangle \mid i \in I, q \in \mathfrak{Q}_i\}$*;*
- $\mathfrak{F} \mathrel{\hat{=}} \{\langle i, q\rangle \mid i \in I, q \in \mathfrak{F}_i\}$*;*
- $\mathfrak{R} \mathrel{\hat{=}} \bigcup_{i \in I} \{f(\langle i, q_1\rangle, \ldots, \langle i, q_n\rangle) \to \langle i, q\rangle \mid f(q_1, \ldots, q_n) \to q \in \mathfrak{R}_i\}$*;*
- $\mathfrak{B}$ *maps each* $\langle i, q\rangle$*,* $i \in I$*,* $q \in \mathfrak{Q}_i$ *to* $\mathfrak{B}_i(q)$*.*

Then, for any environment ρ*,* t *is recognized by* $\mathfrak{A}$ *in* ρ *if and only if* t *is recognized by some* $\mathfrak{A}_i$*,* $i \in I$*, in* ρ*.*

We use the abbreviation $\mathfrak{A}_1 \sqcup \mathfrak{A}_2$ *to denote* $\bigsqcup_{i \in \{1,2\}} \mathfrak{A}_i$*.*

Proof. First observe that: $(*)$ every term t is recognized by $\mathfrak{A}$ in ρ at $\langle i, q\rangle$ if and only if it is recognized by $\mathfrak{A}_i$ in ρ at q. This is an easy structural induction on t. The lemma follows easily. □

Intersections cannot be computed exactly. Consider these two simple $\vee$PTAs:

$$\checkmark q_{10} \mathrel{\hat{=}} a : \tau_0 \qquad \checkmark q_{20} \mathrel{\hat{=}} X : \tau_0$$

where $\Sigma(a) = \to \tau_0$ and X is a propositional variable of type τ_0. The set of terms recognized by both in ρ is either $\{a\}$ if $\rho(X)$ contains a, or the empty set otherwise. But there is no way to represent this as an $\vee$PTA. Although there are ways to cure this, we won't need the extra complexity, and keep $\vee$PTAs as they are.

Lemma 3 (Intersection, Upper Approximation). *Let* $\mathfrak{A}_i \mathrel{\hat{=}} (\mathfrak{Q}_i, \mathfrak{F}_i, \mathfrak{R}_i, \mathfrak{B}_i)$*,* $i \in \{1,2\}$*, be two parameterized tree automata of the same type* τ_0*. Define the* infimum $\mathfrak{A}_1 \sqcap \mathfrak{A}_2$ *of the latter as the following automaton* $\mathfrak{A} \mathrel{\hat{=}} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$*. For any* $q_1 \in \mathfrak{Q}_1$*,* $q_2 \in \mathfrak{Q}_2$ *of the same type* τ*, choose a state* $[q_1, q_2]$ *of type* τ *in such a way that* $[q_1, q_2] = [q_3, q_3]$ *implies* $q_1 = q_3$ *and* $q_2 = q_4$*, and* $[q_1, q_2]$ *is distinct from every state* $\langle i, q\rangle$ *(see Lemma 2). Let:*

- $\mathfrak{Q}$ *be the set of all such states* $[q_1, q_2]$*, plus all states of the form* $\langle 1, q_1\rangle$*,* $q_1 \in \mathfrak{Q}_1$ *and all states of the form* $\langle 2, q_2\rangle$*,* $q_2 \in \mathfrak{Q}_2$*;*
- $\mathfrak{F} \mathrel{\hat{=}} \{[q_1, q_2] \in \mathfrak{Q} \mid q_1 \in \mathfrak{F}_1, q_2 \in \mathfrak{F}_2\}$*;*
- $\mathfrak{R}$ *is the union of:*
  1. $\{f([q_{11}, q_{21}], \ldots, [q_{1n}, q_{2n}]) \to [q_1, q_2]$
     $\mid [q_1, q_2] \in \mathfrak{Q}, f(q_{11}, \ldots, q_{1n}) \to q_1 \in \mathfrak{R}_1, f(q_{21}, \ldots, q_{2n}) \to q_2 \in \mathfrak{R}_2\}$,
  2. $\{f(\langle i, q_{i1}\rangle, \ldots, \langle i, q_{in}\rangle) \to [q_1, q_2]$
     $\mid [q_1, q_2] \in \mathfrak{Q}, i \in \{1,2\}, f(q_{i1}, \ldots, q_{in}) \to q_i \in \mathfrak{R}_i, \mathfrak{B}_{\bar{i}}(q_{\bar{i}}) \not\equiv \mathbf{0}\}$,
  3. $\{f(\langle i, q_{i1}\rangle, \ldots, \langle i, q_{in}\rangle) \to \langle i, q_i\rangle \mid i \in \{1,2\}, f(q_{i1}, \ldots, q_{in}) \to q_i \in \mathfrak{R}_i\}$

  where $\bar{i}$ *denotes* 2 *if* $i = 1$, 1 *if* $i = 2$*;*
- $\mathfrak{B}$ *maps each* $[q_1, q_2]$ *to* $\mathfrak{B}_1(q_1) \wedge \mathfrak{B}_2(q_2)$*, and each* $\langle i, q\rangle$ *to* $\mathfrak{B}_i(q)$*,* $i \in \{1,2\}$*,* $q \in \mathfrak{Q}_i$*.*

Then, for any environment ρ*, if* t *is recognized by* $\mathfrak{A}_1$ *and by* $\mathfrak{A}_2$ *in* ρ*, then* t *is recognized by* $\mathfrak{A}$ *in* ρ*.*

Proof. Observe that: $(*)$ the terms recognized by $\mathfrak{A}$ in ρ at $\langle i, q\rangle$ are those recognized by $\mathfrak{A}_i$ in ρ at q.

We show that, for every type τ, for every term t of type τ, for every $q_1 \in \mathfrak{Q}_1$ and $q_2 \in \mathfrak{Q}_2$ of type τ, such that t is recognized by $\mathfrak{A}_1$ in ρ at q_1 and by $\mathfrak{A}_2$ in ρ at q_2, then t is recognized by $\mathfrak{A}$ in ρ at $[q_1, q_2]$, by structural induction on t. There are four cases:

- $t \in [\![\mathfrak{B}_1(q_1)]\!]\rho_\tau$ and $t \in [\![\mathfrak{B}_2(q_2)]\!]\rho$: then $t \in [\![\mathfrak{B}([q_1, q_2])]\!]\rho_\tau$ by construction;
- $t \in [\![\mathfrak{B}_1(q_1)]\!]\rho_\tau$ and t is of the form $f(t_1, \ldots, t_n)$ with t_j recognized by $\mathfrak{A}_2$ in ρ at q_{2j}, $1 \leq j \leq n$, for some transition $f(q_{21}, \ldots, q_{2n}) \to q_2$ in $\mathfrak{R}_2$; then, since $t \in [\![\mathfrak{B}_1(q_1)]\!]\rho_\tau$ and using Lemma 1, $\mathfrak{B}_1(q_1) \not\equiv \mathbf{0}$, so there is a transition $f(\langle 1, q_{11}\rangle, \ldots, \langle 1, q_{1n}\rangle) \to [q_1, q_2]$ in $\mathfrak{R}$ by point 2 of the definition of $\mathfrak{R}$; by $(*)$ each t_j is recognized by $\mathfrak{A}$ in ρ at $\langle 1, q_{1j}\rangle$, so t is recognized by $\mathfrak{A}$ in ρ at $[q_1, q_2]$;
- t is of the form $f(t_1, \ldots, t_n)$ with t_j recognized by $\mathfrak{A}_1$ in ρ at q_{1j}, $1 \leq j \leq n$, for some transition $f(q_{11}, \ldots, q_{1n}) \to q_1$ in $\mathfrak{R}_1$, and $t \in [\![\mathfrak{B}_2(q_2)]\!]\rho$: symmetrical to the previous case;

- t is of the form $f(t_1, \dots, t_n)$ with t_j recognized by $\mathfrak{A}_i$ in ρ at q_{ij}, $1 \leq j \leq n$, for some transitions $f(q_{i1}, \dots, q_{in}) \rightarrow q_i$ in $\mathfrak{R}_i$, $i \in \{1, 2\}$: then t_j is recognized by $\mathfrak{A}$ in ρ at $[q_{1j}, q_{2j}]$ by induction hypothesis, and following the transition $f([q_{11}, q_{21}], \dots, [q_{1n}, q_{2n}]) \rightarrow [q_1, q_2]$ in $\mathfrak{R}$ (point 1 of its definition), t is recognized by $\mathfrak{A}$ in ρ at $[q_1, q_2]$. ☐

The following is a sufficient criterion for must-emptiness:

Lemma 4 (Must-Emptiness). *Let* $\mathfrak{A} \hat{=} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$ *be an* $\vee$*PTA. For each state* $q \in \mathfrak{Q}$*, let* ne_q *be a fresh Boolean variable. Let* $S_{\mathfrak{A}}$ *be the following set of Horn clauses:*

1. *for each* $q \in \mathfrak{Q}$ *such that* $\mathfrak{B}(q) \not\equiv \mathbf{0}$*, the unit clause* ne_q*;*
2. *for each* $f(q_1, \dots, q_n) \rightarrow q \in \mathfrak{R}$ *such that* $\mathfrak{B}(q) \equiv \mathbf{0}$*, the clause* $ne_{q_1} \wedge \dots \wedge ne_{q_n} \Rightarrow ne_q$*;*
3. *the unit clauses* $\neg ne_q$*, for each* $q \in \mathfrak{F}$*.*

If $S_{\mathfrak{A}}$ *is satisfiable (in the standard, Tarskian sense) then no term is recognized by* $\mathfrak{A}$ *in any environment* ρ*—in the latter case, we say that* $\mathfrak{A}$ *is* definitely empty.

Proof. Let's show by structural induction on t that, if t is recognized by $\mathfrak{A}$ in ρ at q, then ne_q must be true in any standard environment satisfying $S_{\mathfrak{A}}$. If $\mathfrak{B}(q) \not\equiv \mathbf{0}$, then $S_{\mathfrak{A}}$ contains the unit clause ne_q (item 1) and we are done. Otherwise, t cannot be in $[\![\mathfrak{B}(q)]\!]\rho_{\tau_q}$, so t is of the form $f(t_1, \dots, t_n)$, where for some $f(q_1, \dots, q_n) \rightarrow q \in \mathfrak{R}$ t_1 is recognized by $\mathfrak{A}$ in ρ at q_1, ..., t_n is recognized by $\mathfrak{A}$ in ρ at q_n. By induction hypothesis, ne_{q_1}, ..., ne_{q_n} must all be true in every standard environment satisfying $S_{\mathfrak{A}}$, so ne_q as well since $S_{\mathfrak{A}}$ contains the clause $ne_{q_1} \wedge \dots \wedge ne_{q_n} \Rightarrow ne_q$ by item 2.

Now, if t is recognized by $\mathfrak{A}$ in ρ at some $q \in \mathfrak{F}$, then ne_q must be true in any standard environment satisfying $S_{\mathfrak{A}}$. But this contradicts the clauses in item 3. ☐

This gives a fast sufficient criterion for must-emptiness, since satisfiability of Horn clauses can be done in linear time, provided that the test $\mathtt{B}(q) \equiv \mathbf{0}$ can be done in constant time—which is the case if we represent $\mathtt{B}(q)$ as BDDs. In fact, the Davis-Putnam procedure allows one to compute in linear time the *minimal model* of the set of clauses defined in items 1 and 2 of Lemma 4 [29]: then state q is definitely empty if and only if ne_q is false in this model. Such states can then be removed from the automaton, along with all transitions that refer to it. This pruning, or garbage-collection process is useful to limit the size of automata in concrete applications.

It is often the case that we wish to reason on automata not for every environment ρ, but only for environments ranging over a class defined by a series of *assumptions*:

Definition 3 (Assumptions). *An* assumption $\mathfrak{H}$ *is a map from types* τ *to formulae of type* τ*. We say that the environment* $\rho \hat{=} (\rho_\tau)_{\tau \in \mathcal{T}}$ satisfies $\mathfrak{H}$*, written* $\rho \models \mathfrak{H}$*, if and only if* $[\![\mathfrak{H}(\tau)]\!]\rho_\tau$ *is the set of all terms of type* τ*, for every type* τ*.*

Whenever F *and* G *are two formulae of type* τ*, the assumption* $F \cap G = \emptyset$ *denotes the function mapping* τ *to* $\neg(F \wedge G)$ *and every other type to* $\mathbf{1}$*. Given a finite family of assumptions* $\mathfrak{H}_i$*,* $i \in I$*, the assumption* $\bigwedge_{i \in I} \mathfrak{H}_i$ *maps every type* τ *to* $\bigwedge_{i \in I} \mathfrak{H}_i(\tau)$*. We write* $\mathfrak{H}_1 \wedge \mathfrak{H}_2$ *for* $\bigwedge_{i \in \{1,2\}} \mathfrak{H}_i$*.*

The following is easy and explains the notations above:

Lemma 5. *$\rho \models F \cap G = \emptyset$ if and only if $\llbracket F \rrbracket \rho_\tau \cap \llbracket G \rrbracket \rho_\tau = \emptyset$.*
$\rho \models \bigwedge_{i \in I} \mathfrak{H}_i$ if and only if $\rho \models \mathfrak{H}_i$ for every $i \in I$.

Given an assumption $\mathfrak{H}$, we may simplify any $\vee$PTA by simplifying each $\mathfrak{B}(q)$, knowing that it will only be used with environments ρ satisfying $\mathfrak{H}$:

Lemma 6 (Reduction). *Given an assumption $\mathfrak{H}$ and an $\vee$PTA $\mathfrak{A} \hat{=} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$, define $\mathfrak{A}_{|\mathfrak{H}}$ as $\mathfrak{A} \hat{=} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B}')$, where $\mathfrak{B}'$ maps each state q to $\mathfrak{B}(q) \wedge \mathfrak{H}(\tau_q)$.*

Then for every term t, for every environment ρ satisfying $\mathfrak{H}$, t is recognized by $\mathfrak{A}$ in ρ if and only if t is recognized by $\mathfrak{A}_{|\mathfrak{H}}$ in ρ. Moreover, if $\mathfrak{A}_{|\mathfrak{H}}$ is definitely empty, then for no environment ρ satisfying $\mathfrak{H}$, $\mathfrak{A}$ recognizes any term—in the latter case, we say that $\mathfrak{A}$ is definitely empty modulo $\mathfrak{H}$.

Proof. The first part is an easy induction on t: since ρ satisfies $\mathfrak{H}$, $\llbracket \mathfrak{B}(q) \wedge \mathfrak{H}(\tau_q) \rrbracket \rho_{\tau_q} = \llbracket \mathfrak{B}(q) \rrbracket \rho_{\tau_q}$ for each state q.

For the second part, if $\mathfrak{A}_{|\mathfrak{H}}$ is definitely empty, then by the first part $\mathfrak{A}$ cannot recognize any term in any environment satisfying $\mathfrak{H}$. ☐

4 Messages, Intruders and What Intruders Know

4.1 The Algebra of Messages

We consider a set $\mathcal{T}$ of types containing the type `msg` of *messages*; the type `msglist` of tuples of messages, which we shall use to build argument lists to the `t` tupling operator below; the type `K` of *raw keys*, e.g. integers of some fixed sizes used to build actual keys, of type `key`; the type `D` of *raw data*, e.g. integers, reals, strings, etc.

We use a series of function symbols to build actual keys from raw keys, i.e., of signature `K` $\to$ `key` : `symk` builds symmetric keys, `asymk1` and `asymk2` build the two parts of asymmetric keys (usually called the *public key* and the *private key*). These functions inject raw keys inside the space of actual keys, in such a way that no symmetric key can be confused with any asymmetric key, and that no public key can be confused with any private key. By convention, the *inverse key* of $\texttt{symk}(k)$ is $\texttt{symk}(k)$ itself, that of $\texttt{asymk1}(k)$ is $\texttt{asymk2}(k)$ and that of $\texttt{asymk2}(k)$ is $\texttt{asymk1}(k)$. If M is encrypted with K to yield the ciphertext $\{M\}_K$, then the inverse key of K can be used to recover M from $\{M\}_K$. This decrypting process is not represented in the term algebra we consider, as in [24] among others; rather decryption will be part of the concrete semantics of our language, as in [4], [27] or [1] (see Figure 2).

For practical reasons, we shall use a few other constructors of actual keys. The first is `sk` : `msg` $\times$ `msg` $\to$ `key`; the term $\texttt{sk}(id_A, id_B)$, where id_A and id_B are messages denoting the respective identities of some principals A and B, is intended to represent a long-term symmetric key shared between A and B. Similarly, `pubk` : `msg` $\to$ `key` and `privk` : `msg` $\to$ `key` are such that $\texttt{pubk}(id_A)$ returns A's public key and $\texttt{privk}(id_A)$ returns A's private key. This is needed in public-key-based protocols, where B needs to get hold of A's public key to initiate secret communications to A. We model this by having B use $\texttt{pubk}(id_A)$. This ignores the difficulty of ensuring that B actually gets A's public key and no other. However, we may assume that getting authentic public keys is the task of some other cryptographic protocol, which will be analyzed separately.

Since the terms $\mathtt{sk}(id_A, id_B)$ and $\mathtt{symk}(k)$ are always distinct, the informal semantics of $\mathtt{symk}(k)$ has to be modified: such keys are symmetric keys that are distinct from any long-term symmetric key, and similarly $\mathtt{asymk1}(k)$ and $\mathtt{asymk2}(k)$ are asymmetric keys that are distinct from any public and private keys owned by principals.

Finally, it would be possible to split these kinds of keys further, and have several function symbols $\mathtt{symk}_n$, where n is the number of bits of the key, instead of just one `symk`, or even symbols $\mathtt{symk}_{a,n}$ where a indicates which cryptographic algorithm the key is to be used with, and similarly for `asymk1`, `asymk2`, `sk`, `pubk`, `privk`. This is certainly desirable in actual verification tools, but would complicate the presentation; this is why we won't consider this extension here.

Any actual key is a message, and this is represented by an explicit conversion symbol $\mathtt{k} : \mathtt{key} \to \mathtt{msg}$. Similarly, any basic datum is a message, as represented by the explicit conversion $\mathtt{d} : \mathtt{D} \to \mathtt{msg}$. Note that, qua message, no key can be confused with any basic data: this is accomplished in practice by using explicitly typed messages, as already advocated in Section 2. Additionally, the function symbol $\mathtt{c} : \mathtt{msg} \times \mathtt{key} \to \mathtt{msg}$ is used to build ciphertexts: $\mathtt{c}(M, K)$ is what we used to write $\{M\}_K$ until now. The fact that the second argument to `c` has to be a key, and not just any message, is less natural than it might seem: in practice nothing really distinguishes (raw) keys from general (raw) messages, as they are just sequences of bits. The typing discipline above forces implementors to actually check their keys before they use them to encrypt messages. It turns out that this also helps in verifying protocols, as we shall see in Section 4.2.

Finally, any list of messages can be made into a message, using the tupling constructor $\mathtt{t} : \mathtt{msglist} \to \mathtt{msg}$. Lists of messages, of type `msglist`, are build using the standard Lisp constructors $\mathtt{nil} :\to \mathtt{msglist}$ and $\mathtt{cons} : \mathtt{msg} \times \mathtt{msglist} \to \mathtt{msglist}$. For example, the triple (M_1, M_2, M_3) will be coded by the term $\mathtt{t}(\mathtt{cons}(M_1, \mathtt{cons}(M_2, \mathtt{cons}(M_3, \mathtt{nil}))))$. For legibility we shall abbreviate $\mathtt{cons}(M_1, \ldots, \mathtt{cons}(M_n, \mathtt{nil}) \ldots)$ as $[M_1, \ldots, M_n]$.

Note that not only there is no ambiguity as to whether a given message is a tuple, a ciphertext, a key or another basic datum, but there is no ambiguity either as to the number of components of tuples. That is, we assume that it is impossible to confuse any triple with any couple for example. This eliminates quite a number of possible attacks (see [24] for an example) but reflects good engineering practice. Using ASN.1 sequence types to represent tuples entices implementors to check not only that the message they expect is a tuple, but also that the number of components is as they expect, too.

To sum up, we consider the *basic signature* Σ_0 described in Figure 1. We assume that the actual signature Σ we use is of the form $\Sigma_0 \uplus \Sigma_1$, where Σ_1 is an unspecified collection of function symbols of signatures $\tau_1 \times \ldots \times \tau_n \to \tau$ where $\tau \in \{\mathtt{D}, \mathtt{K}\}$. The set $\mathcal{T}$ of types is any set containing $\{\mathtt{K}, \mathtt{D}, \mathtt{key}, \mathtt{msg}, \mathtt{msglist}\}$. Leaving $\mathcal{T}$ and Σ partly unspecified allows us to deal with extensible types for raw keys and raw data. However, we restrict messages (of type `msg`) to be either keys, data, tuples or ciphertexts only, and lists of messages to be either `nil` or lists starting by `cons`. We have not introduced any function symbol to deal with cryptographic hash functions: the hash of M is just $\mathtt{c}(M, *)$, where $*$ is a key that has no inverse. We define inverse keys as follows:

Definition 4 (Inverse). *For any keys K and K' (of type* `key`*), we say that K' is an* inverse *of K if and only if $K = \mathtt{symk}(k)$ or $K = \mathtt{sk}(M_1, M_2)$ and $K' = K$; or*

$$
\begin{array}{rl@{\qquad}rl}
\mathtt{symk} : & \mathtt{K} \to \mathtt{key} & \mathtt{k} : & \mathtt{key} \to \mathtt{msg} \\
\mathtt{asymk1} : & \mathtt{K} \to \mathtt{key} & \mathtt{d} : & \mathtt{D} \to \mathtt{msg} \\
\mathtt{asymk2} : & \mathtt{K} \to \mathtt{key} & \mathtt{c} : & \mathtt{msg} \times \mathtt{key} \to \mathtt{msg} \\
\mathtt{sk} : & \mathtt{msg} \times \mathtt{msg} \to \mathtt{key} & \mathtt{t} : & \mathtt{msglist} \to \mathtt{msg} \\
\mathtt{pubk} : & \mathtt{msg} \to \mathtt{key} & \mathtt{nil} : & \to \mathtt{msglist} \\
\mathtt{privk} : & \mathtt{msg} \to \mathtt{key} & \mathtt{cons} : & \mathtt{msg} \times \mathtt{msglist} \to \mathtt{msglist} \\
* : & \to \mathtt{key} & &
\end{array}
$$

Fig. 1. The basic signature

$K = \mathtt{asymk1}(k)$ and $K' = \mathtt{asymk2}(k)$; or $K = \mathtt{asymk2}(k)$ and $K' = \mathtt{asymk1}(k)$; or $K = \mathtt{pubk}(M)$ and $K' = \mathtt{privk}(M)$; or $K = \mathtt{privk}(M)$ and $K' = \mathtt{pubk}(M)$.

4.2 Deductive Capabilities of Intruders

Intruders can read on any communication line, and collect what they read. Let E be a set of messages that the intruders have collected (this set might be infinite). These intruders can then forge new messages from E and send them to other principals. Following [4], we model intruders as a deductive system. Write $E \mapsto M$ the predicate "from the set E of messages, the intruders may deduce the message M", defined by the rules in Figure 2. E, M denotes the union of E with $\{M\}$.

$$
\frac{}{E, M \mapsto M}\,(Ax)
$$

$$
\frac{E \mapsto M \quad E \mapsto \mathtt{k}(K)}{E \mapsto \mathtt{c}(M, K)}\,(CryptI)
\qquad
\frac{E \mapsto \mathtt{c}(M, K) \quad E \mapsto \mathtt{k}(K') \quad (K' \text{ inverse of } K)}{E \mapsto M}\,(CryptE)
$$

$$
\frac{E \mapsto M_1 \quad \dots \quad E \mapsto M_n}{E \mapsto \mathtt{t}([M_1, \dots, M_n])}\,(TupleI)
\qquad
\frac{E \mapsto \mathtt{t}([M_1, \dots, M_n])}{E \mapsto M_i}\,(TupleE_i), 1 \leq i \leq n
$$

Fig. 2. What intruders may deduce, in natural deduction format

Rule (Ax) says that intruders may always replay messages without altering them. D. Bolignano [4] uses this to observe that all communications, even untapped ones, can be simulated by sending messages to the intruders, having the intruders deduce whatever messages, and send them to other arbitrary principals.

The rules ending in E are *elimination rules*, and allow the intruders to decrypt and read components off tuples. The rules ending in I are *introduction rules*, and allow the intruders to encrypt and build tuples. There is no rule allowing the intruders to *crack* ciphertexts, i.e. to recover M or K from $\mathtt{c}(M, K)$ without them deducing K first.

A *derivation* is any finite tree of judgments of the form $E \mapsto M$ connected by instances of the rules of Figure 2, and whose leaves (on top) are instances of (Ax) or of $(TupleI)$ with $n = 0$. Its root (at the bottom) is the *conclusion* of the derivation. We say that $E \mapsto M$ is *derivable* if and only if it is the conclusion of some derivation.

Definition 5. *Given a set E of messages, the set $Ded(E)$ of messages* deducible *from E is the set of messages M such that $E \mapsto M$ is derivable in the system of Figure 2.*

M is extractible *from E if $E \mapsto M$ is derivable, using only (Ax) and the elimination rules. Let $Ext(E)$ be the set of messages extractible from E.*

M is constructible *from E if $E \mapsto M$ is derivable, using only (Ax) and the introduction rules. Let $Con(E)$ be the set of messages constructible from E.*

The following theorem states that we can always assume that intruders do all extractions (e.g., decryptions) before all constructions (e.g., encryptions):

Theorem 1 ([4]). $Ded(E) = Con(Ext(E))$

Proof. This can be done as in [4], although the deduction rules are different. A simpler proof is based on the proof-theoretic tool of *proof normalization* [28]. Define a rewrite relation on derivations by the following rules:

$$\dfrac{\dfrac{\begin{matrix}\vdots\,\pi_1\\ E\mapsto M\end{matrix}\quad \begin{matrix}\vdots\,\pi_2\\ E \mapsto \mathtt{k}(K)\end{matrix}}{E\mapsto \mathtt{c}(M,K)}\,(CryptI)\qquad \begin{matrix}\vdots\,\pi_3\\ E\mapsto \mathtt{k}(K')\end{matrix}}{E\mapsto M}\,(CryptE) \quad\longrightarrow\quad \begin{matrix}\vdots\,\pi_1\\ E\mapsto M\end{matrix}$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\,\pi_1\\ E\mapsto M_1\end{matrix}\ \ \ldots\ \ \begin{matrix}\vdots\,\pi_n\\ E\mapsto M_n\end{matrix}}{E\mapsto \mathtt{t}([M_1,\ldots,M_n])}\,(TupleI)}{E\mapsto M_i}\,(TupleE_i) \quad\longrightarrow\quad \begin{matrix}\vdots\,\pi_i\\ E\mapsto M_i\end{matrix}$$

where K' is an inverse of K. These rules terminate, as they make the derivation decrease in size. So every derivable judgment has a *normal* derivation, one to which none of the rewrite rules above applies.

Now normal derivations π satisfy: $(*)$ if the last rule of π is an elimination, then all rules in π are eliminations or (Ax). This is a straightforward structural induction on π.

Let M be deducible from E. We show that $M \in Con(Ext(E))$ by structural induction on a normal derivation π of $E \mapsto M$. If the last rule of π is (Ax), then $M \in E$, so this is clear. If the last rule is an introduction, then the result is by the induction hypothesis. If the last rule is an elimination, then by $(*)$ $M \in Ext(E)$, whence the result. So $Ded(E) \subseteq Con(Ext(E))$. The converse inclusion is obvious. ☐

As we have already said, this does not work any longer if arbitrary messages can be used as keys. Assume, informally, that $\{M\}_K$ can be used as a symmetric key.

Then, using any reasonable tentative extension of the system of Figure 2, M_1 must be deducible from $E \hat{=} \{M, K, \{M_1\}_{\{M\}_K}\}$: encrypt M to get $\{M\}_K$, then use the latter to decrypt $\{M_1\}_{\{M\}_K}$ and get M_1. But encryption has to precede decryption here.

4.3 Completion of Automata

We shall represent sets of messages E by $\vee$PTAs. One central problem we shall face in the sequel is how to compute $Ded(E)$, or a superset of it, as the set of messages recognized by some $\vee$PTA, computable from E. This is done in two steps, using Theorem 1.

We shall however use only special automata:

Definition 6. *An $\vee$PTA $(\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$ is* normal *if and only if, for every $q \in \mathfrak{Q}$ of type* `msg`, `msglist` *or* `key`, *$\mathfrak{B}(q) \equiv \mathbf{0}$; and every transition $f(q_1, \ldots, q_n) \to q$ is such that f is in the basic signature Σ_0.*

Note that, by the way, intersection of normal automata is computed exactly. Whereas this is not crucial, we shall still need to restrict to normal automata in the following lemma, which states a way of computing a normal $\vee$PTA recognizing at least all messages extractible from a set E of messages recognized by a given normal $\vee$PTA $\mathfrak{A}$.

Lemma 7. *Let $\mathfrak{A} \hat{=} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$ be a normal $\vee$PTA of type* `msg`*, and E be the set of messages recognized by $\mathfrak{A}$.*

For any state $q \in \mathfrak{Q}$, let $\mathfrak{A}_q$ denote the automaton $(\mathfrak{Q}, \{q\}, \mathfrak{R}, \mathfrak{B})$. Given two states q_1 and q_2 in $\mathfrak{Q}$ of the same type, we say that q_1 and q_2 possibly intersect *if and only if $\mathfrak{A}_{q_1} \sqcap \mathfrak{A}_{q_2}$ is not definitely empty.*

Given two states q_1 and q_2 in $\mathfrak{Q}$ of type `key`*, we say that q_2* contains possible inverses of *q_1 if and only if there are transitions $f_1(q_{11}, \ldots, q_{1n}) \to q_1$ and $f_2(q_{21}, \ldots, q_{2n}) \to q_2$ such that q_{1j} and q_{2j} intersect possibly for every $1 \leq j \leq n$, where f_1, f_2 and n are given in the following table:*

f_1	`symk`	`asymk1`	`asymk2`	`sk`	`pubk`	`privk`
f_2	`symk`	`asymk2`	`asymk1`	`sk`	`privk`	`pubk`
n	1	1	1	2	1	1

Given a state q in $\mathfrak{Q}$ of type `msglist`*, let $Elems(q)$ be the smallest subset of $\mathfrak{Q}$ containing q and such that, for any transition $\texttt{cons}(q_1, q_2) \to q_3$ with $q_3 \in Elems(q)$, $q_1 \in Elems(q)$ and $Elems(q_2) \subseteq Elems(q)$.*

Let $\mathfrak{F}_{ext}$ be the smallest subset of $\mathfrak{Q}$ containing $\mathfrak{F}$ such that:

1. *for any transition $\texttt{c}(q_1, q_2) \to q_3$ with $q_3 \in \mathfrak{F}_{ext}$, for any transition $\texttt{k}(q_4) \to q_5$ with $q_5 \in \mathfrak{F}_{ext}$, if q_4 contains possible inverses of q_2, then $q_1 \in \mathfrak{F}_{ext}$;*
2. *for any transition $\texttt{t}(q_1) \to q$ with $q \in \mathfrak{F}_{ext}$, $Elems(q_1) \subseteq \mathfrak{F}_{ext}$.*

Then:

- (i) *if there is a term recognized by $\mathfrak{A}$ in ρ at states q_1 and q_2, then q_1 and q_2 intersect possibly;*
- (ii) *if there is a key K recognized by $\mathfrak{A}$ in ρ at q_1, a key K' recognized by $\mathfrak{A}$ in ρ at q_2 and K' is an inverse of K, then q_2 contains possible inverses of q_1;*

(iii) *if there is a list* $[M_1, \ldots, M_n]$ *of messages recognized by* $\mathfrak{A}$ *in* ρ *at* q*, then each* M_j*,* $1 \leq j \leq n$*, is recognized by* $\mathfrak{A}$ *in* ρ *at some state of* $Elems(q)$*;*

(iv) *if* M *is extractible from* E*, then* M *is recognized by* $\mathfrak{A}$ *in* ρ *at some state of* $\mathfrak{F}_{ext}$*.*

By abuse of notation, let $Ext(\mathfrak{A}) \hat{=} (\mathfrak{Q}, \mathfrak{F}_{ext}, \mathfrak{R}, \mathfrak{B})$*: so* $[\![Ext(\mathfrak{A})]\!]\rho \supseteq Ext([\![\mathfrak{A}]\!]\rho)$*.*

Proof. (i) is by Lemma 3. For (ii), notice that by normality $\mathtt{B}(q_1) \equiv \mathtt{B}(q_2) \equiv \mathbf{0}$, so if K (resp. K') is recognized by $\mathfrak{A}$ in ρ at q_1 (resp. q_2), then K is of the form $f_1(t_{11}, \ldots, t_{1n})$ (resp. K' is of the form $f_2(t_{21}, \ldots, t_{2n})$) and there are transitions $f_i(q_{i1}, \ldots, q_{in}) \to q_i, i \in \{1, 2\}$, such that t_{ij} is recognized by $\mathfrak{A}$ in ρ at q_{ij}, $i \in \{1, 2\}$, $1 \leq j \leq n$. Since K' is an inverse of K, Definition 4 shows that f_1, f_2 and n are as in the table above, and moreover $t_{1j} = t_{2j}$ for every j, $1 \leq j \leq n$. By (i), q_{1j} and q_{2j} intersect possibly for each j, $1 \leq j \leq n$. So q_2 contains possible inverses of q_1.

(iii) is by induction on $j \geq 1$: if $j = 1$, the fact that $[M_1, \ldots, M_n]$ is recognized by $\mathfrak{A}$ in ρ at q entails that there is a transition $\mathtt{cons}(q_1, q_2) \to q$ with M_1 recognized by $\mathfrak{A}$ in ρ at q_1 (since by normality $\mathfrak{B}(q) \equiv \mathbf{0}$). Since $q \in Elems(q)$, q_1 is in $Elems(q)$ as well. If $j \geq 2$, then the fact that $[M_1, \ldots, M_n]$ is recognized by $\mathfrak{A}$ in ρ at q entails similarly that there is a transition $\mathtt{cons}(q_1, q_2) \to q$ with $[M_2, \ldots, M_n]$ recognized by $\mathfrak{A}$ in ρ at q_2: we conclude by appealing to the induction hypothesis.

As for (iv), let π be any derivation of $E \mapsto M$ that uses only (Ax) and elimination rules. We show that M is recognized at some state of $\mathfrak{F}_{ext}$ by structural induction on π. If the last rule of π is (Ax), then $M \in E$, hence by definition M is recognized at some state of $\mathfrak{F}$, and we conclude by the fact that $\mathfrak{F} \subseteq \mathfrak{F}_{ext}$. If the last rule of π is $(CryptE)$, then π derives $E \mapsto M$ from two smaller derivations of $E \mapsto \mathtt{c}(M, K)$ and $E \mapsto \mathtt{k}(K')$ with K' inverse of K. So by induction hypothesis $\mathtt{c}(M, K)$ is recognized at some $q_3 \in \mathfrak{F}_{ext}$, and $\mathtt{k}(K')$ is recognized at some $q_5 \in \mathfrak{F}_{ext}$. By normality $\mathfrak{B}(q_3) \equiv \mathfrak{B}(q_5) \equiv \mathbf{0}$, so there must be transitions $\mathtt{c}(q_1, q_2) \to q_3$ and $\mathtt{k}(q_4) \to q_5$ such that M is recognized at q_1, K is recognized at q_2 and K' is recognized at q_4. Since K' is an inverse of K, by (ii) q_4 contains possible inverses of q_2, so $q_1 \in \mathfrak{F}_{ext}$ by item 1, whence the claim. If the last rule of π is $(TupleE_j)$, $1 \leq j \leq n$, then π derives $E \mapsto M$ from a smaller derivation of $E \mapsto \mathtt{t}([M_1, \ldots, M_n])$, with $M = M_j$. By induction hypothesis $\mathtt{t}([M_1, \ldots, M_n])$ is recognized at some $q \in \mathfrak{F}_{ext}$. By normality $\mathfrak{B}(q) \equiv \mathbf{0}$, so there must be a transition $\mathtt{t}(q_1) \to q$ with $[M_1, \ldots, M_n]$ recognized at q_1. By (iii), M is recognized at some state of $Elems(q_1)$, so M is recognized at some state of $\mathfrak{F}_{ext}$ by item 2. ☐

$Ext(\mathfrak{A})$ is easily computable: compute $\mathfrak{F}_{ext}$ as a least fixpoint, i.e. by completing $\mathfrak{F}$ using the rules 1 and 2 until no more state can be added. Note also that, since in normal automata intersection is computed exactly, the set of messages recognized by $Ext(\mathfrak{A})$ in ρ is exactly $Ext(E)$. We leave the verification of this claim to the reader.

Lemma 8. *Let* $\mathfrak{A} \hat{=} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$ *be a normal* $\vee$*PTA of type* `msg`*, and* E *be the set of messages recognized by* $\mathfrak{A}$ *in* ρ*.*

Let qm *and* ql *be two distinct fresh states of respective types* `msg` *and* `msglist`*. Let* $Con(\mathfrak{A})$ *be the normal* $\vee$*PTA* $(\mathfrak{Q}', \mathfrak{F}', \mathfrak{R}', \mathfrak{B}')$*, where:*

- $\mathfrak{Q}' \hat{=} \mathfrak{Q} \cup \{qm, ql\}$*;*
- $\mathfrak{F}' \hat{=} \{qm\}$*;*

- $$\begin{aligned} \mathfrak{R}' \hat{=} \mathfrak{R} &\cup \{\mathtt{nil}() \to ql, \mathtt{cons}(qm, ql) \to ql, \mathtt{t}(ql) \to qm\} && (a)\\ &\cup \{\mathtt{c}(qm, q) \to qm \mid \mathtt{k}(q) \to q' \in \mathfrak{R}, q' \in \mathfrak{F}\} && (b)\\ &\cup \{\mathtt{c}(q_1, q_2) \to qm \mid \mathtt{c}(q_1, q_2) \to q' \in \mathfrak{R}, q' \in \mathfrak{F}\} && (c)\\ &\cup \{\mathtt{t}(q) \to qm \mid \mathtt{t}(q) \to q' \in \mathfrak{R}, q' \in \mathfrak{F}\} && (d)\\ &\cup \{\mathtt{k}(q) \to qm \mid \mathtt{k}(q) \to q' \in \mathfrak{R}, q' \in \mathfrak{F}\} && (e)\\ &\cup \{\mathtt{d}(q) \to qm \mid \mathtt{d}(q) \to q' \in \mathfrak{R}, q' \in \mathfrak{F}\} && (f) \end{aligned}$$
- $\mathfrak{B}'$ *maps each* $q \in \mathfrak{Q}$ *to* $\mathfrak{B}(q)$*,* qm *and* ql *to* $\mathbf{0}$.

If M *is constructible from* E*, then* M *is recognized by* $Con(\mathfrak{A})$ *in* ρ*. In other words,* $[\![Con(\mathfrak{A})]\!]\rho \supseteq Con([\![\mathfrak{A}]\!]\rho)$*.*

Proof. First: $(*)$ t is recognized at $q \in \mathfrak{Q}$ in ρ by $Con(\mathfrak{A})$ if and only if t is recognized at q in ρ by $\mathfrak{A}$. This is an easy structural induction on t, noticing (in the only if direction) that all transitions in $\mathfrak{R}'$ of the form $f(q_1, \ldots, q_n) \to q'$ with $q' \in \mathfrak{Q}$ are already in $\mathfrak{R}$.

Then: $(**)$ if the message M is recognized in ρ by $\mathfrak{A}$, then M is recognized in ρ at qm by $Con(\mathfrak{A})$. This is by case analysis on M. If M is of the form $\mathtt{c}(M_1, K_1)$, then since $\mathfrak{A}$ is normal, there must be a transition $\mathtt{c}(q_1, q_2) \to q'$ in $\mathfrak{R}$, with $q' \in \mathfrak{F}$, such that M_1 is recognized at q_1 and K_1 at q_2 by $\mathfrak{A}$ in ρ. By $(*)$, M_1 is recognized at q_1 and K_1 at q_2 by $Con(\mathfrak{A})$ in ρ. By item (c) of the definition, there is a transition $\mathtt{c}(q_1, q_2) \to qm$ in $\mathfrak{R}'$, so M is recognized in ρ at qm by $Con(\mathfrak{A})$. Similarly if M starts with $\mathtt{t}$, $\mathtt{k}$ or $\mathtt{d}$, using items (d), (e) or (f) respectively.

Also: $(***)$ if $\mathtt{k}(K)$ is constructible from E, then $\mathtt{k}(K) \in E$. This is obvious: neither $(CryptI)$ nor $(TupleI)$ applies, so (Ax) must have been used.

We now prove the claim by structural induction on any given derivation π of $E \mapsto M$ that uses only (Ax) and introduction rules. If π ends in (Ax), then $M \in E$, so M is recognized by $\mathfrak{A}$ in ρ. By $(**)$, M is recognized by $Con(\mathfrak{A})$ in ρ.

If π ends in $(CryptI)$, then M is of the form $\mathtt{c}(M_1, K_1)$, where M_1 and $\mathtt{k}(K_1)$ are constructible from E. So, first, M_1 is recognized by $Con(\mathfrak{A})$ in ρ at qm (1), by induction hypothesis; second, $\mathtt{k}(K_1)$ is in E by $(***)$, and is therefore recognized by $\mathfrak{A}$: by normality, there is a transition $\mathtt{k}(q) \to q'$ in $\mathfrak{R}$ with $q' \in \mathfrak{F}$ such that K_1 is recognized by $\mathfrak{A}$ in ρ at q. By $(*)$, K_1 is recognized by $Con(\mathfrak{A})$ in ρ at q. Using (1) and item (b) of the definition, $\mathtt{c}(M_1, K_1)$ is recognized by $Con(\mathfrak{A})$ in ρ at qm.

If π ends in $(TupleI)$, then M is of the form $\mathtt{t}([M_1, \ldots, M_n])$, where $M_1, \ldots,$ M_n are constructible from E. By induction hypothesis, $M_1, \ldots, M_n$ are recognized by $Con(\mathfrak{A})$ in ρ at qm. An easy induction on j, using the transitions of item (a), then shows that each sublist $[M_j, \ldots, M_n]$ is recognized by $Con(\mathfrak{A})$ in ρ at ql, hence M is recognized by $Con(\mathfrak{A})$ in ρ at qm. ☐

We leave the reader the task to check that this construction actually represents $Con(E)$ exactly. It follows that the following inclusion can in fact be strengthened to an equality:

Theorem 2. *For every normal* $\vee$*PTA* $\mathfrak{A}$*, let* $Ded(\mathfrak{A}) \hat{=} Con(Ext(\mathfrak{A}))$*. Then* $[\![Ded(\mathfrak{A})]\!]\rho \supseteq Ded([\![\mathfrak{A}]\!]\rho)$*.*

Proof. By Theorem 1, Lemma 7, Lemma 8, and the fact that Con is $\subseteq$-monotonic. ☐

4.4 Initial Intruder Knowledge

Finally, we shall need to know an automaton characterizing what the intruders know at most at the beginning of a protocol. To simplify things, assume that long-term keys $\mathtt{sk}(id_A, id_B)$, and long-term private asymmetric keys $\mathtt{privk}(id_A)$ are always initially unknown to the intruders, but that all public keys $\mathtt{pubk}(id_A)$ are known to the intruders.

We shall also have to model freshness of messages. Say that a term of type `K` or `D` *exists* if and only if no principal ever created it yet. Fresh raw keys and fresh data will be terms of type `K` or `D` respectively (in the unspecified signature Σ) that did not exist yet. In the following definition, $\mathcal{K}_0$ is the set of all existing raw keys at the beginning of the protocol, and $\mathcal{D}_0$ the set of all existing raw data (terms of type `D`) at the beginning of the protocol. $\mathcal{SSK}_0, \mathcal{SAK}1_0, \mathcal{SAK}2_0$ are sets of raw keys k such that $\mathtt{symk}(k)$, $\mathtt{asymk1}(k)$ and $\mathtt{asymk2}(k)$ are secret respectively—secret meaning impossible to deduce by the intruders. $\mathcal{SD}_0$ is the set of raw data d such that $\mathtt{d}(d)$ is secret. Expressing that they have to actually be secret will be done in Lemma 9.

Definition 7. *Given sets $\mathcal{K}_0$ of terms of type* `K` *and $\mathcal{D}_0$ of terms of type* `D`*, the set of $\mathcal{K}_0$,* $\mathcal{D}_0$-existing terms of type τ *is the smallest such that:*

- *every $t \in \mathcal{K}_0$ exists of type* `K`*;*
- *every $t \in \mathcal{D}_0$ exists of type* `D`*;*
- *for every $f : \tau_1 \times \ldots \times \tau_n \to \tau$ in Σ_0, for every existing terms t_1 of type τ_1, ..., t_n of type τ_n, $f(t_1, \ldots, t_n)$ exists of type τ.*

Given sets $\mathcal{SSK}_0$, $\mathcal{SAK}1_0$, $\mathcal{SAK}2_0$ of terms of type `K` *and a set $\mathcal{SD}_0$ of terms of type* `D`*, let the* set of known terms $Known(\mathcal{K}_0, \mathcal{D}_0, \mathcal{SSK}_0, \mathcal{SAK}1_0, \mathcal{SAK}2_0, \mathcal{SD}_0)$ *be the largest set E of $\mathcal{K}_0$, $\mathcal{D}_0$-existing terms of type* `msg` *such that: (†) no term of the form $\mathtt{k}(\mathtt{sk}(M_1, M_2))$, or of the form $\mathtt{k}(\mathtt{privk}(M))$, or of the form $\mathtt{k}(\mathtt{symk}(K))$ ($K \in \mathcal{SSK}_0$), or $\mathtt{k}(\mathtt{asymk1}(K))$ ($K \in \mathcal{SAK}1_0$), or $\mathtt{k}(\mathtt{asymk2}(K))$ ($K \in \mathcal{SAK}2_0$), or $\mathtt{d}(D)$ ($D \in \mathcal{SD}_0$) is deducible from E.*

It is not obvious that $Known(\mathcal{K}_0, \mathcal{D}_0, \mathcal{SSK}_0, \mathcal{SAK}1_0, \mathcal{SAK}2_0, \mathcal{SD}_0)$ makes sense. The following shows that it does, and gives a representation of it as a $\vee$PTA:

Lemma 9. *Let XK_0, SSK_0, $SAK1_0$, $SAK2_0$ be formulae of type* `K` *and XD_0 and SD_0 be formulae of type* `D`*. Build the following normal $\vee$PTAs $\mathfrak{Exist}_\tau(XK_0, XD_0)$—for every type τ—and $\mathfrak{Known}(XK_0, XD_0, SSK_0, SAK1_0, SAK2_0, SD_0)$.*

$\mathfrak{Exist}_\tau(XK_0, XD_0)$ is the $\vee$PTA $(\mathfrak{Qx}, \mathfrak{Fx}, \mathfrak{Rx}, \mathfrak{Bx})$ (we drop the dependencies on XK_0 and XD_0 for legibility), where:

- *$\mathfrak{Qx}$ is a set of distinct states $q_{\tau'}$, one for each type τ';*
- *$\mathfrak{Fx} \hat{=} \{q_\tau\}$;*
- *$\mathfrak{Rx} \hat{=} \{f(q_{\tau_1}, \ldots, q_{\tau_n}) \to q_{\tau'} \mid f : \tau_1 \times \ldots \times \tau_n \to \tau' \in \Sigma_0\}$;*
- *$\mathfrak{Bx}$ maps* `K` *to XK_0,* `D` *to XD_0, and every other type to $\mathbf{0}$.*

Then whenever $[\![XK_0]\!]\rho = \mathcal{K}_0$ and $[\![XD_0]\!]\rho = \mathcal{D}_0$, $[\![\mathfrak{Exist}_\tau(XK_0, XD_0)]\!]\rho$ is the set of $\mathcal{K}_0, \mathcal{D}_0$-existing terms of type τ.

$\mathfrak{Known}(XK_0, XD_0, SSK_0, SAK1_0, SAK2_0, SD_0)$ is the following $\vee$PTA $(\mathfrak{Qk}, \mathfrak{Fk}, \mathfrak{Rk}, \mathfrak{Bk})$

$$\begin{array}{rcll}
\checkmark\, qm & ::= & \mathtt{c}(qm, q_{\mathtt{key}}) \mid \mathtt{c}(q_{\mathtt{msg}}, qk^{-1}) \mid \mathtt{t}(ql) \mid \mathtt{k}(qk) \mid \mathtt{d}(qd) & : \mathtt{msg} \\
ql & ::= & \mathtt{nil}() \mid \mathtt{cons}(qm, ql) & : \mathtt{msglist} \\
qk & ::= & \mathtt{symk}(qsk) \mid \mathtt{asymk1}(qak_1) \mid \mathtt{asymk2}(qak_2) \mid \mathtt{pubk}(q_{\mathtt{msg}}) \mid *() & : \mathtt{key} \\
qsk & ::= & XK_0 \wedge \neg SSK_0 & : \mathtt{K} \\
qak_1 & ::= & XK_0 \wedge \neg SAK1_0 & : \mathtt{K} \\
qak_2 & ::= & XK_0 \wedge \neg SAK2_0 & : \mathtt{K} \\
qk^{-1} & ::= & \mathtt{symk}(qsk^{-1}) \mid \mathtt{asymk1}(qak_1^{-1}) \mid \mathtt{asymk2}(qak_2^{-1}) & \\
 & & \mid \mathtt{sk}(q_{\mathtt{msg}}, q_{\mathtt{msg}}) \mid \mathtt{pubk}(q_{\mathtt{msg}}) \mid *() & : \mathtt{key} \\
qsk^{-1} & ::= & XK_0 \wedge SSK_0 & : \mathtt{K} \\
qak_1^{-1} & ::= & XK_0 \wedge SAK2_0 & : \mathtt{K} \\
qak_2^{-1} & ::= & XK_0 \wedge SAK1_0 & : \mathtt{K} \\
qd & ::= & XD_0 \wedge \neg SD_0 & : \mathtt{D}
\end{array}$$

plus all transitions and states of $\mathfrak{Exist}_{\mathtt{msg}}(XK_0, XD_0)$.

Then whenever $[\![XK_0]\!]\rho = \mathcal{K}_0$, $[\![XD_0]\!]\rho = \mathcal{D}_0$, $[\![SSK_0]\!]\rho = \mathcal{SSK}_0$, $[\![SAK1_0]\!]\rho = \mathcal{SAK}1_0$, $[\![SAK2_0]\!]\rho = \mathcal{SAK}2_0$, and $[\![SD_0]\!]\rho = \mathcal{SD}_0$, then $[\![\mathfrak{Known}(XK_0, XD_0, SSK_0, SAK1_0, SAK2_0, SD_0)]\!]\rho = Known(\mathcal{K}_0, \mathcal{D}_0, \mathcal{SSK}_0, \mathcal{SAK}1_0, \mathcal{SAK}2_0, \mathcal{SD}_0)$.

The difficult part in $\mathfrak{Known}(XK_0, XD_0, SSK_0, SAK1_0, SAK2_0, SD_0)$ is the case of encryptions. The idea is that the messages of the form $\mathtt{c}(M, K)$ that intruders may know have two possible forms: either intruders are able to build $\mathtt{c}(M, K)$ by $(CryptI)$, so in particular M is known, hence recognized at *qm*; or they cannot deduce M, then they cannot use $(CryptE)$ to deduce M from $\mathtt{c}(M, K)$, so K must be recognized at some state (qk^{-1}) recognizing exactly the keys that do not have any deducible inverse. The state *qk* recognizes all keys that *are* deducible.

Proof. By an easy structural induction on t, t is recognized by $\mathfrak{Exist}_\tau(XK_0, XD_0)$ in ρ at $q_{\tau'}$ if and only if t exists of type τ'. So $[\![\mathfrak{Exist}_\tau(XK_0, XD_0)]\!]\rho$ is the set of $\mathcal{K}_0, \mathcal{D}_0$-existing terms of type τ.

Let $\mathfrak{A}$ the $\vee$PTA $\mathfrak{Known}(XK_0, XD_0, SSK_0, SAK1_0, SAK2_0, SD_0)$ and call a *good set* any set of messages obeying the conditions (†) of Definition 7. The terms t recognized by $\mathfrak{A}$ at q_τ are the existing terms of type τ, by an easy induction on t.

Observe that: (1) *qk* does not contain any possible inverses of qk^{-1} (see Lemma 7).

Observe also that: (2) each term t recognized by $\mathfrak{A}$ in ρ at any state of any type τ is an existing term of type τ. This is a straightforward structural induction on t.

We then show that: (3) $Ded([\![\mathfrak{A}]\!]\rho) = [\![\mathfrak{A}]\!]\rho$, where ρ is as given in the last part of the Lemma. The inclusion $\supseteq$ is obvious. In the $\subseteq$ direction, let M be a message deducible from $[\![\mathfrak{A}]\!]\rho$, and π be a derivation of $[\![\mathfrak{A}]\!]\rho \mapsto M$: we show that M is in $[\![\mathfrak{A}]\!]\rho$ by an easy structural induction on π, using (2) in case $M = \mathtt{c}(M_1, K)$ is deduced by $(CryptI)$ to guarantee that K exists of type `key`, and using (1) in case π ends in $(CryptE)$. The latter is the only interesting case: by induction hypothesis there is a message $\mathtt{c}(M, K)$ recognized at *qm*, and an inverse K' of K such that $\mathtt{k}(K')$ is also recognized at *qm*; since $\mathtt{c}(M, K)$ is recognized at *qm*, either M is recognized at *qm* and K is recognized at $q_{\texttt{key}}$ (and we are done) or M is recognized at $q_{\texttt{msg}}$ and K is recognized at qk^{-1} (but then K' is recognized at qk, and is an inverse of K, which is impossible by (1)).

We can now show that: (4) $[\![\mathfrak{A}]\!]\rho$ is a good set, where ρ is as given in the last part of the Lemma. By (3) it is enough to show that no term that $(\dagger)$ specifies should not be deducible from $[\![\mathfrak{A}]\!]\rho$ is actually in $[\![\mathfrak{A}]\!]\rho$. This is by inspection of the rules.

We now wish to show that no strict superset of $[\![\mathfrak{A}]\!]\rho$ can be good. We first observe that: (5) if K exists of type `key` and is not recognized at qk^{-1}, then K has an inverse K' recognized at qk.

Conversely to (4), we claim: (6) for any set E of existing messages strictly containing $[\![\mathfrak{A}]\!]\rho$, E cannot be good. In this case, $E \setminus [\![\mathfrak{A}]\!]\rho$ is not empty, so $Ded(E) \setminus [\![\mathfrak{A}]\!]\rho$ is not empty either: let M be any message in $Ded(E) \setminus [\![\mathfrak{A}]\!]\rho$. We show by structural induction on M that there must be a term M_0 of type `msg` in $Ded(E)$ falling under one of the cases forbidden by $(\dagger)$. If M is of the form $\texttt{k}(K)$, then either K is not recognized at qk, so K is of the form $\texttt{symk}(k)$ with $k \notin [\![XK_0 \wedge \neg SSK_0]\!]\rho$ (but $k \in [\![XK_0]\!]\rho$ since M exists, so $k \in \mathcal{SSK}_0$, entailing that M is one of the terms forbidden by condition $(\dagger)$), or of the form $\texttt{asymk1}(k)$ (similar argument), or of the form $\texttt{asymk2}(k)$ (similar argument), or of the form $\texttt{sk}(M_1, M_2)$ or $\texttt{privk}(M_1)$ (but these are directly forbidden by $(\dagger)$). If M is of the form $\texttt{d}(d)$, then d is not recognized at qd, and since d exists, we must have $d \in \mathcal{SD}_0$, which is forbidden by $(\dagger)$. If M is of the form $\texttt{t}([M_1, \ldots, M_n])$, then since M is not recognized at qm, some M_j, $1 \leq j \leq n$, must fail to be recognized at qm; but since M is deducible from E, M_j is, too, using rule $(TupleE_j)$: we may therefore apply the induction hypothesis on M_j, so the claim obtains. Finally, if M is of the form $\texttt{c}(M_1, K)$, then since M is not recognized at qm: (a) M_1 is not recognized at qm or K is not recognized at $q_{\texttt{key}}$, and (b) M_1 is not recognized at $q_{\texttt{msg}}$ or K is not recognized at qk^{-1}. Since M exists, M_1 and K exist as well, so M_1 is recognized at $q_{\texttt{msg}}$ and K is recognized at $q_{\texttt{key}}$. So (a) and (b) entail: (c) M_1 is not recognized at qm and K is not recognized at qk^{-1}. By (5) K has an inverse K' recognized at qk, so $\texttt{k}(K')$ is recognized at qm, hence is in E. Using $(CryptE)$ and the fact that both M and $\texttt{k}(K')$ are deducible from E, $M_1 \in Ded(E)$: since M_1 is not recognized at qm by (c), we conclude by applying the induction hypothesis to M_1.

The last part of the Lemma then follows from (4) and (6). ☐

5 Simulating Protocol Runs

We shall analyze cryptographic protocols by doing an abstract interpretation [11] of programs in a simple, domain-specific language. We describe the syntax, the concrete semantics and the abstract semantics of this language, and prove that the abstract semantics is correct w.r.t. the concrete semantics. In Section 5.3, we shall indicate how we approximate the behavior of parallel multi-session principals.

5.1 A Simple Language for Cryptographic Protocols

We consider a language of *expressions*, *patterns* and *commands*. Expressions will denote messages, and in general terms; patterns will be used to match terms, and commands do the actual work. For simplicity, and although this entails a possible risk of confusion, we shall represent expressions as just terms over the basic signature Σ_0; note that, apart from simplicity, there is no special reason that forces us to do so. To avoid any confusion, we shall call the terms in $T_\tau(\Sigma, \mathcal{X})$ that we used to work on in previous sections the *values*. *Patterns* will be used to match values, and will also be coded as terms in a specific signature, enriched with the construction $\mathtt{exact}(e)$ (also written $\langle t \rangle$) whose intended meaning is to match *exactly* the value of expression e. Moreover, in patterns `c` will always be applied to a pattern and an expression: matching $\mathtt{c}(pat, e)$ against the value M means attempting to decrypt M using the value of e, then matching the resulting plaintext against pat. Formally:

Definition 8 (Expressions, Patterns). *Let $\mathcal{V}_\tau$, for each $\tau \in \mathcal{T}$, denote a countably infinite of so-called* program variables of type τ, *and* $\mathcal{V} \hat{=} \bigcup_{\tau \in \mathcal{T}} \mathcal{V}_\tau$.

The set $\mathcal{E}_\tau$ of expressions of type τ *is* $T_\tau(\Sigma_0, \mathcal{V})$.

The set $\mathcal{P}_\tau$ of patterns of type τ *is the family of sets defined by the grammar:*

$$
\begin{aligned}
\mathcal{P}_{\mathtt{msg}} &::= \mathtt{k}(\mathcal{P}_{\mathtt{key}}) \mid \mathtt{d}(\mathcal{P}_{\mathtt{D}}) \mid \mathtt{c}(\mathcal{P}_{\mathtt{msg}}, \mathcal{E}_{\mathtt{key}}) \mid \mathtt{t}(\mathcal{P}_{\mathtt{msglist}}) \mid \langle \mathcal{E}_{\mathtt{msg}} \rangle \mid \mathcal{V}_{\mathtt{msg}} \\
\mathcal{P}_{\mathtt{key}} &::= \mathtt{symk}(\mathcal{P}_{\mathtt{K}}) \mid \mathtt{asymk1}(\mathcal{P}_{\mathtt{K}}) \mid \mathtt{asymk2}(\mathcal{P}_{\mathtt{K}}) \mid * \mid \langle \mathcal{E}_{\mathtt{key}} \rangle \mid \mathcal{V}_{\mathtt{key}} \\
\mathcal{P}_{\mathtt{D}} &::= \langle \mathcal{E}_{\mathtt{D}} \rangle \mid \mathcal{V}_{\mathtt{D}} \\
\mathcal{P}_{\mathtt{K}} &::= \langle \mathcal{E}_{\mathtt{K}} \rangle \mid \mathcal{V}_{\mathtt{K}} \\
\mathcal{P}_{\mathtt{msglist}} &::= \mathtt{nil} \mid \mathtt{cons}(\mathcal{P}_{\mathtt{msg}}, \mathcal{P}_{\mathtt{msglist}})
\end{aligned}
$$

There is no $\langle \ldots \rangle$ or variable pattern of type `msglist`: they would not be usable anyway. Also, in practice, we assume that patterns are *linear*, in that they do not bind the same variable twice, to avoid confusing behavior; but we don't require it. Note finally that we don't have patterns of the form $\mathtt{sk}(pat_1, pat_2)$, $\mathtt{pubk}(pat)$ or $\mathtt{privk}(pat)$ which would allow one to recover the identities of the owners of long-term keys.

Commands are described in Figure 3. A *program* or a *role* is any finite list of commands. Note that programs don't have tests or loops. Indeed specifications of the roles of principals participating in cryptographic protocols are almost always of this form. Tests are only used to check conformance of formats of messages w.r.t. expected patterns, and in case of non-conformance the principal aborts. Loops never occur inside one session of the protocol, and will be discussed in Section 7.

$$
\begin{array}{rlll}
c \in \mathcal{C}om ::= & \mathtt{write}\, e & (e \in \mathcal{E}_{\mathtt{msg}}) & \text{output} \\
\mid & \mathtt{read}\, pat & (pat \in \mathcal{P}_{\mathtt{msg}}) & \text{input} \\
\mid & \mathtt{new}\, x & (x \in \mathcal{V}_{\mathtt{D}}) & \text{fresh data (nonces)} \\
\mid & \mathtt{new\_key}\, x & (x \in \mathcal{V}_{\mathtt{K}}) & \text{fresh key}
\end{array}
$$

Fig. 3. A simple language for cryptographic protocols

The concrete semantics of this language is a small-step operational semantics, and is as follows. The external world is modeled by an *external configuration* $\mathcal{W} \hat{=} (E, \mathcal{K}, \mathcal{D})$ where E is a set of messages known to intruders (more precisely, such that $Ded(E)$

is the set of messages that the intruders may produce), $\mathcal{K}$ is the current set of *existing raw keys*, $\mathcal{D}$ is the current set of *existing raw data*. The run-time environment of a given program is a mapping σ from $\mathcal{V}$ to values of the same type, i.e., $\sigma \in \biguplus_{\tau \in \mathcal{T}} (\mathcal{V}_\tau \to T_\tau(\Sigma, \mathcal{X}))$. So σ is a *substitution*, and evaluating an expression e in the environment σ returns the value $e\sigma$ obtained by replacing every variable x free in e by $\sigma(x)$. *Pattern-matching* is defined as the partial function $match(\sigma, pat, t)$ specified in Figure 4, which attempts to match t against pat in the environment σ and returns the environment, updated. We let $\sigma[x := t]$ denote the environment mapping x to t and every other variable y to $\sigma(y)$. Note that pattern-matching works from left to right (this comes in handy in situations where we match a tuple, get a key from the first component and use it to decrypt a message in the second component). In all cases where the rules of Figure 4 do not apply, $match(\sigma, pat, t)$ is undefined.

$$
\begin{array}{lll}
match(\sigma, \langle e \rangle, t) & \hat{=}\ \sigma & \text{(if } e\sigma = t\text{)} \\
match(\sigma, x, t) & \hat{=}\ \sigma[x := t] & \\
match(\sigma, \mathtt{c}(pat, e), \mathtt{c}(M, K)) & \hat{=}\ match(\sigma, pat, M) & \text{(if } e\sigma \text{ is an inverse of } K\text{)} \\
match(\sigma, f(), f()) & \hat{=}\ \sigma & (f \in \{*, \mathtt{nil}\}) \\
match(\sigma, f(pat), f(t)) & \hat{=}\ match(\sigma, pat, t) & \\
 & \multicolumn{2}{r}{(f \in \{\mathtt{k}, \mathtt{d}, \mathtt{t}, \mathtt{symk}, \mathtt{asymk1}, \mathtt{asymk2}, \mathtt{pubk}, \mathtt{privk}\})} \\
match(\sigma, f(pat_1, pat_2), f(t_1, t_2)) \ \hat{=} & & \\
\multicolumn{3}{r}{match(match(\sigma, pat_1, t_1), pat_2, t_2) \ (f \in \{\mathtt{cons}, \mathtt{sk}\})}
\end{array}
$$

Fig. 4. Pattern-matching

A *configuration* is a pair $\mathcal{W}, \sigma$ of an external configuration and a run-time environment. The *judgments* of the semantics relate configurations and are of the form $\mathcal{W}, \sigma \vdash c \Rrightarrow \mathcal{W}', \sigma'$ and are described in Figure 5. A *local run* of a role $[c_1, \dots, c_n]$ is a finite sequence of judgments of the form $\mathcal{W}_0, \sigma_0 \vdash c_1 \Rrightarrow \mathcal{W}_1, \sigma_1 \vdash c_2 \Rrightarrow \dots \Rrightarrow \mathcal{W}_{m-1}, \sigma_{m-1} \vdash c_m \Rrightarrow \mathcal{W}_m, \sigma_m$, where $0 \leq m \leq n$.

$$
\begin{array}{ll}
(E, \mathcal{K}, \mathcal{D}), \sigma \vdash \mathtt{write}\, e \Rrightarrow (E \cup \{e\sigma\}, \mathcal{K}, \mathcal{D}), \sigma & \\
(E, \mathcal{K}, \mathcal{D}), \sigma \vdash \mathtt{read}\, pat \Rrightarrow (E, \mathcal{K}, \mathcal{D}), match(\sigma, pat, M) & \\
\multicolumn{2}{r}{\text{(if } E \mid\!\!\rightarrow M \text{ and } match(\sigma, pat, M) \text{ is defined)}} \\
(E, \mathcal{K}, \mathcal{D}), \sigma \vdash \mathtt{new}\, x \Rrightarrow (E, \mathcal{K}, \mathcal{D} \cup \{d\}), \sigma[x := d] & (d \in T_{\mathtt{D}}(\Sigma, \mathcal{X}) \setminus \mathcal{D}) \\
(E, \mathcal{K}, \mathcal{D}), \sigma \vdash \mathtt{new\_key}\, x \Rrightarrow (E, \mathcal{K} \cup \{k\}, \mathcal{D}), \sigma[x := k] & (k \in T_{\mathtt{K}}(\Sigma, \mathcal{X}) \setminus \mathcal{K})
\end{array}
$$

Fig. 5. Concrete semantics

Given a set $\mathcal{R}$ of roles $R_i \hat{=} [c_{i1}, \dots, c_{in_i}]$, $i \in I$, a *global configuration* is a triple $(\mathcal{W}, \mathcal{R}, (\sigma_i)_{i \in I})$: $\mathcal{W}$ is the current external configuration, each $i \in I$ indexes some principal satisfying the role R_i, and for each principal $i \in I$, σ_i is the (local) run-time environment of principal i.

The semantics of a set $\mathcal{R}$ of roles is by interleaving. That is, we let $(\mathcal{W}, (R_i)_{i \in I}, (\sigma_i)_{i \in I}) \Rightarrow (\mathcal{W}', (R'_i)_{i \in I}, (\sigma'_i)_{i \in I})$ if and only if for some $i \in I$, $R_i = [c_{i1}, \dots, c_{in_i}]$ with $n_i \geq 1$, $R'_i = [c_{i2}, \dots, c_{in_i}]$, and it holds that $\mathcal{W}, \sigma_i \vdash c_{i1} \Rightarrow \mathcal{W}', \sigma'_i$; and for all $j \in I$, $j \neq i$, $R'_j = R_j$ and $\sigma'_j = \sigma_j$. A *run* is any finite sequence of global configurations such that each is related to the next by $\Rightarrow$. Our goal is, given a few initial assumptions about $\mathcal{W}$, to guarantee that no run starting from a global configuration where these assumptions are satisfied may reveal any message whose confidentiality we want to see preserved.

5.2 Abstract Semantics

This goal will be achieved simply by replacing actual sets of messages known by intruders by $\vee$PTAs, and by assumptions (in the sense of Definition 3) expressing relations between various sets of raw keys and raw data. The description of the abstract semantics mimics that of the concrete semantics.

To model freshness, we first require that a propositional variable X of type `D`, resp. `K`, be associated with each `new` or `new_key` command respectively in each principal. To emphasize this binding, we shall write $\texttt{new}_X$ or $\texttt{new\_key}_X$ in the abstract semantics. These variables need not be distinct, and will be used to denote sets of raw data, resp. raw keys, that may ever be produced by the designated command. We shall express that these commands create *fresh* data, resp. raw keys, by adding to the assumption $\mathfrak{H}$ all the elementary assumptions $X \cap Y = \emptyset$, for every such pair of distinct variables X and Y. We write $[X]$ the $\vee$PTA $(\{q\}, \{q\}, \emptyset, \{q \mapsto X\})$, where q has the same type as X: in ρ, $[X]$ recognizes exactly the set $[\![X]\!]\rho_\tau$, where τ is the type of X. Call such propositional variables *freshness variables*.

An *abstract external configuration* $\mathcal{W}^\sharp$ is a 4-tuple $(\mathfrak{H}, \mathfrak{A}, XK, XD)$ where $\mathfrak{H}$ is an assumption, $\mathfrak{A}$ is an $\vee$PTA, XK is a formula of type `K` and XD is a formula of type `D`.

An *abstract run-time environment* $\sigma^\sharp$ is a map from program variables to normal $\vee$PTAs that respects the types, i.e., for every $x \in \mathcal{V}_\tau$, $\sigma^\sharp(x)$ is an $\vee$PTA of type τ. The intent is that $\sigma^\sharp(x)$ recognizes all possible values that x may take under the assumption $\mathfrak{H}$. To evaluate an expression in a given abstract run-time environment, define:

Lemma 10 (Eval). *For any expression e of type τ, for any abstract run-time environment $\sigma^\sharp$, let $Eval(e)\sigma^\sharp$ be the following normal $\vee$PTA $(\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$. For any $x \in \mathcal{V}$, let $\sigma^\sharp(x)$ be written $(\mathfrak{Q}_x, \mathfrak{F}_x, \mathfrak{R}_x, \mathfrak{B}_x)$. For each $q \in \mathfrak{Q}_x$, let $\langle x, q\rangle$ be a state of type τ_q chosen such that $\langle x, q\rangle = \langle y, q'\rangle$ implies $x = y$ and $q = q'$. For any subterm e' of e, let $q_{e'}$ be some state of the same type as e', chosen such that $q_{e'} = q_{e''}$ implies $e' = e''$, and no $q_{e'}$ equals any $\langle x, q\rangle$. Then:*

- $\mathfrak{Q} \hat{=} \{\langle x, q\rangle \mid x \text{ free in } e\} \cup \{q_{e'} \mid e' \text{ subterm of } e\}$;
- $\mathfrak{F} \hat{=} \{q_e\}$;
- $\mathfrak{R} \hat{=} \{f(\langle x, q_1\rangle, \dots, \langle x, q_n\rangle) \to \langle x, q\rangle \mid x \text{ free in } e, f(q_1, \dots, q_n) \to q \in \mathfrak{R}_x\}$
  $\cup \{f(\langle x, q_1\rangle, \dots, \langle x, q_n\rangle) \to q_x \mid x \text{ free in } e, f(q_1, \dots, q_n) \to q \in \mathfrak{R}_x, q \in \mathfrak{F}_x\}$
  $\cup \{f(q_{e_1}, \dots, q_{e_n}) \to q_{f(e_1, \dots, e_n)} \mid f(e_1, \dots, e_n) \text{ subterm of } e\}$

- *$\mathfrak{B}$ maps each state $\langle x, q\rangle$ to $\mathfrak{B}_x(q)$, each state q_x with x free in e to $\bigvee_{q\in\mathfrak{F}_x} \mathfrak{B}_x(q)$, and every other state to* **0**.

For every environment ρ, for every run-time environment σ, if for every variable x free in e $\sigma(x)$ is recognized by $\sigma^\sharp(x)$ in ρ then $e\sigma$ is recognized by $Eval(e)\sigma^\sharp$ in ρ.

Proof. By induction on t: $(*)$ if t is recognized by $\sigma^\sharp(x)$ in ρ at q, then it is recognized by $Eval(e)\sigma^\sharp$ in ρ at $\langle x, q\rangle$.

Now we claim that: $(**)$ for every x free in e, $\sigma(x)$ is recognized by $Eval(e)\sigma^\sharp$ in ρ at q_x. Indeed, this is the case if $\sigma(x) \in [\![\mathfrak{B}_x(q)]\!]\rho_\tau$ for some $q \in \mathfrak{F}_x$ (where τ is the type of x), by definition; otherwise, there must be a transition $f(q_1, \ldots, q_n) \to q$ with $q \in \mathfrak{F}_x$ in $\sigma^\sharp(x)$ such that $\sigma(x)$ is of the form $f(t_1, \ldots, t_n)$ and each t_j is recognized by $\sigma^\sharp(x)$ in ρ at q_j, $1 \leq j \leq n$. By $(*)$ each t_j is recognized by $Eval(e)\sigma^\sharp$ in ρ at $\langle x, q_j\rangle$, so $\sigma(x) = f(t_1, \ldots, t_n)$ is recognized by $Eval(e)\sigma^\sharp$ in ρ at q_x.

A straightforward structural induction on the subterm e' of e then shows that $e'\sigma$ is recognized by $Eval(e')\sigma^\sharp$ in ρ at $q_{e'}$, whence the result. ☐

An *abstract configuration* is a pair $\mathcal{W}^\sharp, \sigma^\sharp$ where $\mathcal{W}^\sharp$ is an abstract external configuration $(\mathfrak{H}, \mathfrak{A}, XK, XD)$ and $\sigma^\sharp$ is an abstract run-time environment. Its *meaning* is the set of configurations $\gamma(\mathcal{W}^\sharp, \sigma^\sharp) \hat{=} \{(E, \mathcal{K}, \mathcal{D}), \sigma \mid \rho \models \mathfrak{H}, E \subseteq [\![\mathfrak{A}]\!]\rho, \mathcal{K} \subseteq [\![XK]\!]\rho_{\mathtt{K}}, \mathcal{D} \subseteq [\![XD]\!]\rho_{\mathtt{D}}, \forall x \in \mathcal{V} \cdot \sigma(x) \in [\![\sigma^\sharp(x)]\!]\rho\}$.

For every finite family of abstract run-time environments $(\sigma_i^\sharp)_{i\in I}$, define $\bigsqcup^*_{i\in I} \sigma^\sharp$ as the abstract run-time environment mapping each x to $\bigsqcup_{i\in I} \sigma^\sharp(x)$.

By extension of Lemma 7, say that two $\vee$PTAs $\mathfrak{V}_1$ and $\mathfrak{V}_2$ *possibly intersect* if and only if $\mathfrak{V}_1 \sqcap \mathfrak{V}_2$ is not definitely empty; and that $\mathfrak{V}_1 \hat{=} (\mathfrak{Q}_1, \mathfrak{F}_1, \mathfrak{R}_1, \mathfrak{B}_1)$ *contains possible inverses* of $\mathfrak{V}_2 \hat{=} (\mathfrak{Q}_2, \mathfrak{F}_2, \mathfrak{R}_2, \mathfrak{B}_2)$ if and only if there are transitions $f_1(q_{11}, \ldots, q_{1n}) \to q_1$ in $\mathfrak{R}_1$, with $q_1 \in \mathfrak{F}_1$, and $f_2(q_{21}, \ldots, q_{2n}) \to q_2$ in $\mathfrak{R}_2$, with $q_2 \in \mathfrak{F}_2$, such that $(\mathfrak{Q}_1, \{q_{1j}\}, \mathfrak{R}_1, \mathfrak{B}_1)$ and $(\mathfrak{Q}_2, \{q_{2j}\}, \mathfrak{R}_2, \mathfrak{B}_2)$ intersect possibly for every $1 \leq j \leq n$, where f_1, f_2 and n are given in the table of Lemma 7. Properties (i) and (ii) of the latter lemma still hold: (i) if two $\vee$PTAs recognize the same term in ρ, then they intersect possibly, and (ii) if $\mathfrak{V}_1$ recognizes some key K in ρ and $\mathfrak{V}_2$ recognizes some inverse key K' in ρ, then $\mathfrak{V}_2$ contains possible inverses of $\mathfrak{V}_1$.

Abstract pattern-matching is defined just like concrete pattern-matching, replacing actual values by $\vee$PTAs, and equality tests between values t_1 and t_2 by checks that two $\vee$PTAs intersect possibly. Figure 6 defines the corresponding partial function $match^\sharp(\sigma^\sharp, pat, \mathfrak{V})$, where $\mathfrak{V} \hat{=} (\mathfrak{Q}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B})$ is an $\vee$PTA of the same type as pat.

Abstract pattern-matching is a correct approximation of pattern-matching:

Lemma 11. *If $\sigma' = match(\sigma, pat, t)$ is defined, t is recognized by the normal $\vee$PTA $\mathfrak{V}$ in ρ, and for every $x \in \mathcal{V}$ $\sigma(x) \in [\![\sigma^\sharp(x)]\!]\rho$, then $match^\sharp(\sigma^\sharp, pat, \mathfrak{V})$ is defined, and writing it ${\sigma'}^\sharp$, it obtains that for every $x \in \mathcal{V}$, $\sigma'(x) \in [\![{\sigma'}^\sharp(x)]\!]\rho$.*

Proof. By structural induction on pat. If pat is of the form $\langle e\rangle$, then by assumption $e\sigma = t$ and $\sigma' = \sigma$: but then $e\sigma$ is in $[\![Eval(e)\sigma^\sharp]\!]\rho$ by Lemma 10, so $Eval(e)\sigma^\sharp$ and $\mathfrak{V}$ intersect possibly, and the claim follows. We only deal with the case where pat is of the form $\mathtt{c}(pat', e)$, then t must be of the form $\mathtt{c}(M, K)$ with $e\sigma$ an inverse of K. Since $\mathfrak{V}$ is normal and $\mathtt{c}(M, K)$ is recognized by $\mathfrak{V}$ in ρ, there must be a transition

$$
\begin{array}{ll}
\mathit{match}^\sharp(\sigma^\sharp, \langle e \rangle, \mathfrak{V}) & \hat{=}\ \sigma^\sharp \quad \text{(if } \mathit{Eval}(e)\sigma^\sharp \text{ and } \mathfrak{V} \text{ intersect possibly)} \\
\mathit{match}^\sharp(\sigma^\sharp, x, \mathfrak{V}) & \hat{=}\ \sigma^\sharp[x := \mathfrak{V}] \\
\mathit{match}^\sharp(\sigma^\sharp, \mathtt{c}(\mathit{pat}, e), \mathfrak{V}) & \hat{=}\ \mathit{match}^\sharp(\sigma^\sharp, \mathit{pat}, \mathfrak{V}') \\
\multicolumn{2}{r}{\text{(where } \mathfrak{V}' \hat{=} \bigsqcup(\mathfrak{Q}, \{q_1\}, \mathfrak{R}, \mathfrak{B}) \text{ and the union is over}} \\
\multicolumn{2}{r}{\text{all } \mathtt{c}(q_1, q_2) \to q \in \mathfrak{R} \text{ such that } q \in \mathfrak{F}} \\
\multicolumn{2}{r}{\text{and } \mathit{Eval}(e)\sigma^\sharp \text{ contains possible inverses of } (\mathfrak{Q}, \{q_2\}, \mathfrak{R}, \mathfrak{B}))} \\
\mathit{match}^\sharp(\sigma^\sharp, f(), \mathfrak{V}) & \hat{=}\ \sigma^\sharp \\
\multicolumn{2}{r}{(f \in \{*, \mathtt{nil}\}, \mathit{Eval}(f)\sigma^\sharp \text{ and } \mathfrak{V} \text{ intersect possibly)}} \\
\mathit{match}^\sharp(\sigma^\sharp, f(\mathit{pat}), \mathfrak{V}) & \hat{=}\ \mathit{match}^\sharp(\sigma^\sharp, \mathit{pat}, \mathfrak{V}') \\
\multicolumn{2}{r}{\text{(where } f \in \{\mathtt{k}, \mathtt{d}, \mathtt{t}, \mathtt{symk}, \mathtt{asymk1}, \mathtt{asymk2}, \mathtt{pubk}, \mathtt{privk}\},} \\
\multicolumn{2}{r}{\mathfrak{V}' \hat{=} \bigsqcup(\mathfrak{Q}, \{q_1\}, \mathfrak{R}, \mathfrak{B}) \text{ and the union is over}} \\
\multicolumn{2}{r}{\text{all } f(q_1) \to q \in \mathfrak{R} \text{ such that } q \in \mathfrak{F})} \\
\mathit{match}^\sharp(\sigma^\sharp, f(\mathit{pat}_1, \mathit{pat}_2), \mathfrak{V}) & \hat{=}\ \bigsqcup^*_{i \in I} \mathit{match}^\sharp(\sigma^\sharp_i, \mathit{pat}_2, \mathfrak{V}_{2i}) \\
\multicolumn{2}{r}{\text{(where } f \in \{\mathtt{cons}, \mathtt{sk}\}, I \text{ is the set of transitions } f(q_{1i}, q_{2i}) \to q_i \text{ with } q_i \in \mathfrak{F}} \\
\multicolumn{2}{r}{\text{such that } \sigma^\sharp_i \hat{=} \mathit{match}^\sharp(\sigma^\sharp, \mathit{pat}_1, \mathfrak{V}_{1i}) \text{ is defined,}} \\
\multicolumn{2}{r}{\text{and such that } \mathit{match}^\sharp(\sigma^\sharp_i, \mathit{pat}_2, \mathfrak{V}_{2i}) \text{ is defined,}} \\
\multicolumn{2}{r}{\text{where } \mathfrak{V}_{1i} \hat{=} (\mathfrak{Q}, \{q_{1i}\}, \mathfrak{R}, \mathfrak{B}), \mathfrak{V}_{2i} \hat{=} (\mathfrak{Q}, \{q_{2i}\}, \mathfrak{R}, \mathfrak{B}))}
\end{array}
$$

Fig. 6. Abstract pattern-matching

$\mathtt{c}(q_1, q_2) \to q$ in $\mathfrak{R}$ with $q \in \mathfrak{F}$, such that M is recognized at q_1 and K is recognized at q_2. Since $e\sigma$ is an inverse of K and is recognized by $\mathit{Eval}(e)\sigma^\sharp$ in ρ by Lemma 10, $\mathit{Eval}(e)\sigma^\sharp$ contains possible inverses of $(\mathfrak{Q}, \{q_2\}, \mathfrak{R}, \mathfrak{B})$. In any case M is recognized by the union $\bigsqcup(\mathfrak{Q}, \{q_1\}, \mathfrak{R}, \mathfrak{B})$ as specified in Figure 6, whence the claim. ☐

The judgments of the abstract semantics have the form $\mathcal{W}^\sharp, \sigma^\sharp \vdash c \Rrightarrow \mathcal{W}'^\sharp, \sigma'^\sharp$, and are copied from those of the concrete semantics: see Figure 7.

$$
\begin{array}{l}
(\mathfrak{H}, \mathfrak{A}, XK, XD), \sigma^\sharp \vdash \mathtt{write}\, e \Rrightarrow (\mathfrak{H}, \mathit{Ded}(\mathfrak{A} \sqcup \mathit{Eval}(e)\sigma^\sharp), XK, XD), \sigma^\sharp \\
(\mathfrak{H}, \mathfrak{A}, XK, XD), \sigma^\sharp \vdash \mathtt{read}\, \mathit{pat} \Rrightarrow (\mathfrak{H}, \mathfrak{A}, XK, XD), \mathit{match}^\sharp(\sigma^\sharp, \mathit{pat}, \mathfrak{A}_{|\mathfrak{H}}) \\
\hfill \text{(if } \mathit{match}^\sharp(\sigma^\sharp, \mathit{pat}, \mathfrak{A}_{|\mathfrak{H}}) \text{ is defined)} \\
(\mathfrak{H}, \mathfrak{A}, XK, XD), \sigma^\sharp \vdash \mathtt{new}_X\, x \Rrightarrow (\mathfrak{H}, \mathfrak{A}, XK, XD), \sigma^\sharp[x := [X]] \\
(\mathfrak{H}, \mathfrak{A}, XK, XD), \sigma^\sharp \vdash \mathtt{new\_key}_X\, x \Rrightarrow (\mathfrak{H}, \mathfrak{A}, XK, XD), \sigma^\sharp[x := [X]]
\end{array}
$$

Fig. 7. Abstract semantics

An *abstract global configuration* is a triple $(\mathcal{W}^\sharp, (R_i)_{i \in I}, (\sigma^\sharp_i)_{i \in I})$ consisting of an abstract external configuration, a family of roles and a family of corresponding abstract run-time environments. Finally, *abstract runs* are defined as finite sequences of global configurations, where we go from one to the next by executing one command of one principal, as in the concrete case.

The abstract semantics is correct w.r.t. the concrete semantics:

Lemma 12. *Let* $\mathcal{R} \hat{=} (R_i)_{i \in I}$ *be a family of roles, and consider any run* $(\mathcal{W}_0, \mathcal{R}_0, (\sigma_{0i})_{i \in I}) \Rrightarrow (\mathcal{W}_1, \mathcal{R}_1, (\sigma_{1i})_{i \in I}) \Rrightarrow \ldots \Rrightarrow (\mathcal{W}_n, \mathcal{R}_n, (\sigma_{ni})_{i \in I})$.

Assume that all freshness variables are distinct from XK_0, XD_0, SSK_0, $SAK1_0$, $SAK2_0$, SD_0. *For each freshness variable* X *of type* `D` *(resp.* `K`*), let* $\rho(X)$ *be the set of raw data (resp. raw keys) created by* $\texttt{new}_X$ *commands (resp.* $\texttt{new\_key}_X$ *commands) during the run, i.e., such that for some* j, $1 \leq j \leq n$, *for some* $i \in I$, *for some* $x \in \mathcal{V}$, $\mathcal{W}_{j-1}, \sigma_{j-1\ i} \vdash \texttt{new}_X\, x$ *(resp.* $\texttt{new\_key}_X\, x$*)* $\Rrightarrow \mathcal{W}_j, \sigma_{ji}$. *Let* ρ *also map* XK_0, XD_0, SSK_0, $SAK1_0$, $SAK2_0$, SD_0 *to the sets indicated in Lemma 9.*

Let $\mathfrak{H}$ *be the conjunction of the assumptions* $X \cap Y = \emptyset$, *where* X *and* Y *range over pairs of distinct variables of the same type among the freshness variables,* XK_0 *and* XD_0. *Then* $\rho \models \mathfrak{H}$.

Let $\mathfrak{A}_0 \hat{=} \mathfrak{Known}(XK_0, XD_0, SSK_0, SAK1_0, SAK2_0, SD_0)$. *Let* XK *(resp.* XD*) be the disjunction of all freshness variables of type* `K` *(resp.* `D`*) and of* XK_0 *(resp.* XD_0*). Let* $(\sigma^\sharp_{0i})_{i \in I}$ *be any family of abstract run-time environments such that for all* $i \in I$, *for all* $x \in \mathcal{V}$, $\sigma_{0i}(x) \in [\![\sigma^\sharp_{0i}(x)]\!]\rho$. *Then there is an abstract run* $(\mathcal{W}^\sharp_0, \mathcal{R}_0, (\sigma^\sharp_{0i})_{i \in I}) \Rrightarrow (\mathcal{W}^\sharp_1, \mathcal{R}_1, (\sigma^\sharp_{1i})_{i \in I}) \Rrightarrow \ldots \Rrightarrow (\mathcal{W}^\sharp_n, \mathcal{R}_n, (\sigma^\sharp_{ni})_{i \in I})$, *such that for each* j, $0 \leq j \leq n$, *for every* $i \in I$*:*

$$(\mathcal{W}_j, \sigma_{ji}) \in \gamma(\mathcal{W}^\sharp_j, \sigma^\sharp_{ji}) \tag{1}$$

Proof. That $\rho \models \mathfrak{H}$ follows from the easily proved fact that no data or raw key created at any step of the run equals any data or raw key created at some other step or existing at the start of the run.

We show (1) by induction on j. More precisely, write each $\mathcal{W}_j$ as $(E_j, \mathcal{K}_j, \mathcal{D}_j)$ and build $\mathcal{W}^\sharp_j$ as $(\mathfrak{H}, \mathfrak{A}_j, XK, XD)$. By construction we have: (a) $\mathcal{K}_j \subseteq [\![XK]\!]\rho_{\texttt{K}}$, and (b) $\mathcal{D}_j \subseteq [\![XD]\!]\rho_{\texttt{K}}$. We now build $\mathfrak{A}_j$ by induction on j so that: (c) $Ded(E_j) \subseteq [\![\mathfrak{A}_j]\!]\rho$, and (d) $\sigma_{ji}(x) \in [\![\sigma^\sharp_{ji}(x)]\!]\rho$ for every $i \in I$, $x \in \mathcal{V}$. From (c) it follows: (e) $E_j \subseteq [\![\mathfrak{A}_j]\!]\rho$. So from the first part of the Lemma, (e), (a), (b) and (d) we will be able to infer (1).

When $j = 0$, (c) is by Lemma 9, (d) is by assumption, and (e) follows from the fact (see the proof of Lemma 9) that the messages deducible from the ones recognized by $\mathfrak{A}_0$ in ρ are exactly those recognized by $\mathfrak{A}_0$ in ρ. When $j \geq 1$, then let i_0 be the principal which did a transition, i.e., $\mathcal{W}_{j-1}, \sigma_{j-1\ i_0} \vdash c \Rrightarrow \mathcal{W}_j, \sigma_{ji_0}$. Then the abstract semantic rule is dictated by c:

- If $c = \texttt{write}\, e$, then $E_{j+1} = E_j \cup \{e\sigma_{ji_0}\}$ by induction hypothesis (c), so by Lemma 10 $E_{j+1} \subseteq [\![\mathfrak{A}_{j-1} \sqcup Eval(e)\sigma^\sharp_{j-1\ i_0}]\!]\rho$. By Theorem 2 $Ded(E_{j+1}) \subseteq Ded([\![\mathfrak{A}_{j-1} \sqcup Eval(e)\sigma^\sharp_{j-1\ i_0}]\!]\rho) \subseteq [\![Ded(\mathfrak{A}_{j-1} \sqcup Eval(e)\sigma^\sharp_{j-1\ i_0})]\!]\rho = [\![\mathfrak{A}_j]\!]\rho$, so (c) holds; and (d) is by induction hypothesis.
- If $c = \texttt{read}\, pat$, then by Lemma 6, since $\rho \models \mathfrak{H}$, $\mathfrak{A}_{j-1}$ and $\mathfrak{A}_{j-1|\mathfrak{H}}$ recognize the same messages in ρ. An easy structural induction on pat then shows that $match^\sharp(\sigma^\sharp_{j-1\ i_0}, pat, \mathfrak{A}_{j-1|\mathfrak{H}})(x)$ and $match^\sharp(\sigma^\sharp_{j-1\ i_0}, pat, \mathfrak{A}_{j-1})(x)$ recognize the same terms in ρ for every x. By Lemma 11, using the induction hypothesis and $\sigma^\sharp_{ji_0} = match^\sharp(\sigma^\sharp_{j-1\ i_0}, pat, \mathfrak{A}_{j-1|\mathfrak{H}})$, it follows that (d) holds when $i = i_0$. (The point in using $\mathfrak{A}_{|\mathfrak{H}}$ instead of $\mathfrak{A}$ is that it provides more accurate pattern-matching.) The cases $i \neq i_0$ and item (c) are by induction hypothesis.

- The cases $c = \texttt{new}_X\ x$ and $c = \texttt{new\_key}_X\ x$ are by construction. $\square$

So the abstract semantics always produces, at each step j, an upper approximation $[\![\mathfrak{A}_j]\!]\rho$ of the set of messages deducible from the set E_j of messages that the intruders have got. It follows that if $[\![Eval(e)\sigma_j^\sharp]\!]\rho$ and $[\![\mathfrak{A}_j]\!]\rho$ do not possibly intersect, then $e\sigma_j$ is necessarily secret. This is easy to test: we instrument each role $i \in I$ with *secrecy tests* of the form $\texttt{secret}(e)$, where $e \in \mathcal{E}_{\texttt{msg}}$, and simulate all possible abstract runs (all possible interleavings): whenever the simulation of some interleaving runs across a secrecy test for i, it computes $Eval(e)\sigma_j^\sharp \sqcap \mathfrak{A}_{j_{|\mathfrak{H}}}$; if the result is definitely empty, then at this step in this interleaving the intruders cannot forge the value of e. This is the guarantee that we were claiming we would look after in the introduction.

A few remarks are in order. First, we can also check some freshness conditions. This is already what we do in pattern-matching: when we create a nonce by $\texttt{new}_X\ N_a$, then pattern-match a value M we just read against patterns like $\langle N_a \rangle$, what we really check in the abstract semantics is that M was a piece of data that was created in the temporal window represented by the logical variable X.

Second, instead of testing all interleavings, we can use *partial order reductions*, i.e., restrict to particular interleavings that still express all possible cases. It is not hard to see that we can always permute the execution of commands from two different principals, provided these commands are neither `write`s nor `read`s. In fact, if one of the commands is a `write` and the other is not a `read`, we can again permute them without changing the semantics. Finally, if principal i reads a value, then principal i' writes another value, we may also permute these to have i' write first, then i read: this changes the semantics, however this only increases the set $Ded(E)$ of messages deducible by intruders. So, if the interleaving that does the write first, then the read keeps a given set of values secret, then the other one does, too. To sum up, we can always force the deterministic execution of all commands of all principals except `read`s. The only required non-determinism in the abstract execution of the protocols is in the order in which `read`s are performed by the various principals.

5.3 Simulating Parallel Multi-Session Principals

The abstract semantics of Section 5.2 allows us to check secrecy properties for finite families of principals running in parallel, but it ignores the important case of *parallel multi-session principals*. In π-calculus notation, such a principal obeys roles of the form $!S$, i.e. it runs an unbounded number of copies of some role S in parallel. For instance, in the Otway-Rees protocol [26], the two principals A and B that wish to communicate attempt to do so with the help of a key server that obeys the role consisting of the following three commands:

$$\begin{array}{l}
\texttt{read}\ \texttt{t}([\texttt{d}(M), \texttt{d}(id_A), \texttt{d}(id_B),\\
\qquad\qquad \texttt{c}(\texttt{t}([\texttt{d}(N_a), \langle\texttt{d}(M)\rangle, \langle\texttt{d}(id_A)\rangle, \langle\texttt{d}(id_B)\rangle]), \texttt{sk}(\texttt{d}(id_A), \texttt{d}(id_S))),\\
\qquad\qquad \texttt{c}(\texttt{t}([\texttt{d}(N_b), \langle\texttt{d}(M)\rangle, \langle\texttt{d}(id_A)\rangle, \langle\texttt{d}(id_B)\rangle]), \texttt{sk}(\texttt{d}(id_B), \texttt{d}(id_S)))]);\\
\texttt{new\_key}_X\ K_{ab};\\
\texttt{write}\ \texttt{t}([\texttt{d}(M), \texttt{c}(\texttt{t}([\texttt{d}(N_a), \texttt{k}(\texttt{symk}(K_{ab}))]), \texttt{sk}(\texttt{d}(id_A), \texttt{d}(id_S))),\\
\qquad\qquad\qquad \texttt{c}(\texttt{t}([\texttt{d}(N_b), \texttt{k}(\texttt{symk}(K_{ab}))]), \texttt{sk}(\texttt{d}(id_B), \texttt{d}(id_S)))])
\end{array}$$

where id_S of type `D` denotes the identity of the server S. However, the server does not obey this role exactly, but spawns one new thread obeying the role above for each message sent to it. In effect, everything works as if we had an unbounded number of copies of S running in parallel, all sharing the same identity id_S.

As it is impossible to simulate an unbounded number of principals in the method of Section 5.2, we shall seek a safe, computable upper approximation of the behaviors of intruders together with parallel multi-session principals. As in [12], we shall consider that, as far as security properties are concerned, such principals work as accomplices to the intruders, and only contribute to the verification in as much as they offer new deduction capabilities to the intruder. To take a simple example, assume that some other server S' expects a message of the form $\{M\}_{K_{bs}}$ and then writes M in clear, then the intruder can hire S' to decrypt messages with the otherwise secret key K_{bs}.

In general, our problem is: given an $\vee$PTA $\mathfrak{A}$ and a set of roles $S_1^*, \ldots, S_n^*$, find an $\vee$PTA $\mathfrak{A}'$ that recognizes any message M that may be deducible by the intruders after any run involving any number of copies of $S_1^*, \ldots, S_n^*$, starting from the set of messages $E \hat{=} [\![\mathfrak{A}]\!]\rho$. For short, we say informally that M is *deducible in the extended sense*. Computing $\mathfrak{A}'$ involves some approximation:

Definition 9 (Components [4]). *For any set E of terms, the set $Comp(E)$ of* components of *E is the smallest set of terms containing E such that, for any $f(t_1, \ldots, t_n) \in Comp(E)$, for every j, $1 \leq j \leq n$, $t_j \in Comp(E)$. Let $Comp_\tau(E)$ be the set of components of E of type τ.*

Notice that every message extractible from E is a component of E. The notion of components is used to approximate the behavior of pattern-matching.

Let us formalize the *abstract semantic equations* that we shall solve later.

Definition 10. *An* abstract semantics equation *is a formal expression of the form:*

1. $\mathsf{Ded}(q)$ *where q is a state of type* `msg`, `D`, `key` *or* `K` *("terms recognized at q are possibly deducible in the extended sense");*
2. *or* $\mathsf{Comp}(q)$ *where q is any state ("terms recognized at q are possible components of terms deducible in the extended sense");*
3. *or $q = f(q_1, \ldots, q_n)$ where $f : \tau_1 \times \ldots \times \tau_n \to \tau$, q is of type τ, and each q_j is of type τ_j, $1 \leq j \leq n$;*
4. *or $q \in \mathfrak{A}$, where $\mathfrak{A}$ is some $\vee$PTA of the same type as the state q*

Given a set of messages E, say that the term d of type `D` *is* deducible *from E if and only if $\mathtt{d}(d) \in Ded(E)$ (let $Ded_{\mathtt{D}}(E)$ be the set of such terms); that K of type* `key` *is* deducible *from E if and only if $\mathtt{k}(K) \in Ded(E)$ (let $Ded_{\mathtt{key}}(E)$ be the set of such terms); and that k of type* `K` *is* deducible *from E if and only if $\mathtt{symk}(k)$, $\mathtt{asymk1}(k)$ or $\mathtt{asymk2}(k)$ is in $Ded_{\mathtt{key}}(E)$ (let $Ded_{\mathtt{K}}(E)$ be the set of such terms). By extension, let $Ded_{\mathtt{msg}}(E)$ be $Ded(E)$.*

Given a set E of messages, a map κ from states q to sets of terms of type τ_q, an environment ρ, and an abstract semantic equation eq, we say that E, κ, ρ satisfies *eq, in short $E, \kappa, \rho \models eq$, if and only if:*

1. *$eq = \mathsf{Ded}(q)$ and $\kappa(q) \supseteq Ded_{\tau_q}(E)$ and if $\tau_q = \mathtt{msg}$, then $\kappa(q) \supseteq Ded(\kappa(q))$;*

2. *or* $eq = \mathsf{Comp}(q)$ *and* $\kappa(q) \supseteq Comp_{\tau_q}(Ded_{\tau_q}(E))$*;*
3. *or* $eq = (q = f(q_1, \dots, q_n))$ *and* $\kappa(q) \supseteq \{f(t_1, \dots, t_n) \mid t_1 \in \kappa(q_1), \dots, t_n \in \kappa(q_n)\}$*;*
4. *or* $eq = (q \in \mathfrak{A})$ *and* $\kappa(q) \supseteq [\![\mathfrak{A}]\!]\rho$*.*

A system Eqs *of abstract semantic equations is a finite set of abstract semantic equations; we let* $E, \sigma, \rho \models Eqs$ *if and only if* $E, \sigma, \rho \models eq$ *for every* $eq \in Eqs$*.*

Assume for simplicity that each variable is bound at most once in the whole set of parallel multi-session principals $S_1^*, \dots, S_n^*$. Map each variable x occurring free or bound in $S_1^*, \dots, S_n^*$ to some state q_x of the same type as x. Then do pattern-matching in the following very crude way:

Definition 11 (Crude Pattern Matching). *For any pattern pat, the set* $\mathrm{bv}(pat)$ *of* variables bound by *pat is defined by:* $\mathrm{bv}(x) \hat{=} \{x\}$, $\mathrm{bv}(\langle e \rangle) \hat{=} \emptyset$, $\mathrm{bv}(\mathtt{c}(pat, e)) \hat{=} \mathrm{bv}(pat)$*, and* $\mathrm{bv}(f(pat_1, \dots, pat_n)) \hat{=} \bigcup_{j=1}^{n} \mathrm{bv}(pat_j)$ *for every other function symbol* f*.*

For any variable x *bound by pat, say that* x *is* visible *in pat if and only if* $pat = x$*, or pat is not of the form* $\langle e \rangle$ *and is of the form* $f(pat_1, \dots, pat_n)$ *with* $f \neq \mathtt{c}$ *and* x *visible in some* pat_j*,* $1 \leq j \leq n$ *(i.e., the ones that are not visible are those that occur only inside encryption patterns).*

Let $Eqs(pat)$ *be the set of abstract semantic equations of the form* $\mathsf{Ded}(q_x)$*, where* x *is visible in pat, or of the form* $\mathsf{Comp}(q_x)$*, where* x *is bound but not visible in pat.*

Writing a message is approximated as in Lemma 14. First, notice the following:

Lemma 13. *Let* E *be a set of messages, and* E_0 *be a set of messages containing* $Ded(E)$ *and* M*. If in addition* $E_0 \supseteq Ded(E_0)$*, then* E_0 *contains* $Ded(E \cup \{M\})$*.*

Proof. Let M' be any message deducible from $E \cup \{M\}$. We show by structural induction on a derivation π of $E, M \models M'$ that M' is in E_0. If π ends in (Ax), then M' is in E or equals M, and this is clear. If π ends in $(CryptI)$, then M' is of the form $\mathtt{c}(M_1', K')$ with $E, M \models M_1'$ and $E, M \models \mathtt{k}(K')$, so by induction hypothesis M_1' and $\mathtt{k}(K')$ are in E_0; since $E_0 \supseteq Ded(E_0)$, M' is in E_0. All other cases are similar. ☐

Lemma 14 (Crude Output). *Let* E' *be* $E \cup \{e\sigma\}$ *as specified in the semantics of the* `write` *command, where* e *is of type* `msg`*. For each non-variable sub-expression* e' *of* e*, let* $q_{e'}$ *be some state of the same type as* e'*.*

Let $Eqs(e)$ *be the set of abstract semantic equations* $q_{f(q_{e_1}, \dots, q_{e_n})} = f(q_{e_1}, \dots, q_{e_n})$*, where* $f(e_1, \dots, e_n)$ *ranges over all non-variable subexpressions of* e*, plus the abstract semantic equation* $\mathsf{Ded}(q_e)$*.*

Then whenever $\sigma(x) \in \kappa(q_x)$ *for every* x *free in* e *and* $E, \kappa, \rho \models Eqs(e)$*, then* $E', \kappa, \rho \models Eqs(e)$*.*

Proof. Since E, κ, ρ satisfies all equations of the form $q_{f(q_{e_1}, \dots, q_{e_n})} = f(q_{e_1}, \dots, q_{e_n})$, a straightforward structural induction on e shows that $e'\sigma$ is in $\kappa(q_{e'})$ for every sub-expression e' of e. So $e\sigma \in \kappa(q_e)$. Since $E, \kappa, \rho \models \mathsf{Ded}(q_e)$, $\kappa(q_e)$ also contains $Ded(E)$. By Lemma 13 with $E_0 \hat{=} \kappa(q_e)$ and $M \hat{=} e\sigma$, using the fact that, since E, κ, ρ satisfies $\mathsf{Ded}(q_e)$ and q_e is of type `msg`, $E_0 \supseteq Ded(E_0)$, it follows that $\kappa(q_e) \supseteq Ded(E')$, hence that E', κ, ρ satisfies $\mathsf{Ded}(q_e)$. The fact that E', κ, ρ satisfies all the other abstract semantic equations of $Eqs(e)$ follows directly from the fact that E, κ, ρ satisfied them already. ☐

Theorem 3. *Let S_1^*, …, S_n^* be n roles. Let $(R_{0i})_{i \in I}$ be a family of roles, where each R_{0i} is in $\{S_1^*, \dots, S_n^*\}$ (a copy of some S_j^*), and let $(\mathcal{W}_0, (R_{0i})_{i \in I}, (\sigma_{0i})_{i \in I}) \Rrightarrow (\mathcal{W}_1, (R_{1i})_{i \in I}, (\sigma_{1i})_{i \in I}) \Rrightarrow \dots \Rrightarrow (\mathcal{W}_p, (R_{pi})_{i \in I}, (\sigma_{pi})_{i \in I})$ be a run. Write each $\mathcal{W}_\ell$ as $(E_\ell, \mathcal{K}_\ell, \mathcal{D}_\ell)$, $0 \leq \ell \leq p$. Let ρ be an environment mapping each freshness variable X occurring in any S_j^*, $1 \leq j \leq n$, to a superset of all values created at instructions* $\texttt{new}_X\ x$ *or* $\texttt{new\_key}_X\ x$ *by any R_i in the run.*

Assume that, for each program variable x, we know a $\vee$PTA $\mathfrak{A}_x$ such that $\sigma_{0i}(x) \in [\![\mathfrak{A}_x]\!]\rho$ for every $i \in I$. (When x is an input parameter given to the roles S_1^*, …, S_n^*, $\mathfrak{A}_x$ allows us to express a constraint on the initial values that x may take; otherwise, we typically let $\mathfrak{A}$ recognized all terms that exist initially).

Let $Eqs(S_1^*, \dots, S_n^*)$ be the set of abstract semantic equations consisting of:

1. *$q_x \in \mathfrak{A}_x$, $1 \leq j \leq m$;*
2. *all equations in $Eqs(pat)$ for every command* `read` *pat in S_1^*, …, or S_n^*;*
3. *all equations in $Eqs(e)$ for every command* `write` *e in S_1^*, …, or S_n^*;*
4. *$q_x \in [X]$ for every command* $\texttt{new}_X\ x$ *or* $\texttt{new\_key}_X\ x$ *in S_1^*, …, or S_n^*;*

Let κ map each state q_x to any superset of the set of values taken by x in any $\sigma_{\ell i}$ (i.e., in any R_i at any step ℓ of the run), and assume that $E, \kappa, \rho \models Eqs(S_1^*, \dots, S_n^*)$. Then for every $1 \leq \ell \leq p$, $E_\ell \subseteq E$.

Proof. We prove the claim by induction on ℓ. The only interesting case is when the transition from step $\ell - 1$ to step ℓ of the run is by a command of the form `write` e. Then the result follows from Lemma 14. ☐

Note that we only used equations of type 2 here. The other equations are meant to force any solution to respect the initial conditions (type 1), to be able to represent any value read (type 2) or created (type 4) during any run. The fact that there is an $\vee$PTA $Ded[S_1^*, \dots, S_n^*](E_0)$ recognizing a superset of all sets E such that $E_0 \subseteq E$ (where E_0 is the initial set of messages known to the intruder) and $E, \kappa, \rho \models Eqs(S_1^*, \dots, S_n^*)$ is satisfiable is technical, and is left to another paper.

The gist of Theorem 3 is then that $Ded[S_1^*, \dots, S_n^*](E_0)$ represents a safe upper approximation of all messages that intruders may deduce with the help of an unbounded number of copies of each principal in parallel multi-session, from an initial set E_0 of messages. We can therefore verify security properties of families of principals S_1, …, S_m (in mono-session, i.e. running sequentially) and S_1^*, …, S_n^* (in multi-session) by using the algorithm of Section 5.2 on just the principals S_1, …, S_m, but replacing the computation of $Ded(\mathfrak{A} \sqcup Eval(e)\sigma^\sharp)$ in the case of `write` commands by $Ded[S_1^*, \dots, S_n^*](\mathfrak{A} \sqcup Eval(e)\sigma^\sharp)$, and conjoining to $\mathfrak{H}$ all assumptions $X \cap Y = \emptyset$ with X, Y, pairwise distinct variables of the same type, that are either freshness variables in any principal or XK_0 or XD_0.

6 Experimental Results

We have implemented the algorithms of Section 5 in a bytecode compiler for HimML, a variant of Standard ML incorporating facilities for handling finite sets and maps elegantly and efficiently [14]. We have then tested this implementation against standard cryptographic protocols [9], on a 166MHz Intel Pentium machine running Linux 2.0.30. Each of these protocols are three-party protocols, involving two principals A and B that wish to get a secret key K_{ab} by interacting with a key server S. All of these protocols were tested with an empty assumption $\mathfrak{H}$.

Results and running times are given in Figure 8. The "#Branches" column indicates how much non-determinism is involved in checking all relevant interleavings of the protocol. The (*) mark indicates that we have simulated A, B and S in mono-session, the (!) mark signals cases where we tested A and B in mono-session and S in parallel multi-session. Times are in seconds, and total the whole exploration of all relevant interleavings; in other words, our tool does not just stop after the first possible flaw.

Protocol	Result	Time (s.)	#Branches
Needham-Schroeder shared key (*)	possibly flawed	1.94	4
Needham-Schroeder shared key (!)	possibly flawed	1.56	3
Otway-Rees (*)	correct	1.56	3
Otway-Rees (!)	correct	1.56	3
Wide-Mouthed Frog (*)	possibly flawed	0.34	2
Yahalom (*)	possibly flawed	1.17	4
Yahalom (!)	possibly flawed	1.2	3
SimplerYahalom (*)	correct	1.16	3
SimplerYahalom (!)	correct	1.52	3
Otway-Rees2 (!)	correct	14.57	15

Fig. 8. Benchmark results

Cognoscenti might be reassured that the Needham-Schroeder protocol was found to be flawed, and indeed our tool finds the standard attack on this protocol where the intruder plays the second part of the session alone against B, without A or S participating at all. However, the same cognoscenti might be surprised by the fact that the Yahalom protocol was found to be flawed. Whether or not our tool has found an attack remains to be examined—indeed, reading attacks off $\vee$PTAs is not an easy task!—, but early attempts to understand what was happening reveals that this is probably due to our tool losing some information about messages. The problem is that, as noticed in [9], the Yahalom protocol is a very subtle one, where principal B checks that some late message is fresh, which allows her to conclude that an earlier message is fresh, too. The point in question is the last `read` done by B, which is:

$$\texttt{read}(\texttt{c}((\langle id_A\rangle, \texttt{k}(\texttt{symk}(K_{ab}))), \texttt{sk}(id_B, id_S)), \texttt{c}(\langle N_b\rangle, \texttt{symk}(K_{ab})))$$

(where tuples $\texttt{t}([pat_1, \ldots, pat_n])$ are abbreviated as $(pat_1, \ldots, pat_n)$ to improve readability). In this case, the key K_{ab} is assumed not to be secret by our tool, because the intruder can fool B by sending her another message $\texttt{c}((id_A, K'), \texttt{sk}(id_B, id_S))$ with an old K' instead of K_{ab}. The Yahalom protocol is however intended to allow B to reject this K' on the grounds that she won't be able to match $\texttt{c}(\langle N_b\rangle, \texttt{symk}(K_{ab}))$ with K' instead of K_{ab}. Our tool is not able to establish this, and this might indicate some fragility of the protocol. On the other hand, the tool detects no flaw in A's behavior. That is, although B might be advised not to trust the data she got, A on the other hand can be confident that she really got a secret key in the end.

Another conclusion of this experiment is that we definitely need additional tools to explain to a user why the tool estimates that a given protocol is flawed. Today, the only output we have is a sequence of commands from each principal leading to a point where some possible flaw is discovered, without any indication of which sequences of decryptions, encryptions, and so on, the intruders might have attempted.

On the other hand, the SimplerYahalom protocol (an improved version of the Yahalom protocol given in [9]) is found to be correct by our tool, confirming the opinion of op.cit. that this second version is is easier to show correct than the original one.

The last line of the table shows a simulation of two sessions of the Otway-Rees protocol in sequence. Or rather, OtwayRees2 simulates a principal A_2 playing the role of A twice in a row (with A's identity, and trying to communicate with the same B twice), a principal B_2 that plays the role of B twice in a row (with B's identity, but without checking that its peer is the same A in both sessions), and a server S in parallel multi-session. The time taken by our tool is still very reasonable, although the combinatorics of all possible interleavings starts to be daunting. We are saved by the fact that several interleavings are impossible: our tool discovers that some `read`s must block ($match^\sharp$ fails), as no intruder can produce any message of the expected form.

Sometimes we are not as lucky. For instance, launching our tool on the Needham-Schroeder public key protocol (where A and B attempt to exchange two secret data with the help of a parallel multi-session server S) takes 60.95 s. to discover flaws on 10 possible branches. Running the same simulation with S in mono-session takes even more time: 720.85 s. for 126 branches.

In fact, our simulation algorithms are not efficient in any complexity-theoretic sense: already the $match^\sharp$ algorithm is exponential in the size of the pattern, because of its behavior on binary function symbols, and therefore produces automata of size possibly exponential in the size of its input automaton—and this is done at each `read`! Nonetheless, the nice news is that verification of actual protocols is quite fast on average, while still maintaining a high level of accuracy.

7 Conclusion and Perspectives

We hope to have convinced the reader that automatic verification of cryptographic protocols was now possible, including some limited form of deduction, and allowing us to prove properties like "M is definitely secret at program point p, whatever the initial messages known to the intruder, provided that assumption $\mathfrak{H}$ is verified". Our technique is not simple, however is is natural, provides actual secrecy guarantees—and to a lesser extent freshness guarantees—, and works fast in practice.

Lots of work remain to be done. As far as algorithmics is concerned, only one major problem remains to be solved, that of modeling *sequential mono-session* principals. As should be apparent from Section 6, principals that are not in parallel multi-session are assumed to run once and then stop. But this does not address the fact that these protocols are meant to be used not once but several times in a row. Formal and partially mechanized methods have long addressed this problem [4, 5, 27], but the present automatic method, as well as D. Monniaux', cannot yet deal with this problem except in crude ways (e.g., modeling sequential multi-session by parallel multi-session). The problem here is to invent ways to deal with loops, and notably to design a widening operator [11] on $\vee$PTAs that does not lose too much useful information; current known widenings on tree automata are syntactic and tend to be too inaccurate.

But there are many other, more semantic, problems to solve. For one, modeling messages by terms assumes quite a lot, e.g., that no tuple can be confused with any ciphertext, and this has to be confronted to actual implementations and coding practices. Another problem is that, as D. Bolignano does, we model a set of honest principals A, B, S, say, and collect all the others in a bag called "intruders". However, it is all too easy to posit limitations on intruders that they may overthrow in the real world: e.g., if A inadvertently shares a secret with E, but we don't care about E because we are interested in communications between A and B, then E will be able to do things that our intruders cannot do (e.g. decrypt messages coming from A). So we have to be careful and either include E in the set of simulated, honest principals, or take weaker assumptions on intruders. Although the present work takes descriptions of cryptographic protocols that are much closer to reality than, say, [9], modeling a protocol in a faithful way is still a difficult challenge. One way out is to relate methods like ours to rigorous models like the spi-calculus (despite differences in the actual meaning of "secret", for example). This was started in [7], and needs pursuing.

8 Acknowledgments

I would like to thank Dominique Bolignano, from whom I learned much; David Monniaux, with whom I talked on several occasions and to whom I would like to pay homage; and Mourad Debbabi, who provided me an early version of [12].

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*. ACM Press, 1997.
2. M. Bellare and P. Rogaway. Provably secure session key distribution–the three party case. In *27th ACM Symposium on Theory of Computing (STOC'95)*, pages 57–66, 1995.
3. P. Bieber. A logic of communication in a hostile environment. In *3rd Computer Security Foundations Workshop*, pages 14–22. IEEE Press, 1990.
4. D. Bolignano. An approach to the formal verification of cryptographic protocols. In *3rd ACM Conference on Computer and Communication Security*, 1996.
5. D. Bolignano. Towards a mechanization of cryptographic protocol verification. In O. Grumberg, editor, *9th International Conference on Computer Aided Verification (CAV'97)*, pages 131–142. Springer LNCS 1254, 1997.
6. D. Bolignano. Towards the formal verification of electronic commerce protocols. In *10th IEEE Computer Security Foundations Workshop*, 1997.
7. V. Bonnard. *Traduction de protocoles cryptographiques en spi-calcul*. Master's thesis (stage de DEA), Université Paris VII, 1999.
8. R. E. Bryant. Graph-based algorithms for boolean functions manipulation. *IEEE Transactions on Computers*, C35(8):677–692, 1986.

9. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society*, 426(1871):233–271, 1989.
10. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on `http://www.grappa.univ-lille3.fr/tata/`, 1997.
11. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. Correct version at `http://www.dmi.ens.fr/~cousot/COUSOTpapers/JLP92.shtml`.
12. M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi. Formal automatic verification of authentication cryptographic protocols. In *1st IEEE International Conference on Formal Engineering Methods (ICFEM'97)*. IEEE, 1997.
13. T. Genet and F. Klay. Rewriting for cryptographic protocol verification (extended version). Technical report, CNET-France Telecom, 1999. Available at `http://www.loria.fr/~genet/Publications/GenetKlay-RR99.ps`.
14. J. Goubault. HimML: Standard ML with fast sets and maps. In *5th ACM SIGPLAN Workshop on ML and its Applications*, 1994.
15. International Standards Organisation. Common criteria for information technology security (CC), 1998. International Standard ISO/IS 15408 Final Committee Draft, version 2.0.
16. D. Kindred and J. M. Wing. Fast, automatic checking of security protocols. In *Second USENIX Workshop on Electronic Commerce*, pages 41–52. USENIX, 1996.
17. J. Larmouth. *Understanding OSI*, chapter 8. Available from `http://www.salford.ac.uk/iti/books/osi/osi.html`, 1994.
18. G. Lowe. An attack on the Needham-Schroeder public-key protocol. *Information Processing Letters*, 1995.
19. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS'96*, pages 147–166. Springer Verlag LNCS 1055, 1996.
20. W. Marrero, E. M. Clarke, and S. Jha. Model checking for security protocols. Technical Report CMU-SCS-97-139, Carnegie Mellon University, 1997.
21. C. A. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–36, 1992.
22. C. A. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 1995.
23. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, pages 1–40 & 41–77, 1992.
24. D. Monniaux. Abstracting cryptographic protocols with tree automata. In *6th International Static Analysis Symposium (SAS'99)*. Springer-Verlag LNCS 1694, 1999.
25. D. Monniaux. Decision procedures for the analysis of cryptographic protocols by logics of belief. In *12th Computer Security Foundations Workshop*. IEEE, 1999.
26. D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.
27. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
28. D. Prawitz. *Natural Deduction, A Proof-Theoretical Study*. Almqvist and Wiskell, Stockholm, 1965.
29. A. Rauzy and R. Génisson. Efficient Horn renaming: Yet a Davis and Putnam's procedure. Technical Report 1092-95, LaBRI, Bordeaux, France, 1995.
30. B. Schneier. *Applied Cryptography*. John Wiley and Sons, 1996.
31. S. D. Stoller. A bound on attacks on authentication protocols. Technical Report 526, Indiana University, 1999. Available from `http://www.cs.indiana.edu/hyplan/stoller.html`.