Computing Symbolic Models for Verifying Cryptographic Protocols

Marcelo Fiore*
Computer Laboratory
University of Cambridge

Martín Abadi†
Bell Labs Research
Lucent Technologies

Abstract

We consider the problem of automatically verifying infinite-state cryptographic protocols. Specifically, we present an algorithm that given a finite process describing a protocol in a hostile environment (trying to force the system into a "bad" state) computes a model of traces on which security properties can be checked. Because of unbounded inputs from the environment, even finite processes have an infinite set of traces; the main focus of our approach is the reduction of this infinite set to a finite set by a symbolic analysis of the knowledge of the environment. Our algorithm is sound (and we conjecture complete) for protocols with shared-key encryption/decryption that use arbitrary messages as keys; further it is complete in the common and important case in which the cryptographic keys are messages of bounded size.

Introduction

The problem of the automatic verification of cryptographic protocols has received great attention and by now various algorithms and tools for checking security properties are available, see *e.g.* [13, 14, 15, 12, 21, 11, 19, 18, 22, 7]. These algorithms and tools typically analyse a formal model that describes the flow of information and the interaction between principals (as specified by the protocol) in a hostile environment, that is, in the presence of attackers. In general this analysis is infinite state. Indeed, even when restricting attention to finite runs of protocols between a finite number of principals, attackers with unbounded capabilities are often considered. In particular, these attackers may send messages of any size obtained by manipulating messages that are available in the environment. (Some complexity-theoretic models simply exclude such attackers, and permit only attackers that send messages of size polynomial in some security parameter. But those models still allow attackers that may send a huge number of messages, so our considerations may apply there too.)

Symbolic techniques have generally proved successful for dealing with infinite-state systems and their application to the analysis of cryptographic protocols has obvious appeal. Recent work pursues symbolic analysis (see '§ Related work' below). In this paper we continue this line of investigation. We choose to model protocols as processes. We then reason about the properties of a protocol by considering the computation traces of a process that describes the protocol. Because of unbounded inputs from the environment, the set of computation traces is generally infinite. Hence, to perform automatic verification, a finite representation of this set is needed. We obtain such a representation by symbolic techniques in two stages. Firstly, we consider a symbolic reduction of processes in which inputs are evaluated formally. The set of symbolic runs of a process is finite; however not all such runs represent concrete traces, typically because of the symbolic evaluation of unconstrained inputs of which the environment has no knowledge. Secondly, we devise a symbolic procedure that analyses the knowledge of the environment (*i.e.*, the messages that can be deduced from previous messages that have been around the network) and use it to construct symbolic models of processes consisting of computationally meaningful symbolic runs. These finite models can then be analysed for verification purposes.

In the remainder of this introduction, we give an informal overview of our work through an example. We also comment on some related work and give an outline for this paper.

Informal overview. Following [2], we model cryptographic protocols in a dialect of the spi calculus; *viz.* a π-calculus [17] extended with cryptographic primitives. The adoption of this model is not crucial, but it is convenient. A typical process will thus consist of the parallel composition (allowing for the interaction) of processes describing the behaviour of each principal involved in the protocol; for details see [2]. For instance consider the following

*Research supported by an EPSRC Advanced Research Fellowship.

†Presently at InterTrust, Strategic Technologies and Architectural Research Laboratory.

simple example

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & \{M\}_k \\ \text{Message 2} & B \rightarrow A: & h(M) \end{array}$$

in which principal A sends a secret message M to principal B encrypted under a private shared key k and, after receiving and decrypting the message, B replies back to A with a hashed copy of the message, so that the message is not revealed and A knows that B has seen it. This protocol is represented by the process $A \mid B$ (the parallel composition of A and B) where A and B are sequential processes that represent the behaviours of the respective principals:

$$\begin{array}{lcl} A & = & !_1\langle\{M\}_k\rangle.\ ?_2(x).\ A'[x] \\ A'[x] & = & \textit{if } x = h(M) \textit{ then } \mathbf{0} \\ B & = & ?_1(x).\ B'[x] \\ B'[x] & = & \textit{case } x \textit{ of } \{y\}_k \textit{ in } B''[y] \\ B''[y] & = & !_2\langle h(y)\rangle.\ \mathbf{0} \end{array}$$

We use ? and ! to denote *input* and *output*, respectively. Informally, we have decorated these constructs with labels, 1 and 2, indicating the protocol messages being modelled. (To improve readability, these labels are not included in the formal calculus of this paper.) We use the notation $\textit{if } M = N \textit{ then } P$ (matching) and $\textit{case } M \textit{ of } \{x\}_N \textit{ in } P$ (shared-key decryption) for testing for equality of messages and for decrypting messages. (For more on the syntax of processes see Section 1.)

The computation traces of a process describing a protocol can be then employed to verify security properties of it. For example, on the computation traces of $A \mid B$ one can verify the correspondence property [25] that whenever the acknowledgement event $!_2$ occurs in a trace, the sending event $!_1$ has previously occurred. Indeed, this is clearly seen in the schema of the computation graph of $A \mid B$ given in Figure 1. (This graph roughly encodes the semantics of the process according to a reduction relation that we define in Section 1 and which formalises how processes evolve while interacting with their environments.) The nodes of the graph are pairs $K; P$ consisting of a set of messages K currently known to the environment together with the current state of the system P. As it is well-known, the information given by the knowledge of the environment is crucial:

(I) input transitions $K; P \xrightarrow{?\langle M\rangle} K; P'$ are allowed only for messages M known to the environment, and

(O) output transitions $K; P \xrightarrow{!\langle M\rangle} K'; P'$ increase the knowledge of the environment: $K' = K \cup \{M\}$.

Even in the simple example we are considering the computation graph of the process modelling the protocol is infinite; each boxed node in Figure 1 has infinite transitions, as the set of messages known to the environment is infinite. Thus, to perform automatic verification we need to capture the information of this infinite structure in a finite one. This may be achieved by symbolic techniques. The basic idea is simple: to avoid the infinitely branching structure, input transitions will take place symbolically (that is, formally). Accordingly, the symbolic computation graph of $A \mid B$ is given in Figure 2. (This graph roughly encodes the symbolic semantics of processes given in Section 1.) However, these symbolic structures are redundant and care is needed when analysing them. For instance, there is a symbolic run in Figure 2 (the leftmost one) that violates the correspondence property discussed above but that, of course, has no concrete counterpart in the computation graph of Figure 1, because since the key k is not known to the environment, there is no message N for which the message $\{N\}_k$ is known to the environment. Hence we need a procedure for identifying which symbolic runs actually correspond to concrete traces and how; for example, the symbolic run

$$\begin{array}{l} !_1\langle\{M\}_k\rangle \cdot ?_1\langle x\rangle \cdot [x \mapsto \{y\}_k] \\ \quad \cdot\, !_2\langle h(y)\rangle \cdot ?_2\langle x'\rangle \cdot [x' \mapsto h(M)] \end{array}$$

corresponds to a trace in the computation graph only under the substitution $[\,y \mapsto M\,]$. A main technical contribution of this paper (Section 2), from which such a procedure can be easily derived (Section 3), is an algorithm that given a set of messages K and a message M computes enough substitutions (that we call *realisers*) under which M is known from K.

We have seen an example of how to verify a correspondence property. We may also verify secrecy properties. To this end we may consider *observers* (not to be confused with attackers, which are implicit in the computational model) that at any time interrogate the environment as to whether it knows any secrets and if so raise an error. For instance, to check whether in our running example the message M is revealed, one considers the process $A \mid B \mid O$, where the observer O is given by the process

$$\begin{array}{lcl} O & = & ?(x).\ O'[x] \\ O'[x] & = & \textit{if } x = M \textit{ then } !\langle \textit{error}\rangle.\ \mathbf{0} \end{array}$$

and one checks if there is a trace with the output message $!\langle\textit{error}\rangle$. The computation graph of $A \mid B \mid O$ is obtained from that of $A \mid B$ (Figure 1) by extending every node $K\ ;\ P$ to $K\ ;\ P \mid O$, and adding transitions

$$K\ ;\ P \mid O \xrightarrow{?\langle N\rangle} K\ ;\ P \mid O'[N] \xrightarrow{t} K'\ ;\ P' \mid O'[N]$$
(where N is known to the environment)

for each transition $K\ ;\ P \xrightarrow{t} K'\ ;\ P'$ in the computation graph of $A \mid B$. Hence, no computation trace of $A \mid B \mid O$

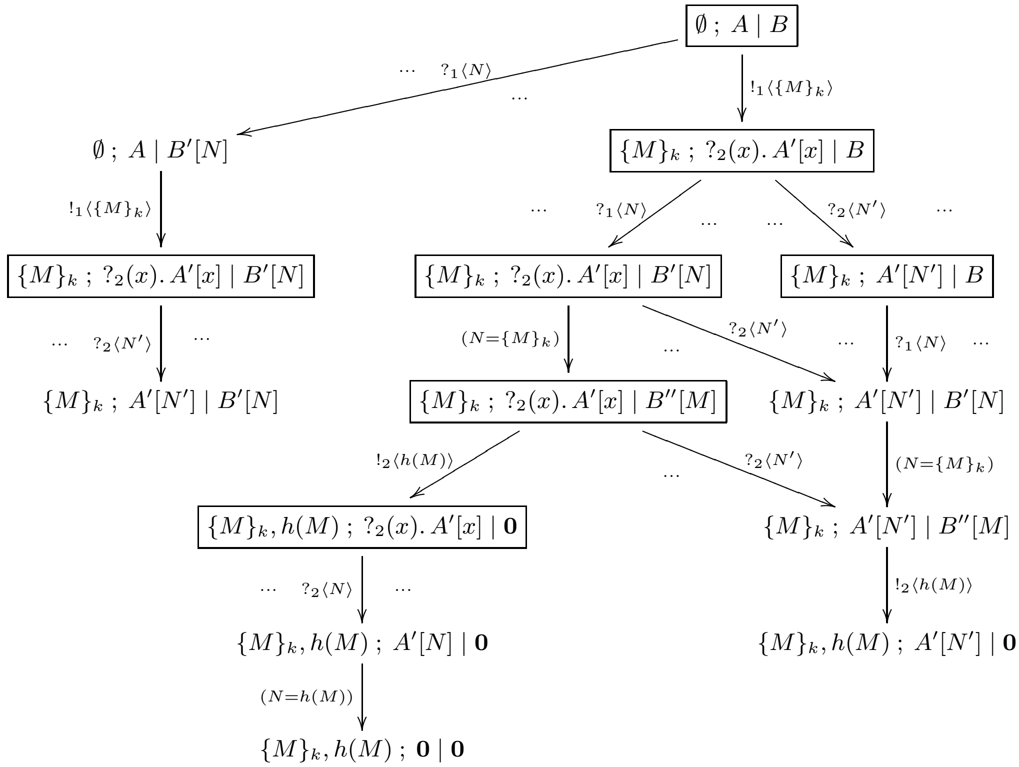

Figure 1. Computation graph of $A \mid B$

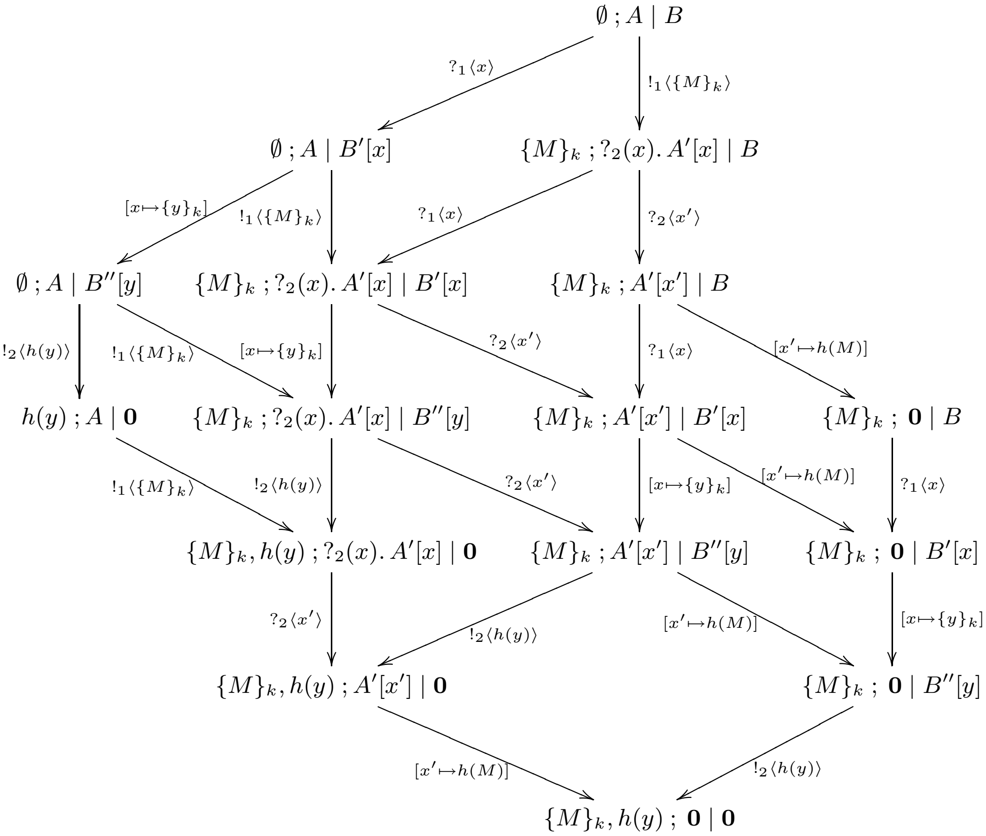

Figure 2. Symbolic computation graph of $A \mid B$

contains $!\langle error\rangle$. However, this is not the case in the symbolic computation graph of $A \mid B \mid O$ where, for instance, we will find the symbolic run

$$\begin{aligned}\emptyset\,;\,A \mid B \mid O &\xrightarrow{!_1\langle\{M\}_k\rangle} \{M\}_k\,;\,?_2(x).\,A'[x] \mid B \mid O\\ &\xrightarrow{?\langle x\rangle} \{M\}_k\,;\,?_2(x).\,A'[x] \mid B \mid O'[x]\\ &\xrightarrow{[x\mapsto M]} \{M\}_k\,;\,?_2(x).\,A'[x] \mid B \mid !\langle error\rangle.\mathbf{0}\\ &\xrightarrow{!\langle error\rangle} \{M\}_k, error\,;\,?_2(x).\,A'[x] \mid B \mid \mathbf{0}\end{aligned}$$

Thus, the above considerations when analysing symbolic runs also apply here.

Related work. The use of symbolic techniques for the analysis of cryptographic protocols is not entirely new. Even the seminal work of Dolev and Yao [8] might be seen as developing symbolic techniques, but the works most relevant to ours are recent. In particular, both Song's Athena [23] and Huima's HPA1 [10] rely on symbolic techniques. In Athena, the techniques are applied in a broad, infinite-state context, and issues such as termination are not addressed. HPA1 targets roughly the same class of systems as our work. More precisely, processes consist of a top-level parallel composition of sequential threads with state variables. These processes can send and receive messages, generate new nonces, branch according to the equality (and inequality) of messages and change state variables. One significant difference with our work is in the way operations on messages are dealt with: in HPA1, data are equipped with both constructors (like pairing, shared-key encryption *E*, *etc.*) together with corresponding destructors (like projections, shared-key decryption *D*, *etc.*) subject to equations (like $D(E(M, N), N) = M$, *etc.*); in our work, we consider only messages with constructors (again, like pairing and shared-key encryption) but equip processes with operations for destructing and using them (like pair splitting and shared-key decryption). Conceptually, the first approach amounts to working with the quotient of the term algebra under the equations, whilst in the second case one just works with the term algebra. Technically, however, a precise comparison of our results is difficult. Two other projects that are roughly contemporaneous to ours are those of Amadio *et al.* [3, 4] and Boreale [5, 6]. Both use a process calculus to model cryptographic protocols, but they treat only the case in which encryption keys are atomic. Amadio and Lugiez [3] give a symbolic method to solve reachability problems and explain how it can be used for protocol verification. In their approach (unlike in ours) there is no separation between symbolic reduction and knowledge analysis. Boreale [5] investigates an alternative approach in which symbolic runs are independent from the knowledge of the environment and are refined to meaningful traces via knowledge analysis. For convenience, we have adopted a symbolic reduction relation in the style of Boreale's, but our methods for knowledge analysis are quite different.

Organisation of the paper. Section 1 presents the syntax of the process calculus used in the paper, together with its reduction semantics and a corresponding concept of symbolic reduction. Section 2 provides an algorithmic analysis of the knowledge of the environment, giving a decision procedure for knowledge checking and a symbolic procedure for knowledge analysis. Section 3 combines the symbolic semantics of Section 1 and the symbolic procedure of Section 2, and provides symbolic models for processes. It also studies soundness and completeness issues. Finally, Section 4 offers some concluding remarks.

Acknowledgements. We have benefited from discussions with Michele Boreale.

1 The calculus and symbolic reduction

We introduce the dialect of the spi calculus [2] that we use in this paper for describing cryptographic protocols. Because of the choices and restrictions discussed below, our dialect of the spi calculus is also close to CCS [16] and CSP [9].

As it is by now common in formal analyses, our model assumes perfect encryption and hashing, and that messages have enough redundancy so that the decryption algorithm can detect whether a ciphertext was encrypted with the expected key. Further, communication is assumed to be global (*i.e.*, through the environment or global network) and the attacker, which is modelled implicitly, is allowed unbounded capabilities for manipulating any messages (by pairing and projecting, encrypting and decrypting, and hashing) available in the environment. Our analysis is restricted to finite processes, that is, to processes without replication. Therefore, it deals only with finite runs of protocols between a finite number of principals. As indicated above, the unbounded inputs of the environment result in infinite sets of runs even for such finite processes. Our focus is taming this source of infinity. (In another direction, one may attempt to apply symbolic methods to tame replication, which we exclude.)

Syntax. The grammar for messages follows. It relies on disjoint categories of variables, names, and tokens respectively ranging over recursive sets $\mathcal{V}$, $\mathcal{N}$, and $\mathcal{T}$. (The tokens represent data that the environment has initially.)

$$
\begin{array}{lll}
M, N & ::= & \text{messages} \\
& x & \text{variables} \\
& n & \text{names} \\
& \tau & \text{tokens} \\
& (M, N) & \text{pairing} \\
& \{M\}_N & \text{shared-key encryption} \\
& h(M) & \text{hashing}
\end{array}
$$

Note that we allow arbitrary messages as encryption keys; for example, this allows us to model a protocol where a key is generated by hashing a fresh shared secret. Note also that we include constructs for shared-key cryptography and hashing, but not, for example, for public-key cryptography. However, our techniques are malleable, and we expect that they can be applied to protocols that use other cryptographic primitives.

The syntax of processes is given below; it is standard except for the *input* and *output* constructs which (as communication is assumed global, unlike in [2]) do not carry channels. Note that the language lacks *restriction* and *replication*. Replication has been omitted because we are restricting our analysis to finite processes and, in its absence, restriction can be omitted without loss of generality.

$$
\begin{array}{lll}
P, Q \ ::= & & \text{processes} \\
\quad \mathbf{0} & & \text{nil} \\
\quad P \mid Q & & \text{parallel composition} \\
\quad ?(x).\, P & & \text{input} \\
\quad !\langle M \rangle . P & & \text{output} \\
\quad \mathit{if}\ M = N\ \mathit{then}\ P & & \text{matching} \\
\quad \mathit{let}\ M = (x, y)\ \mathit{in}\ P & & \text{pair splitting} \\
\quad \mathit{case}\ M\ \mathit{of}\ \{y\}_N\ \mathit{in}\ P & & \text{shared-key decryption}
\end{array}
$$

As usual, *ground* messages are those without variables. Processes with no free variables are *closed*.

Reduction semantics. We give a reduction semantics formalising the rules of interaction between processes and the environment. This semantics is thus a relation between configurations that encompass both the current knowledge of the environment and the state of the system. There are various ways to define reduction, *e.g.*, as outlined in the introduction (*cf.* [1, 3]). Here we adopt the style of [5], in which configurations are pairs consisting of sequences of actions (input or output ground messages) and processes. The former record the history of interactions between the process and the environment, whilst the latter describes the future behaviour of the system.

The reduction rules are relatively standard except for input and output, see (I) and (O) in the introduction. Both of these rules record the interaction with the environment. Further, the input rule is restricted to messages that are known to the environment. Thus, taking as the state of knowledge of the environment the set $|t|$ of messages that have been output along the sequence of interactions t, the input rule depends on an entailment relation $|t| \vdash M$ establishing whether a message M is known (or derivable) from $|t|$. The reduction rules are as follows.

$$
\begin{array}{rl}
(\textsc{In}) & t\ ;\ ?(x).\ P \longrightarrow t \cdot ?\langle M \rangle\ ;\ P[x \mapsto M] \\
& \quad \text{for } M \text{ closed and } |t| \vdash M \text{ derivable} \\[2ex]
(\textsc{Out}) & t\ ;\ !\langle M \rangle .\ P \longrightarrow t \cdot !\langle M \rangle\ ;\ P \\[2ex]
(\textsc{Split}) & t\ ;\ \mathit{let}\ (M, N) = (x, y)\ \mathit{in}\ P \\
& \qquad \longrightarrow t\ ;\ P[x \mapsto M, y \mapsto N] \\[2ex]
(\textsc{Case}) & t\ ;\ \mathit{case}\ \{M\}_N\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P \\
& \qquad \longrightarrow t\ ;\ P[x \mapsto M] \\[2ex]
(\textsc{Match}) & t\ ;\ \mathit{if}\ M = M\ \mathit{then}\ P \longrightarrow t\ ;\ P \\[2ex]
(\textsc{Par}) & \dfrac{t\ ;\ P \longrightarrow t'\ ;\ P'}{t\ ;\ P \mid Q \longrightarrow t'\ ;\ P' \mid Q} \\[3ex]
& \dfrac{t\ ;\ Q \longrightarrow t'\ ;\ Q'}{t\ ;\ P \mid Q \longrightarrow t'\ ;\ P \mid Q'}
\end{array}
$$

The entailment relation $K \vdash M$ can be presented in different ways: by rewrite systems [13, 15], by deductive systems [21, 7], by inductive definitions [20], or by axioms [24]. For our purposes, it is important that it be presented as a deductive system. The definition follows.

$$
\begin{array}{lll}
(\textsc{Tok}) & \dfrac{}{K \vdash \tau} & \\[3ex]
(\textsc{Ax}) & \dfrac{}{K \vdash M} & M \in K \setminus \mathcal{T} \\[3ex]
(\textsc{Pair}) & \dfrac{K \vdash M \qquad K \vdash N}{K \vdash (M, N)} & \\[3ex]
(\textsc{Proj}_1) & \dfrac{K \vdash (M, N)}{K \vdash M} & M \notin \mathcal{T} \\[3ex]
(\textsc{Proj}_2) & \dfrac{K \vdash (M, N)}{K \vdash N} & N \notin \mathcal{T} \\[3ex]
(\textsc{Enc}) & \dfrac{K \vdash M \qquad K \vdash N}{K \vdash \{M\}_N} & \\[3ex]
(\textsc{Dec}) & \dfrac{K \vdash \{M\}_N \qquad K \vdash N}{K \vdash M} & M \notin \mathcal{T} \\[3ex]
(\textsc{Hash}) & \dfrac{K \vdash M}{K \vdash h(M)} &
\end{array}
$$

The set of messages known to the environment is always infinite, as the entailment $K \vdash M$ holds for any K and all M built from tokens.

Definition 1.1 *(Runs and traces) A ground* run *is a sequence of input/output ground messages. A ground* trace t *is a ground run such that whenever* $t = t' \cdot ?\langle M\rangle \cdot s$*, the judgement* $|t'| \vdash M$ *is derivable.*

Note that, for P a closed process, the prefix-closed set of runs

$$\mathsf{Trace}(P) = \{\, t \mid \varepsilon\,;\, P \longrightarrow^* t\,;\, P' \,\}$$

consists of traces.

Symbolic reduction. The idea behind the symbolic semantics is to allow input transitions to take place formally. There are various ways in which this may be formalised; *e.g.*, as outlined in the introduction or as in [10, 3]. Here, matching the style of the reduction semantics, we use a symbolic reduction relation:

$$s\,;\, P \rightsquigarrow_R s'\,;\, P'$$

between configurations consisting of a sequence of input or output messages and a process, and indexed by a substitution. The rules are given below (*cf.* [5]); we write $\mathrm{mgu}\{M = N\}$ for the most general unifier of M and N.

$$\begin{array}{rl}
(\textsc{In}) & s\,;\, ?(x).\ P \rightsquigarrow_{[\,]} s \cdot ?\langle x\rangle\,;\, P \\
& \quad \text{with } x \text{ not in } s \\
(\textsc{Out}) & s\,;\, !\langle M\rangle.\, P \rightsquigarrow_{[\,]} s \cdot !\langle M\rangle\,;\, P \\
(\textsc{Split}) & s\,;\, \textit{let } M = (x,y) \textit{ in } P \rightsquigarrow_R s[R]\,;\, P[R] \\
& \quad \text{where } R = \mathrm{mgu}\{M = (x,y)\} \\
& \quad \text{with } x \text{ and } y \text{ not in } s \\
(\textsc{Case}) & s\,;\, \textit{case } M \textit{ of } \{x\}_N \textit{ in } P \rightsquigarrow_R s[R]\,;\, P[R] \\
& \quad \text{where } R = \mathrm{mgu}\{M = \{x\}_N\} \\
& \quad \text{with } x \text{ not in } s \\
(\textsc{Match}) & s\,;\, \textit{if } M = N \textit{ then } P \rightsquigarrow_R s[R]\,;\, P[R] \\
& \quad \text{where } R = \mathrm{mgu}\{M = N\} \\
(\textsc{Par}) & \dfrac{s\,;\, P \rightsquigarrow_R s'\,;\, P'}{s\,;\, P \mid Q \rightsquigarrow_R s'\,;\, P' \mid Q[R]} \\
& \dfrac{s\,;\, Q \rightsquigarrow_R s'\,;\, Q'}{s\,;\, P \mid Q \rightsquigarrow_R s'\,;\, P[R] \mid Q'}
\end{array}$$

We write $s\,;\, P \rightsquigarrow s'\,;\, P'$ whenever there is a symbolic reduction $s\,;\, P \rightsquigarrow_R s'\,;\, P'$ for some R, and let $\rightsquigarrow^*$ denote the reflexive-transitive closure of $\rightsquigarrow$.

Definition 1.2 *(Runs) A* run *is a sequence of input/output messages such that the first occurrence of a variable in a message of the sequence is in an input message.*

For a closed process P, the set

$$\mathsf{Run}(P) = \{\, s \mid \varepsilon, P \rightsquigarrow^* s, Q \,\}$$

consists of runs. Note that in general the set of symbolic runs of a process is not prefix closed; for instance,

$$\begin{aligned}
&\mathsf{Run}(\,?(x).\ !\langle M\rangle.\ \textit{if } x = N \textit{ then } \mathbf{0}\,) \\
&\quad = \{\, \varepsilon,\ ?\langle x\rangle,\ ?\langle x\rangle \cdot !\langle M\rangle,\ ?\langle N\rangle \cdot !\langle M\rangle \,\}
\end{aligned}$$

The symbolic reduction mimics the reduction semantics.

Lemma 1.3 *Let P be a closed process.*

1. *For every reduction $\varepsilon\,;\, P \longrightarrow^* t\,;\, P'$ there exists a ground substitution ρ and a symbolic reduction $\varepsilon\,;\, P \rightsquigarrow^* s\,;\, Q$ such that $s[\rho] = t$ and $Q[\rho] = P'$.*
2. *For every symbolic reduction $\varepsilon\,;\, P \rightsquigarrow^* s\,;\, Q$ and ground substitution ρ such that $s[\rho]$ is a ground trace there exists a reduction $\varepsilon\,;\, P \longrightarrow^* t\,;\, P'$ such that $t = s[\rho]$ and $P' = Q[\rho]$.*

Corollary 1.4 *(Cf. [5]) For every closed process P and ground run t, we have that $t \in \mathsf{Trace}(P)$ iff there exists a run $s \in \mathsf{Run}(P)$ and a ground substitution ρ such that $t = s[\rho]$ is a ground trace.*

2 Knowledge analysis

This section provides an algorithmic analysis of the knowledge of the environment. This is done in two steps by first devising a procedure for deciding whether a set of ground messages entails a ground message, and then generalising it to deal with arbitrary messages. We start with an example motivating the need for this development.

Consider the process

$$!\langle\{n\}_{h(n)}\rangle.\ \mathbf{0} \ \mid\ ?(x).\ \textit{case } x \textit{ of } \{y\}_n \textit{ in } !\langle y\rangle.\ \mathbf{0}$$

Its set of traces is given by the prefix closure of the following two traces

$$(1)\ \ !\langle\{n\}_{h(n)}\rangle \cdot ?\langle M\rangle \qquad (2)\ \ ?\langle N\rangle \cdot !\langle\{n\}_{h(n)}\rangle$$

where $\{n\}_{h(n)} \vdash M$ and $\vdash N$ are derivable. On the other hand, its set of symbolic runs is given by the prefix closure of

$$\begin{array}{l}
(i)\ \ !\langle\{n\}_{h(n)}\rangle \cdot ?\langle x\rangle \\
(ii)\ \ ?\langle x\rangle \cdot !\langle\{n\}_{h(n)}\rangle \\
(iii)\ \ !\langle\{n\}_{h(n)}\rangle \cdot ?\langle\{y\}_n\rangle \cdot !\langle y\rangle \\
(iv)\ \ ?\langle\{y\}_n\rangle \cdot !\langle\{n\}_{h(n)}\rangle \cdot !\langle y\rangle \\
(v)\ \ ?\langle\{y\}_n\rangle \cdot !\langle y\rangle \cdot !\langle\{n\}_{h(n)}\rangle
\end{array}$$

The runs (i) and (ii) are symbolic counterparts of the traces (1) and (2). However, the other runs are meaningless: the one in (iii) because there is no substitution R for which $\{n\}_{h(n)} \vdash \{y\}_n[R]$ is derivable and, similarly, the ones in (iv) and (v) because there is no substitution R for which $\vdash \{y\}_n[R]$ holds. Thus, a procedure for classifying symbolic runs is needed.

The crux in classifying symbolic runs lies in devising a *message derivation engine* (using the terminology of [7]) analysing the knowledge of the environment (*viz.*, the entailment relation $\vdash$). This involves both deciding the entailment relation $K \vdash M$ (Subsection 2.1) and, more generally, computing substitutions determining constraints under which symbolic entailments may be realised (Subsection 2.2). For instance, in the runs (i) and (ii), no constraints need be imposed on the indeterminate so that the runs are traces, whilst there is no way to constrain the other runs so that they become traces.

2.1 Knowledge checking

We give a procedure for deciding whether judgements $K \vdash M$ hold, for K a finite set of ground messages and M a ground message. Such a procedure is known when the encryption keys are *atomic* (*i.e.*, just names), see *e.g.* [7], and lies at the heart of the inductive approach of [20]. Here however we consider the general case where arbitrary messages can be used as cryptographic keys; this generality is useful for modelling protocols (such as SSL) in which certain keys are not atomic but rather computed by hashing together various components. To this end we identify a class of simple derivations with the following two properties.

(Completeness) Derivable judgements have at least one simple derivation.

(Computability) The set of simple derivations of a judgement is finite and effectively enumerable.

The key technical definition is as follows: a derivation is *simple* if, in every path of its derivation tree, judgements appear at most once. (An inductive definition can also be given: axioms are simple, and any other kind of derivation is simple whenever its sub-derivations are simple and the conclusion does not appear in them.)

A close inspection of simple derivations reveals that they belong to a deductive sub-system of that of Section 1. This sub-system, which we give below, has the important property that the elimination rules (PROJ) and (DEC) are bounded by the notion of active sub-term. The *active sub-terms* of a message are defined as follows:

$$\mathsf{AST}(n) = \mathsf{AST}(\tau) = \emptyset$$
$$\mathsf{AST}((M,N)) = \{\ (M,N)\ \} \cup \mathsf{AST}(M) \cup \mathsf{AST}(N)$$
$$\mathsf{AST}(\{M\}_N) = \{\ \{M\}_N\ \} \cup \mathsf{AST}(M)$$
$$\mathsf{AST}(h(M)) = \emptyset$$

Further, for a set of messages K, we let $\mathsf{AST}(K) = \bigcup_{M \in K} \mathsf{AST}(M)$.

$$(\textsc{Tok}) \quad \frac{}{K \Vdash \tau}$$

$$(\textsc{Ax}) \quad \frac{}{K \Vdash M} \quad M \in K \setminus \mathcal{T}$$

$$(\textsc{Pair}) \quad \frac{K \Vdash M \qquad K \Vdash N}{K \Vdash (M,N)}$$

$$(\textsc{Proj}_1) \quad \frac{K \Vdash (M,N)}{K \Vdash M} \quad \begin{array}{c} M \notin \mathcal{T} \\ (M,N) \in \mathsf{AST}(K) \end{array}$$

$$(\textsc{Proj}_2) \quad \frac{K \Vdash (M,N)}{K \Vdash N} \quad \begin{array}{c} N \notin \mathcal{T} \\ (M,N) \in \mathsf{AST}(K) \end{array}$$

$$(\textsc{Enc}) \quad \frac{K \Vdash M \qquad K \Vdash N}{K \Vdash \{M\}_N}$$

$$(\textsc{Dec}) \quad \frac{K \Vdash \{M\}_N \qquad K \Vdash N}{K \Vdash M} \quad \begin{array}{c} M \notin \mathcal{T} \\ \{M\}_N \in \mathsf{AST}(K) \end{array}$$

$$(\textsc{Hash}) \quad \frac{K \Vdash M}{K \Vdash h(M)}$$

We have the following result.

Theorem 2.1 *The judgement $K \vdash M$ is derivable iff there is a simple derivation of the judgement $K \Vdash M$. Further, the set of simple derivations of the judgement $K \Vdash M$ is finite and there is a procedure that given K and M enumerates them.*

Hence, the problem of determining whether or not the judgement $K \vdash M$ is derivable is decidable.

Decision procedure. In Figure 3, we give a decision procedure for the entailment $K \vdash M$ (see Theorem 2.2 below) based on searching the space of simple derivations of $K \Vdash M$. This is simply done by trying to build a derivation applying the rules in a bottom-up fashion, keeping track of the judgements visited along the search so as to guarantee that the derivation is simple. The decision procedure is given by the program Check, with sub-procedures $\mathsf{Analyse}$ and $\mathsf{Synthetise}$, all of which we present as functions in pseudocode that correspond directly to an implementation of our method.

Theorem 2.2 *1. The program* Check *terminates on all inputs.*

$$
\begin{array}{l}
\mathsf{Check}(K,M,F) = \ (M \notin F)\ \texttt{andalso}\ (\mathsf{Analyse}(K,M,F)\ \texttt{orelse}\ \mathsf{Synthetise}(K,M,F))
\end{array}
$$

$$
\begin{array}{lll}
\mathsf{Analyse}(K,M,F) = & \texttt{let} & F' = \{M\} \cup F \\
& \texttt{in} & \texttt{case}\ M\ \texttt{of} \\
& & \quad (M_1, M_2) \quad \texttt{=>} \quad \mathsf{Check}(K,M_1,F')\ \texttt{andalso}\ \mathsf{Check}(K,M_2,F') \\
& & \quad \{M_1\}_{M_2} \quad \texttt{=>} \quad \mathsf{Check}(K,M_1,F')\ \texttt{andalso}\ \mathsf{Check}(K,M_2,F') \\
& & \quad h(M') \quad \texttt{=>} \quad \mathsf{Check}(K,M',F') \\
& & \quad \_ \quad \texttt{=>} \quad \texttt{false}
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Synthetise}(K,M,F) = & \texttt{case}\ M\ \texttt{of} \\
& \quad \tau \quad \texttt{=>} \quad \texttt{true} \\
& \quad \_ \quad \texttt{=>} \quad (M \in K) \\
& \qquad\qquad \texttt{orelse} \\
& \qquad\qquad \texttt{let} \quad F' = \{M\} \cup F \\
& \qquad\qquad \texttt{in} \quad \texttt{Orelse}_{(N_1,N_2) \in \mathsf{AST}(K) \setminus F'} \left(\begin{array}{l} (N_1 = M\ \texttt{orelse}\ N_2 = M) \\ \texttt{andalso} \\ \mathsf{Synthetise}(K,(N_1,N_2),F') \end{array} \right) \\
& \qquad\qquad\qquad \texttt{orelse} \\
& \qquad\qquad\qquad \texttt{Orelse}_{\{M\}_N \in \mathsf{AST}(K) \setminus F'} \left(\begin{array}{l} \mathsf{Synthetise}(K,\{M\}_N,F') \\ \texttt{andalso} \\ \mathsf{Check}(K,N,F') \end{array} \right)
\end{array}
$$

Figure 3. Decision procedure for knowledge checking

2. *The judgement* $K \Vdash M$ *is derivable iff* $\mathsf{Check}(K,M,\emptyset) = \texttt{true}$.

2.2 Knowledge analysis

We generalise the decision procedure of Figure 3 to deal with arbitrary messages. As pointed out at the beginning of the section this is necessary in order to classify symbolic runs. To stress the point we consider a more sophisticated example. The process

$$
\begin{array}{c}
!\langle\{n\}_{h(k)}\rangle.\ !\langle\{h(n)\}_k\rangle.\ ?(x).\ \textit{let } x = (y,z) \textit{ in} \\
\textit{case } y \textit{ of } \{y'\}_k \textit{ in case } z \textit{ of } \{z'\}_{h(k)} \textit{ in} \\
\textit{if } y' = z' \textit{ then } !\langle \textit{equal} \rangle.\ \mathbf{0}
\end{array}
$$

outputs two distinct plaintexts (n and $h(n)$) encrypted under the keys $h(k)$ and k, both unknown to the environment, then inputs a pair of messages, and if these messages are cyphertexts containing the same plaintext encrypted under the keys k and $h(k)$, it finally outputs *equal*. However, as the environment only knows of the messages $\{n\}_{h(k)}$ and $\{h(n)\}_k$, the plaintexts received by the process can be only $h(n)$ and n; hence the process has no traces with the output message $!\langle \textit{equal} \rangle$.

The set of symbolic runs of the above process is given by the prefix closure of the following

$$
\begin{array}{lll}
(i) & 1. & !\langle\{n\}_{h(k)}\rangle \cdot !\langle\{h(n)\}_k\rangle \cdot ?\langle x \rangle \\
& 2. & !\langle\{n\}_{h(k)}\rangle \cdot !\langle\{h(n)\}_k\rangle \cdot ?\langle (y,z) \rangle \\
(ii) & 1. & !\langle\{n\}_{h(k)}\rangle \cdot !\langle\{h(n)\}_k\rangle \cdot ?\langle (\{y'\}_k, z) \rangle \\
& 2. & !\langle\{n\}_{h(k)}\rangle \cdot !\langle\{h(n)\}_k\rangle \cdot ?\langle (\{y'\}_k, \{z'\}_{h(k)}) \rangle \\
(iii) & & !\langle\{n\}_{h(k)}\rangle \cdot !\langle\{h(n)\}_k\rangle \cdot ?\langle (\{y'\}_k, \{y'\}_{h(k)}) \rangle \\
& & \quad \cdot\ !\langle \textit{equal} \rangle
\end{array}
$$

The symbolic runs in (i) are traces under the empty (unconstrained) substitution (as x, y, z could be any messages known to the environment). But this is not the case for the other runs. The first run in (ii) is a trace only under the constraint $[y' \mapsto h(n)]$ (as the key k is not known to the environment, and z could be any message known to the environment). The second run in (ii) is a trace only under the constraint $[y' \mapsto h(n), z' \mapsto n]$ (as both keys k and $h(k)$ are not known to the environment). Finally, there is no way to constrain y' in the run of (iii) so that it becomes a trace.

The deductive system on which we base the computation of these constraints is given in Figure 4, where we extend the notion of active subterm to arbitrary messages by setting:

$$\mathsf{AST}(x) = \emptyset$$

The judgements of the system are of the form

$$R : K \Vdash M$$

where R is a substitution, K is a set of messages, and M is a message. The intuitive reading of these judgements is that "*R may realise the entailment $K \Vdash M$*" or that "*under the constraints R, the entailment $K \Vdash M$ may hold*".

The system of Figure 4 is a symbolic version of the deductive system in the previous subsection used as the basis for the decision procedure. The best way to understand the rules is under an operational reading. For instance consider the following.

- The (TOK) and (VAR) rules respectively establish that any set of messages entails a token or an indeterminate under no constraints.
- The (AX) rule states that the realisers of the entailment $K \Vdash M$ as an axiom is the most general way in which M corresponds to messages in K.
- The realisers of the entailment of a pair according to the (PAIR) rule are given by the realisers R_1 of the entailment of the first component together with the realisers of the entailment of the second component under the constraints R_1.
- The realisers of the entailment $K \Vdash M$ according to the (DEC) rule are given by the most general way in which M appears under an encryption in the active subterms of K together with the realisers of the encrypted message and the key.

(Note that we have given *sequential*, and hence asymmetric, rules for (PAIR), (ENC), and (DEC). The reason for having done so, instead of giving more elegant symmetric rules, is that it yields a more efficient way of computing realisers—as we found out while implementing a prototype.)

Applying the deductive system to the example of this subsection we see that, for $K = \{n\}_{h(n)}, \{h(n)\}_n$, the judgements

$$[\,] : K \Vdash x \qquad [\,] : K \Vdash (y, z)$$
$$[\, y' \mapsto h(n) \,] : K \Vdash (\{y'\}_n, z)$$
$$[\, y' \mapsto h(n), z' \mapsto n \,] : K \Vdash (\{y'\}_n, \{z'\}_{h(n)})$$

are derivable, whilst there is no R for which the judgement

$$R : K \Vdash (\{y'\}_n, \{y'\}_{h(n)})$$

is derivable.

The deductive system of realisers is subtle. For instance, not all substitutions R such that $K[R] \Vdash M[R]$ holds appear as realisers $R : K \Vdash M$; *e.g.*, all tuples containing a certain message entail the message, but the only realiser of $x \Vdash M$ is $[\, x \mapsto M \,]$. Note also that the derivability of $R : K \Vdash M$ does not imply that of $K[R] \Vdash M[R]$. Indeed, the judgement $[x \mapsto n] : \{n\}_k \Vdash (x, \{x\}_k)$ is derivable but $\{n\}_k \Vdash (n, \{n\}_k)$ is not. However, for empty realisers (*i.e.*, when no constraints are needed), there is a tight correspondence. We make this precise in the proposition below, which depends on extending the entailment relations $\vdash$ and $\Vdash$ to arbitrary messages with a rule for variables:

$$(\mathrm{VAR}) \ \frac{}{K \vdash x} \qquad\qquad (\mathrm{VAR}) \ \frac{}{K \Vdash x}$$

Proposition 2.3 *The judgement $[\,] : K \Vdash M$ is derivable iff $K \Vdash M$ is derivable iff $K \vdash M$ is derivable.*

Symbolic procedure. We present in Figure 5 an algorithm for computing realisers; it can be regarded as a symbolic version of the decision procedure of Figure 3 (see Theorem 2.4 below). This procedure, Realise, essentially consisting of the sub-procedures Analyse and Synthetise, searches the space of simple derivations in the system of realisers applying the rules in a bottom-up fashion. The procedure Analyse accounts for the rules (PAIR), (ENC), and (HASH), whilst Synthetise for the others.

We compute *constraints* by iterating the computation of realisers.

$$
\begin{array}{l}
\mathsf{Constraints}(K, M) \\
\quad = \ \texttt{for}\ R \in \mathsf{Realise}(K, M, \emptyset) \\
\qquad\ \ \texttt{union}\ \ \texttt{if}\ R = [\,] \\
\qquad\qquad\quad \texttt{then}\ \{\, [\,] \,\} \\
\qquad\qquad\quad \texttt{else}\ \texttt{for}\ R' \in \mathsf{Constraints}(K[R], M[R]) \\
\qquad\qquad\qquad\quad \texttt{union}\ \{\ R@R'\ \}
\end{array}
$$

For an example, note that, for $K = \{n\}_k$ and $M = (x, \{x\}_k)$, we have that $\mathsf{Constraints}(K, M) = \emptyset$ and that $\mathsf{Realise}(K, M, \emptyset) = \{\, [\, x \mapsto n \,] \,\}$.

Theorem 2.4 *1. The program* Realise *terminates on all inputs and the judgement $R : K \vdash M$ is derivable for all $R \in \mathsf{Realise}(K, M, F)$.*

2. The program Constraints *terminates on all inputs and, for all $R \in \mathsf{Constraints}(K, M)$, the judgement $K[R] \Vdash M[R]$ is derivable.*

We thus have a procedure that computes constraints under which symbolic entailments are realised. With this procedure symbolic models of runs, and consequently of processes, can be easily computed; this we do in the next section.

$$\text{(TOK)} \quad \frac{}{[\,]: K \Vdash \tau}$$

$$\text{(VAR)} \quad \frac{}{[\,]: K \Vdash x}$$

$$\text{(AX)} \quad \frac{}{\text{mgu}\{M = M'\} : K \Vdash M} \quad M' \in K\ ,\ M \notin \mathcal{T} \cup \mathcal{V}$$

$$\text{(PAIR)} \quad \frac{R_1 : K \Vdash M \qquad R_2 : K[R_1] \Vdash N[R_1]}{R_1@R_2 : K \Vdash (M, N)}$$

$$\text{(PROJ}_1\text{)} \quad \frac{R' : K[R] \Vdash (M[R], N[R])}{R@R' : K \Vdash M} \quad \begin{array}{c} M \notin \mathcal{T} \cup \mathcal{V}\ ,\ (M', N) \in \mathsf{AST}(K) \\ R = \text{mgu}\{M = M'\} \end{array}$$

$$\text{(PROJ}_2\text{)} \quad \frac{R' : K[R] \Vdash (M[R], N[R])}{R@R' : K \Vdash N} \quad \begin{array}{c} N \notin \mathcal{T} \cup \mathcal{V}\ ,\ (M, N') \in \mathsf{AST}(K) \\ R = \text{mgu}\{N = N'\} \end{array}$$

$$\text{(ENC)} \quad \frac{R_1 : K \Vdash M \qquad R_2 : K[R_1] \Vdash N[R_1]}{R_1@R_2 : K \Vdash \{M\}_N}$$

$$\text{(DEC)} \quad \frac{R_1 : K[R] \Vdash \{M[R]\}_{N[R]} \qquad R_2 : K[R][R_1] \Vdash N[R][R_1]}{R@R_1@R_2 : K \Vdash M} \quad \begin{array}{c} M \notin \mathcal{T} \cup \mathcal{V}\ ,\ \{M'\}_N \in \mathsf{AST}(K) \\ R = \text{mgu}\{M = M'\} \end{array}$$

$$\text{(HASH)} \quad \frac{R : K \Vdash M}{R : K \Vdash h(M)}$$

Figure 4. Deductive system for knowledge analysis

(We write $R@R'$ for the concatenation of substitutions with disjoint domain of definition R and R', and use this notation to represent the composition of R followed by R'.)

---

3 Symbolic models

We give procedures for computing symbolic models of both runs and processes, and study their soundness and completeness. The general case is treated in Subsection 3.1, whilst the special case in which encryption keys are restricted to be atomic is treated in Subsection 3.2.

Symbolic traces. We introduce a symbolic counterpart of the notion of ground trace (Definition 1.1) and relate the two notions.

Definition 3.1 *(Traces) A* trace *t is a run such that whenever $t = t' \cdot ?\langle M \rangle \cdot s$, the judgement $|t'| \vdash M$ is derivable.*

(Recall that we have extended the entailment relation to arbitrary, not necessarily closed, messages by the (VAR) rule letting $K \vdash x$ be derivable for all K.)

Definition 3.2 *Let s be a run and let ρ be a ground substitution such that $s[\rho]$ is a ground run. We say that ρ* respects *s if for every variable x in s, we have that the judgement $|s_x|[\rho] \vdash x[\rho]$, where s_x denotes the biggest prefix of s not containing x, is derivable.*

Lemma 3.3 *For every trace t,*

1. *there exists a ground substitution ρ such that ρ respects t, and*
2. *for every ground substitution ρ that respects t, the ground run $t[\rho]$ is a trace.*

3.1 General case

The model $\mathsf{Model}(\varepsilon, s)$ of a run s is the set of substitutions obtained by computing the constraints that make it

$$
\begin{array}{l}
\mathsf{Realise}(K,M,F)\\
\quad = \ \texttt{if}\ M \in F\\
\qquad \texttt{then}\ \emptyset\\
\qquad \texttt{else}\ \mathsf{Analyse}(K,M,F) \cup \mathsf{Synthetise}(K,M,F)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{Analyse}(K,M,F)\\
\quad = \ \texttt{let}\ \ F' = \{M\} \cup F\\
\qquad \texttt{in}\ \ \texttt{case}\ M\ \texttt{of}\\
\qquad\qquad (M_1,M_2) \ \Rightarrow\ \texttt{for}\ R_1 \in \mathsf{Realise}(K,M_1,F')\\
\qquad\qquad\qquad\qquad \texttt{union}\ \ \texttt{for}\ R_2 \in \mathsf{Realise}(K[R_1],M_2[R_1],F'[R_1])\\
\qquad\qquad\qquad\qquad\qquad \texttt{union}\ \{\ R_1@R_2\ \}\\
\qquad\qquad \{M_1\}_{M2} \ \Rightarrow\ \texttt{for}\ R_1 \in \mathsf{Realise}(K,M_1,F')\\
\qquad\qquad\qquad\qquad \texttt{union}\ \ \texttt{for}\ R_2 \in \mathsf{Realise}(K[R_1],M_2[R_1],F'[R_1])\\
\qquad\qquad\qquad\qquad\qquad \texttt{union}\ \{\ R_1@R_2\ \}\\
\qquad\qquad h(M') \ \Rightarrow\ \mathsf{Realise}(K,M',F')\\
\qquad\qquad \_ \ \Rightarrow\ \emptyset
\end{array}
$$

$$
\begin{array}{l}
\mathsf{Synthetise}(K,M,F)\\
\quad = \ \texttt{case}\ M\ \texttt{of}\\
\qquad \tau \ \Rightarrow\ \{\ [\,]\ \}\\
\qquad x \ \Rightarrow\ \{\ [\,]\ \}\\
\qquad \_ \ \Rightarrow\ \ \{\ \mathrm{mgu}\{M = M'\} \mid M' \in K\ \}\\
\qquad\qquad \cup\\
\qquad\qquad\quad \texttt{let}\ \ F' = \{M\} \cup F\\
\qquad\qquad\quad \texttt{in}\ \ \ \texttt{for}\ (N_1,N_2) \in \mathsf{AST}(K) \setminus F'\\
\qquad\qquad\qquad\quad \texttt{union}\ \ \texttt{let}\ \ R = \mathrm{mgu}\{M = N_1\}\\
\qquad\qquad\qquad\qquad\quad \texttt{in}\ \ \texttt{for}\ R' \in \mathsf{Synthetise}(K[R],(M[R],N_2[R]),F'[R])\\
\qquad\qquad\qquad\qquad\qquad \texttt{union}\ \{\ R@R'\ \}\\
\qquad\qquad\qquad\qquad \cup\\
\qquad\qquad\qquad\qquad\quad \texttt{let}\ \ R = \mathrm{mgu}\{M = N_2\}\\
\qquad\qquad\qquad\qquad\quad \texttt{in}\ \ \texttt{for}\ R' \in \mathsf{Synthetise}(K[R],(N_1[R],M[R]),F'[R])\\
\qquad\qquad\qquad\qquad\qquad \texttt{union}\ \{\ R@R'\ \}\\
\qquad\qquad\qquad \cup\\
\qquad\qquad\qquad\quad \texttt{for}\ \{M'\}_N \in \mathsf{AST}(K) \setminus F'\\
\qquad\qquad\qquad\quad \texttt{union}\ \ \texttt{let}\ \ R = \mathrm{mgu}\{M = M'\}\\
\qquad\qquad\qquad\qquad\quad \texttt{in}\ \ \texttt{for}\ R_1 \in \mathsf{Synthetise}(K[R],\{M[R]\}_{N[R]},F'[R])\\
\qquad\qquad\qquad\qquad\qquad \texttt{union}\ \ \texttt{for}\ R_2 \in \mathsf{Realise}(K[R][R_1],N[R][R_1],F'[R][R_1])\\
\qquad\qquad\qquad\qquad\qquad\qquad \texttt{union}\ \{\ R@R_1@R_2\ \}
\end{array}
$$

Figure 5. Symbolic procedure for knowledge analysis

(The `for`-`union` operation for finite sets is analogous to the `map` function for lists: `for` $x \in S$ `union` $f(x) \ = \ \bigcup_{x \in S} f(x)$.)

into a trace (see Lemma 3.4 below). The definition is as follows.

$$
\begin{array}{l}
\mathsf{Model}(t,s) = \\
\quad \texttt{case } s \texttt{ of} \\
\quad\quad \varepsilon \quad\quad\quad \texttt{=>} \ \{ \, [\,] \, \} \\
\quad\quad !\langle M\rangle \cdot s' \ \texttt{=>} \ \mathsf{Model}(t \cdot !\langle M\rangle, s') \\
\quad\quad ?\langle M\rangle \cdot s' \ \texttt{=>} \ \texttt{for } R \in \mathsf{Constraints}(|t|, M) \\
\quad\quad\quad\quad\quad\quad\quad \texttt{union} \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{if } R = [\,] \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{then} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathsf{Model}(t \cdot ?\langle M\rangle, s') \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{else} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \texttt{for } R' \in \mathsf{Model}(\varepsilon, (t\cdot s)[R]) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \texttt{union } \{ \, R@R' \, \}
\end{array}
$$

Lemma 3.4 *For every trace t and run s, the program* $\mathsf{Model}(t,s)$ *terminates and, for every $R \in \mathsf{Model}(t,s)$, the run $(t\cdot s)[R]$ is a trace.*

Symbolic model. The symbolic model of a process is the set of traces obtained from the models of all the symbolic runs of the process. Formally, the symbolic model of a closed process P is given by the set

$$\mathsf{Model}(P) = \{ \, s[R] \mid s \in \mathsf{Run}(P), R \in \mathsf{Model}(\varepsilon, s) \, \}$$

All the possible runs of a process are being considered in its model. How to cut down this space is worth investigating. Note, however, that one cannot just restrict attention to maximal runs. For instance, the process $!\langle h(n)\rangle.\ ?(x).\ \mathit{if}\ x = n\ \mathit{then}\ !\langle \mathit{error}\rangle.\ \mathbf{0}$ with model given by the prefix closure of the trace $!\langle h(n)\rangle \cdot ?\langle x\rangle$ has just one maximal run, *viz.* $!\langle h(n)\rangle\cdot ?\langle n\rangle\cdot !\langle \mathit{error}\rangle$, with empty model.

Soundness. It follows from Lemmas 3.4, 3.3, and 1.3 that symbolic models are sound in the sense that every trace in the model of a process represents some ground trace of it. In technical terms, we have the following result.

Corollary 3.5 (Soundness) *For every closed process P and trace $t \in \mathsf{Model}(P)$ there exists a ground substitution ρ such that $t[\rho] \in \mathsf{Trace}(P)$.*

Symbolic models of processes can be analysed for verification purposes. For instance, it follows from our development that $\mathsf{Model}(\ P \mid O\ )$, where the observer $O = ?(x).\ \mathit{if}\ x = M\ \mathit{then}\ !\langle \mathit{error}\rangle.\ \mathbf{0}$, can be computed and that if it contains a trace with the output message $!\langle \mathit{error}\rangle$ then the protocol being modelled by the process P reveals the message M. Analogously, to verify a correspondence property of a process P, one computes $\mathsf{Model}(P)$ and checks whether the property holds in all traces. By the above soundness result, if a trace violating the property is found then the protocol being modelled does not satisfy the property. In both cases, the substitutions in $\mathsf{Model}(\varepsilon, s)$ for the "bad" runs s provide attacks.

We conjecture that our symbolic models are also complete in the sense that every ground trace in $\mathsf{Trace}(P)$ is an instance of some symbolic trace in $\mathsf{Model}(P)$. If this were the case, then the absence of "bad" traces in a symbolic model would imply the satisfaction of the property that they try to refute. We have been able to prove completeness under the extra assumption that encryption keys be of bounded size. Below we present the atomic case; *i.e.*, when encryption keys are just names. This atomic case is the main focus of most formal methods for protocol analysis.

3.2 Atomic case

Symbolic model. We consider runs of messages in which encryption is atomic; their grammar is as follows.

$M, N \ ::=$		messages with atomic encryption
	x	variables
	n	names
	τ	tokens
	(M, N)	pairing
	$\{M\}_n$	atomic shared-key encryption
	$h(M)$	hashing

Under this restriction, the symbolic model of a process is given with respect to the following modified set of runs

$$\mathsf{Run}(P) = \left\{ \ s[\rho] \ \middle| \ \begin{array}{l} \varepsilon \ ; \ P \leadsto^* \ s \ ; \ Q \ , \\ \rho : \mathsf{KVars}(s) \to \mathsf{Names}(P) \ , \\ s[\rho] \text{ has messages with} \\ \text{atomic encryption} \end{array} \right\}$$

where $\mathsf{Names}(P)$ is the set of names in P and where the "*key variables*" of a run are the variables used as keys in its messages, which we formally define by:

$$
\begin{array}{l}
\mathsf{KVars}(x) = \mathsf{KVars}(n) = \mathsf{KVars}(\tau) = \emptyset \\
\mathsf{KVars}((M,N)) = \mathsf{KVars}(M) \cup \mathsf{KVars}(N) \\
\mathsf{KVars}(\{M\}_N) = \mathsf{KVars}(M) \cup \{ \, N \mid N \in \mathcal{V} \, \} \\
\mathsf{KVars}(h(M)) = \mathsf{KVars}(M)
\end{array}
$$

The bounds on keys allow for the decomposition of a set of messages into atomic parts with the same information (see Proposition 3.6 below). In the atomic case, these *atoms* are defined as follows.

$$
\begin{array}{l}
\mathsf{Atoms}(\{ \, x \, \} \uplus K) = \mathsf{Atoms}(K) \\
\mathsf{Atoms}(\{ \, \tau \, \} \uplus K) = \mathsf{Atoms}(K) \\
\mathsf{Atoms}(\{ \, (M,N) \, \} \uplus K) = \mathsf{Atoms}(\{ \, M, N \, \} \cup K) \\
\mathsf{Atoms}(\{ \, \{M\}_n, n \, \} \uplus K) = \mathsf{Atoms}(\{ \, M, n \, \} \cup K)
\end{array}
$$

Below all our considerations are for messages with atomic encryption and for set of messages, runs, and traces consisting only of those.

Proposition 3.6 *The judgement $K \vdash M$ is derivable iff* $\mathsf{Atoms}(K) \vdash M$ *is derivable.*

Because of this correspondence we refine the definition of model of a run by computing constraints with respect to the atoms (rather than the messages) known to the environment.

```
Model(t, s) =
 case s of
  ε          => { [ ] }
  !⟨M⟩ · s'  => Model(t · !⟨M⟩, s')
  ?⟨M⟩ · s'  => for R ∈ Constraints(Atoms|t|, M)
                union
                   if R = [ ]
                   then
                     Model(t · ?⟨M⟩, s')
                   else
                     for R' ∈ Model(ε, (t · s)[R])
                     union { R@R' }
```

Lemma 3.7 *For every trace t and run s, the program* $\mathsf{Model}(t, s)$ *terminates and, for every $R \in \mathsf{Model}(t, s)$, the run $(t \cdot s)[R]$ is a trace.*

Completeness. A property of the atoms of a set of messages that is crucial for establishing completeness is that they are invariant under substitution.

Lemma 3.8 *For every trace t and ground substitution ρ such that $t[\rho]$ is a ground trace we have that ρ respects t and* $\mathsf{Atoms}(|t|[\rho]) = (\mathsf{Atoms}|t|)[\rho]$.

Finally, we give a soundness and completeness result for runs.

Theorem 3.9 *Let t be a trace and s be a run.*

1. **(Soundness)** *For every $R \in \mathsf{Model}(t, s)$ the trace $(t \cdot s)[R]$ is respectable and, for every ground substitution ρ that respects $(t \cdot s)[R]$ we have that the run $(t \cdot s)[R][\rho]$ is a trace.*

2. **(Completeness)** *For every ground substitution ρ such that ρ respects t and $(t \cdot s)[\rho]$ is a trace, there exists a substitution $R \in \mathsf{Model}(t, s)$ such that ρ is an instance of R.*

4 Concluding remarks

This paper develops symbolic techniques for studying the traces of a class of infinite-state cryptographic protocols. In these protocols, the behaviour of a protocol participant is described by a finite process, while the behaviour of attackers need not be. The symbolic techniques lead to finite representations of models embodying the interactions between the protocol participants and the attackers. These symbolic models can be analysed for verification purposes.

There are obvious directions for further work. From a theoretical perspective, it would be important to settle the conjecture of whether in the general case our symbolic models are complete. From a practical viewpoint, we have implemented a prototype with our algorithms which we used to develop and test the method, but it may be interesting to build a complete verification tool and experiment with it. In this respect, our approach to knowledge analysis seems to be general enough to cope with other cryptographic primitives; perhaps a general theory of knowledge analysis should be considered. Finally, the symbolic analysis of infinite processes (with replication and restriction) should be investigated.

References

[1] M. Abadi. Security protocols and their properties. In *Foundations of Secure Computation (20^{th} International Summer School on Foundations of Secure Computation, Marktoberdorf, Germany (1999))*, NATO Science Series, pages 39–60. IOS Press, 2000.

[2] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[3] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. CONCUR 2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394. Springer-Verlag, 2000.

[4] R. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. Research Report 4147, INRIA, 2001.

[5] M. Boreale. Symbolic analysis of cryptographic protocols in the spi-calculus. Manuscript, 2000.

[6] M. Boreale. Symbolic trace analysis of cryptographic protocols. To appear in *Proc. ICALP*, 2001.

[7] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

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

[9] C. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[10] A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. Formal methods and security protocols, FLOC Workshop*, 1999.

[11] D. Kindred and J. Wing. Fast, automatic checking of security protocols. In *Second USENIX Workshop on Electronic Commerce*, 1996.

[12] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[13] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology — ASIACRYPT'94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–150. Springer-Verlag, 1994.

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

[15] J. Millen, S. Clark, and S. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, 1987.

[16] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[17] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, 1992.

[18] J. Mitchell. Finite-state analysis of security protocols. In *Proc. CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 71–76. Springer-Verlag, 1998.

[19] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proc. Symposium on Research in Security and Privacy*, pages 141–151. IEEE Computer Society, 1997.

[20] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[21] S. Schneider. Security properties and CSP. In *Proc. Symposium on Research in Security and Privacy*, pages 174–187. IEEE Computer Society, 1996.

[22] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proc. CSFW'98*. IEEE Computer Society, 1998.

[23] D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proc. CSFW'99*. IEEE Computer Society, 1999.

[24] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. CADE-16*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 378–382. Springer-Verlag, 1999.

[25] T. Woo and S. Lam. A semantic model for authentication protocols. In *Proc. Symposium on Research in Security and Privacy*. IEEE Computer Society, 1993.