On the Symbolic Analysis of Low-Level Cryptographic Primitives: Modular Exponentiation and the Diffie-Hellman Protocol *

Michele Boreale
Dipartimento di Sistemi e Informatica
Università di Firenze
Via Lombroso 6/17, 50137 Firenze, Italy
boreale@dsi.unifi.it

Maria Grazia Buscemi
Dipartimento di Informatica
Università di Pisa
Via F. Buonarroti 2, 56100 Pisa, Italy
buscemi@di.unipi.it

Abstract

Automatic methods developed so far for analysis of security protocols only model a limited set of cryptographic primitives (often, only encryption and concatenation) and abstract from low-level features of cryptographic algorithms. This paper is an attempt towards closing this gap. We propose a symbolic technique and a decision method for analysis of protocols based on modular exponentiation, such as Diffie-Hellman key exchange. We introduce a protocol description language along with its semantics. Then, we propose a notion of symbolic execution and, based on it, a verification method. We prove that the method is sound and complete with respect to the language semantics.

1 Introduction

During the last decade, a lot of research effort has been directed towards automatic analysis of crypto-protocols. Tools based on finite-state methods ([15, 17, 21]) take advantage of a well established model-checking technology, and are very effective at finding bugs. Infinite-state approaches, based on a variety of symbolic techniques ([3, 5, 9, 13, 19]), have emerged over the past few years. Implementations of these techniques (e.g. [24, 6, 23]) are still at an early stage. However, symbolic methods seem to be very promising in two respects. First, at least when the number of sessions is bounded, they can accomplish a complete exploration of the protocol's state space: thus they provide *proofs or disproofs* of correctness - under Dolev-Yao-like [12] assumptions - even though the protocol's state space is infinite. Second, symbolic methods usually rely on representations of data which help to control very well state-explosion induced by communications.

The application of automatic methods has mostly been confined to protocols built around 'black-box' enciphering and hashing functions. In this paper, we take a step towards broadening the scope of symbolic techniques, so as to include a class of low-level cryptographic operations. In particular, building on the general framework proposed in [7], we devise a complete analysis method for protocols that depend on modular exponentiation operations, like the Diffie-Hellman key-exchange [11]. We expect that our methodology may be adapted to other low-level primitives (like RSA encryption).

The Diffie-Hellman protocol is intended for exchange of a secret key over an insecure medium, without prior sharing of any secret. The protocol has two public parameters: a large prime p and a generator α for the multiplicative group $\mathcal{Z}_p^* = \{1, \ldots, p-1\}$. Assuming A and B want to establish a shared secret key, they proceed as follows. First, A generates a random private value $n_A \in \mathcal{Z}_p^*$ and B generates a random private value $n_B \in \mathcal{Z}_p^*$. Next, A and B exchange their public values ($\mathsf{exp}\,(x, y)$ denotes $x^y \bmod p$):

$$\begin{array}{ll} 1. & A \longrightarrow B: \quad \mathsf{exp}\,(\alpha, n_A) \\ 2. & B \longrightarrow A: \quad \mathsf{exp}\,(\alpha, n_B). \end{array}$$

Finally, A computes the key as $K = \mathsf{exp}\,(\mathsf{exp}\,(\alpha, n_B), n_A) = \mathsf{exp}\,(\alpha, n_A \times n_B)$, and B computes the key as $K = \mathsf{exp}\,(\mathsf{exp}\,(\alpha, n_A), n_B) = \mathsf{exp}\,(\alpha, n_A \times n_B)$. Now A and B share the value K, and A can use it to, say, encrypt a secret datum d and

*This work has been partially supported by EU within the FET - Global Computing initiative, projects MIKADO and PROFUNDIS and by MIUR project NAPOLI.

send it to B:

$$3. \quad A \longrightarrow B : \quad \{d\}_K.$$

The protocol's security depends on the difficulty of the discrete logarithm problem: it is computationally infeasible to compute y if only x and $\mathsf{exp}\,(x, y)$ are known.

When defining a model for low-level protocols of this sort, one is faced with two conflicting requirements. On one hand, one should be accurate in accounting for the operations involved in the protocol (exponentiation, product) and their 'relevant' algebraic laws; even operations that are not explicitly mentioned in protocols, but that are considered feasible (like taking the k^{th} root modulo a prime, and division) must be accounted for, because an adversary could in principle take advantage of them.

On the other hand, one must be careful in keeping the model effectively analysable. In this respect, recent undecidability results on related problems of equational unification [14] indicate that some degree of abstraction is unavoidable. The limitations of our model are discussed in Section 2. Technically, we simplify the model by avoiding explicit commutativity laws and by keeping a free algebra model and ordinary unification. In fact, we 'promote' commutativity to nondeterminism. As an example, upon evaluation of the expression $\mathsf{exp}\,(\mathsf{exp}\,(\alpha, n), m)$, an attacker will nondeterministically produce $\mathsf{exp}\,(\alpha, m \times n)$ or $\mathsf{exp}\,(\alpha, n \times m)$. The intuition is that if there is some action that depends on these two terms being equal modulo $\times$-commutativity, then there is an execution trace of the protocol where this action will take place. This seems reasonable in view of the fact that *we only consider safety properties* (i.e., 'no bad action ever takes place').

Here is a more precise description of our work. In Section 2, parallelling [7], we introduce a syntax for expressions (including $\mathsf{exp}\,(\cdot, \cdot)$ and related operations), along with a notion of evaluation. Based on this, we present a small protocol description language akin to the applied pi [1] and its (concrete) semantics. The latter assumes a Dolev-Yao adversary and is therefore infinitary. In Section 3, we introduce a finitary symbolic semantics, which relies on a form of narrowing strategy, and discuss its relationship with the concrete semantics. A verification method based on the symbolic semantics is presented in Section 4: the main result is Theorem 4.3, which asserts the correctness and completeness of the method with respect to the concrete model. It is remarkable that the presence of the modular root operation plays a crucial role in the completeness proof. In Section 5, we illustrate how the method discovers the well known man-in-the-middle attack on the Diffie-Hellman protocol. A discussion on further research is in Section 6. A separate Appendix contains some most technical details. Due to space limitations, proofs are sketched or omitted but will appear in a forthcoming full version of the paper.

Related Work Very recent work by Millen and Shmatikov [20] shows how to reduce the symbolic analysis problem in the presence of modular exponentiation and multiplication plus encryption to the solution of quadratic Diophantine equations; decidability, however, remains an open issue. Pereira and Quisquater first [22] proposed a technique for analysing group Diffie-Hellman protocols in the presence of an attacker with restricted capabilities (e.g. no symmetric encryption), though not facing the issue of decidability. Blanchet's model [4] abstracts away from operations like inverse, root extraction and random number generation. The resulting method may give rise to false attacks and may not terminate. Closely related to our problem is also protocol analysis in the presence of the xor operation, which has been recently proven to be decidable by Chevalier *et al.* [8] and, independently, by Comon-Lundh and Shmatikov [10].

2 The model

We recall the concept of *frame* from [7], and tailor it to the case of modular exponentiation and multiplication.

Expressions and messages We consider two countable disjoint sets of *names* $m, n, \ldots \in \mathcal{N}$ and *variables* $x, y, \ldots \in \mathcal{V}$. The set $\mathcal{N}$ is in turn partitioned into a countable set of *local names* $a, b, \ldots \in \mathcal{LN}$ and a countable set of *environmental names* $\underline{a}, \underline{b}, \ldots \in \mathcal{EN}$: these two sets represent infinite supplies of fresh basic values (keys, random numbers,...) at disposal of processes and of the (hostile) environment, respectively. It is technically convenient also to consider a set of *marked variables* $\hat{x}, \hat{y}, \hat{z}, \ldots \in \widehat{\mathcal{V}}$, which will be used as placeholders for generic messages known to the environment. The set $\mathcal{N} \cup \mathcal{V} \cup \widehat{\mathcal{V}}$ is ranged over by $u, v, \ldots$. Given a signature Σ of function symbols $f, g, \ldots$, each coming with its arity (constants have arity 0), we denote by $\mathcal{E}_\Sigma$ the algebra of terms (or *expressions*) on $\mathcal{N} \cup \mathcal{V} \cup \Sigma$, given by the grammar:

$$\zeta, \eta \quad ::= \quad u \;\mid\; f(\widetilde{\zeta})$$

where $\widetilde{\zeta}$ is a tuple of terms of the expected length. A *term context* $C[\cdot]$ is a term with a hole that can be filled with any expression ζ, thus yielding an expression $C[\zeta]$.

Definition 2.1 (a frame for exponentiation). *A* frame $\mathcal{F}$ *is a triple* $(\Sigma, \mathcal{M}, \downarrow)$*, where:*

- Σ *is a signature;*
- $\mathcal{M} \subseteq \mathcal{E}_\Sigma$ *is a set of* messages $M, N, \ldots$*;*
- $\downarrow \subseteq \mathcal{E}_\Sigma \times \mathcal{E}_\Sigma$ *is an* evaluation relation.

A frame for exponentiation and shared key cryptography, $\mathcal{F}_{DH} = (\Sigma, \mathcal{M}, \downarrow)$*, is presented in Table 1. We write* $\zeta \downarrow \eta$ *for* $(\zeta, \eta) \in \downarrow$ *and say that* ζ evaluates to η*.*

Besides shared-key encryption $\{\zeta\}_\eta$ and decryption $\mathsf{dec}_\eta(\zeta)$ (with η used as the key), the other symbols of Σ represent arithmetic operations modulo a fixed and public prime number, which is kept implicit. In particular, we have exponentiation $\mathsf{exp}\,(\zeta, \eta)$, root extraction $\mathsf{root}\,(\zeta, \eta)$, a constant α that represents a public generator and two constants for multiplicative unit (unit, 1), two distinct symbols for the product $\mathsf{mult}(\zeta, \eta)$ and its result $\zeta \times \eta$, three symbols, $\mathsf{inv}(\zeta)$, $\mathsf{inv}'(\zeta)$ and ζ^{-1}, representing the multiplicative inverse operation. The reason for using different symbols for the same operation is discussed below. All the underlying operations are computationally feasible[1]. A message is either a product of up to l values, for a fixed l, or a key or a message encrypted under a key. A key can be an atomic object, or an exponential with basis α and a product as exponent ($\mathsf{exp}\,(\alpha, F)$).

Evaluation ($\downarrow$) is the reflexive and transitive closure of an auxiliary relation $\rightsquigarrow$, as presented in Table 1. There, we use $\zeta_1 \times \zeta_2 \times \cdots \times \zeta_n$ as a shorthand for $\zeta_1 \times (\zeta_2 \times \cdots \times \zeta_n)$, while $(i_1, \ldots, i_n)$ is any permutation of $(1, \ldots, n)$. The relation $\rightsquigarrow$ is terminating, but not confluent. In fact, the non-determinism of $\rightsquigarrow$ is intended to model the commutativity and the associativity of the product operation, as reflected in the rule (MULT). Also note rule (ROOT): in modular arithmetic, taking the k^{th} root amounts to raising to the $(k^{-1} \bmod p-1)^{th}$ power. The adoption of distinct symbols for product (mult and $\times$), inverse (inv, inv' and $()^{-1}$), and unit (unit and 1), and the form of the rules, ensure termination of both $\rightsquigarrow$ and of the induced narrowing relation, to be introduced in the next section.

The choice of the above message and rule formats corresponds to imposing the following restrictions on the attacker and on the honest participants:

1. there is a fixed upper bound (l) on the number of factors;
2. product and inverse operations cannot be applied to exponentials and to encrypted terms;
3. exponentiation starts from the basis α, and exponents can only be products.

More accurately, starting from a term obeying the above conditions, an attacker is capable of 'deducing' all - though not necessarily only - AC variants of the message represented by the term, in a sense made precise below. Terms not obeying these restrictions are just not guaranteed to produce any message. Restriction (1) might be relaxed at the cost of introducing a class of mult_l operations, for each $l \geq 0$, but for simplicity we shall stick to the above model in this paper.

Below, we define a deduction relation which expresses how the environment can generate new messages starting from an initial set of messages S. Note that environmental names and marked variables are treated as terms known to the environment. We denote by $\mathcal{P}_f(X)$ the set of finite subsets of X.

Definition 2.2 (deduction relation). *For* $S \subseteq \mathcal{M}$*, the set* $\mathcal{H}(S)$ *is inductively defined by the following rules:*

$$\begin{array}{rcl} \mathcal{H}^0(S) & = & S \cup \mathcal{EN} \cup \widehat{\mathcal{V}} \\ \mathcal{H}^{i+1}(S) & = & \mathcal{H}^i(S) \cup \{ f(\widetilde{\zeta}) : \ f \in \Sigma, \ \widetilde{\zeta} \subseteq \mathcal{H}^i(S) \} \\ \mathcal{H}(S) & = & \bigcup_{i \geq 0} \mathcal{H}^i(S). \end{array}$$

The deduction relation $\vdash \subseteq \mathcal{P}_f(\mathcal{M}) \times \mathcal{M}$ *is defined by:*

$$S \vdash M \quad \textit{iff} \quad \exists\, \zeta \in \mathcal{H}(S) \,:\, \zeta \downarrow M.$$

Example 2.3. *Consider a set* $S = \{\, n_A, \mathsf{exp}\,(\alpha, n_B) \,\}$*. Then,* $S \vdash \mathsf{exp}\,(\alpha, n_A \times n_B)$ *and* $S \vdash \mathsf{exp}\,(\alpha, n_B \times n_A)$*. A further example involves the use of the root extraction. Consider a set* $S = \{\, \{m\}_{\mathsf{exp}\,(\alpha, k \times l)},\ \mathsf{exp}\,(\alpha, k \times h),\ h,\ l \,\}$*. Then,* $S \vdash m$ *since there exists* $\zeta \in \mathcal{H}(S)$*,* $\zeta = \mathsf{dec}_\eta(\{m\}_{\mathsf{exp}\,(\alpha, k \times l)})$*, with* $\eta = \mathsf{exp}\,(\mathsf{root}\,(\mathsf{exp}\,(\alpha, k \times h), h), l)$*, s.t.* $\zeta \downarrow m$*.*

Processes We consider a variant of the applied pi [1]. The syntax of *agents* is presented in Table 2.

Terms ζ and η range over $\mathcal{E}_\Sigma$. We consider a set $\mathcal{L}$ of *labels* which is ranged over by $\mathsf{a}, \mathsf{b}, \ldots$. A unique public channel is assumed, thus input and output labels ($\mathsf{a}, \mathsf{b}, \ldots$) are simply 'tags' attached to process actions for ease of reference. The only construct (let) for evaluating expressions replaces the ad-hoc constructs found in the spi-calculus [2] for encryption, decryption and other cryptographic operations. The construct

[1] Note that the inverse of k modulo n is defined only if $\gcd(k, n) = 1$ (see [18]). Our model abstracts away from this condition. Another abstraction we make is that just one operation is used to model both inverse mod p and inverse mod $p-1$ (the latter operation arises only inside exponents).

$$
\begin{array}{rl}
\textsc{Signature} & \Sigma \;=\; \{\, \alpha,\ \mathsf{unit},\ 1,\ \{\cdot\}_{(\cdot)},\ \mathsf{dec}_{(\cdot)}(\cdot),\ \mathsf{exp}\,(\cdot,\cdot),\ \mathsf{root}\,(\cdot,\cdot),\\
& \qquad \cdot \times \cdot,\ \mathsf{mult}(\cdot,\cdot),\ \mathsf{inv}(\cdot),\ \mathsf{inv}'(\cdot),\ (\cdot)^{-1}\,\} \\
\textsc{Factors} & f ::= u \;\mid\; u^{-1} \\
\textsc{Products} & F ::= 1 \;\mid\; f_1 \times \cdots \times f_k \\
\textsc{Keys} & K, H ::= f \;\mid\; \mathsf{exp}\,(\alpha, F) \\
\textsc{Messages} & M, N ::= F \;\mid\; K \;\mid\; \{M\}_K
\end{array}
$$

$$
\begin{array}{rl}
(\textsc{Dec}) & \mathsf{dec}_\eta(\{\zeta\}_\eta) \rightsquigarrow \zeta \\
(\textsc{Mult}) & \mathsf{mult}(\zeta_1 \times \cdots \zeta_k, \zeta_{k+1} \times \cdots \times \zeta_n) \rightsquigarrow \zeta_{i_1} \times \cdots \times \zeta_{i_n} \qquad 1 \le k < n \le l \\
(\textsc{Inv}_1) & \mathsf{inv}(\zeta_1 \times \cdots \times \zeta_n) \rightsquigarrow \mathsf{inv}'(\zeta_1) \times \cdots \times \mathsf{inv}'(\zeta_n) \qquad n \le l \\
(\textsc{Inv}_2) & \mathsf{inv}'(\zeta^{-1}) \rightsquigarrow \zeta \quad (\textsc{Inv}_3) \quad \mathsf{inv}'(\zeta) \rightsquigarrow \zeta^{-1} \quad (\textsc{Inv}_4) \quad \mathsf{inv}'(\zeta) \times \zeta \rightsquigarrow \mathsf{unit} \\
(\textsc{Unit}_1) & \mathsf{unit} \times \zeta \rightsquigarrow \zeta \quad (\textsc{Unit}_2) \quad \mathsf{unit} \rightsquigarrow 1 \\
(\textsc{Exp}) & \mathsf{exp}\,(\mathsf{exp}\,(\xi, \eta), \zeta) \rightsquigarrow \mathsf{exp}\,(\xi, \mathsf{mult}(\eta, \zeta)) \\
(\textsc{Root}) & \mathsf{root}\,(\mathsf{exp}\,(\xi, \eta), \zeta) \rightsquigarrow \mathsf{exp}\,(\xi, \mathsf{mult}(\eta, \mathsf{inv}(\zeta))) \\
(\textsc{Ctx}) & \dfrac{\zeta \rightsquigarrow \zeta'}{C[\zeta] \rightsquigarrow C[\zeta']} \qquad \textsc{Evaluation} \quad \zeta \downarrow \eta \quad \text{iff} \quad \zeta \rightsquigarrow^* \eta
\end{array}
$$

Table 1: $\mathcal{F}_{DH}$, a frame for modular exponentiation

$$
\begin{array}{rll}
A, B & ::= & \\
& \mathbf{0} & \text{(null)} \\
\mid & \mathsf{a}(x).\, A & \text{(input)} \\
\mid & \overline{\mathsf{a}}\langle\zeta\rangle.\, A & \text{(output)} \\
\mid & \mathsf{let}\, x = \zeta\, \mathsf{in}\, A & \text{(evaluation)} \\
\mid & [\zeta = \eta]A & \text{(matching)} \\
\mid & A \parallel B & \text{(parallel composition)} \\
\mid & (\mathsf{new}\, a)\, A & \text{(restriction)}
\end{array}
$$

Table 2: Syntax of agents

$(\mathsf{new}\, a)\, A$ denotes the creation of a new name a, which is private to A. Note that the occurrences of variable x are bound in input and let operators.

Given the presence of binders for variables, notions of *free variables*, $\mathrm{v}(A) \subseteq \mathcal{V}$, *substitution* $[\zeta/u]$ for any ζ and u, and *alpha-equivalence* arise as expected. We shall identify alpha-equivalent agents. We denote by $\mathrm{en}(A)$ the set of environmental names occurring in A. An agent A is said to be *closed* or a *process* if $\mathrm{v}(A) = \emptyset$; the set of processes $\mathcal{P}$ is ranged over by $P, Q, \ldots$.

Example 2.4 (the Diffie-Hellman key exchange). *The process P below is a formalisation of the Diffie-Hellman protocol presented in the introduction. For simplicity, we just describe a one-session version of the protocol.*

$$
\begin{array}{rcl}
A & = & (\mathsf{new}\, n_A)\, \overline{\mathsf{a1}}\langle \mathsf{exp}\,(\alpha, n_A)\rangle.\, \mathsf{a2}(x). \\
& & \mathsf{let}\, z = \mathsf{exp}\,(x, n_A)\, \mathsf{in}\, \overline{\mathsf{a3}}\langle\{d\}_z\rangle.\, \mathbf{0} \\
B & = & (\mathsf{new}\, n_B)\, \mathsf{b1}(y).\, \overline{\mathsf{b2}}\langle \mathsf{exp}\,(\alpha, n_B)\rangle. \\
& & \mathsf{let}\, w = \mathsf{exp}\,(y, n_B)\, \mathsf{in}\, \mathsf{b3}(t). \\
& & \mathsf{let}\, t' = \mathsf{dec}_w(t)\, \mathsf{in}\, B' \\
P & = & A \,||\, B.
\end{array}
$$

Here B' is a continuation of B after the reception of the encrypted datum d.

Operational Semantics The semantics of the calculus is given in terms of a transition relation $\longrightarrow$, which will sometimes be referred to as 'concrete' (as opposed to the 'symbolic' one we shall introduce later on). We model the state of the system as a pair $\langle s,\ P\rangle$, where s records the current environment's knowledge (i.e., the sequence of messages the environment has 'seen' on the network up to a given moment) and P is a process term. Formally, an *action* is a term of the form $\mathsf{a}\langle M\rangle$ (*input* action) or $\overline{\mathsf{a}}\langle M\rangle$ (*output* action), for a a label and M a message. The set of actions Act is ranged over by $\alpha, \beta, \ldots$, while the set Act^* is ranged over by $s, s', \ldots$. String concatenation is written '$\cdot$'. We denote by $\mathrm{act}(s)$ and $\mathrm{msg}(s)$ the set of actions and mes-

sages, respectively, appearing in s, and write '$s \vdash M$' for $\mathrm{msg}(s) \vdash M$.

Below we define *traces*, that is, sequences of actions that may result from the interaction between a process and its environment. In traces, each message received by a process (input message) can be synthesised from the knowledge the environment has previously acquired.

Definition 2.5 (traces and configurations). *A* trace *is a string $s \in Act^*$ such that for each s_1, s_2 and $\mathsf{a}\langle M\rangle$, if $s = s_1 \cdot \mathsf{a}\langle M\rangle \cdot s_2$ then $s_1 \vdash M$. A* configuration*, written as $\langle s,\ P\rangle$, is a pair consisting of a ground trace s and a process P. A configuration is* initial *if $\mathrm{en}(s, P) = \emptyset$. Configurations are ranged over by $\mathcal{C}, \mathcal{C}', \cdots$.*

The concrete transition relation on configurations is defined by the rules in Table 3. Each action taken by the process is recorded in the configuration's first component. Rule (INP) makes the transition relation infinitely-branching, as M ranges over the infinite set $\{M : s \vdash M\}$. In rule (OUT), ζ is evaluated and the action takes place only if the outcome is a message. By rule (LET), the evaluation of ζ replaces any free occurrence of y in P. In rule (NEW), a function $\mathrm{new}_{\mathcal{LN}}(V)$ is assumed that yields a local name $a \notin V$. No handshake communication is provided: all messages go through the environment (rule (PAR)). Note that, by (OUT), the nondeterminism of $\rightsquigarrow$ is lifted to processes, and this is used to render commutativity and associativity of product. We are of course ignoring any 'efficiency' consideration, given that the model is infinite anyway (rule (INP)).

Properties We focus on correspondence assertions of the kind 'for every generated trace, whenever action β occurs in the trace, then action α must have occurred at some previous point in the trace'. Formally, given a configuration $\langle s,\ P\rangle$ and a trace s', we say that $\langle s,\ P\rangle$ *generates* s', written $\langle s,\ P\rangle \searrow s'$, if $\langle s,\ P\rangle \longrightarrow^* \langle s',\ P'\rangle$ for some P'. A substitution θ is a finite partial map from $\mathcal{V} \cup \widehat{\mathcal{V}}$ to the set of messages $\mathcal{M}$. For any object t (i.e. variable, message, process, trace, ...), we denote by $t\theta$ the result of simultaneously replacing each $x \in \mathrm{v}(t) \cap \mathrm{dom}(\theta)$ by $\theta(x)$. We let ρ range over ground substitutions, i.e. substitutions that map variables to ground messages.

Definition 2.6 (properties and satisfaction relation). *Let α and β be actions and s be a trace. We say that α* occurs prior to *β in s if whenever $s = s' \cdot \beta \cdot s''$ then $\alpha \in \mathrm{act}(s')$. For $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$, we write $s \models \alpha \hookleftarrow \beta$, and say s* satisfies *$\alpha \hookleftarrow \beta$, if for each ground substitution ρ it holds that $\alpha\rho$ occurs prior to $\beta\rho$ in s. We say that a configuration $\mathcal{C}$* satisfies *$\alpha \hookleftarrow \beta$, and write $\mathcal{C} \models \alpha \hookleftarrow \beta$, if all traces generated by $\mathcal{C}$ satisfy $\alpha \hookleftarrow \beta$.*

Assertions $\alpha \hookleftarrow \beta$ can express interesting authentication (see [7]) and secrecy properties. In particular, secrecy in the style of [3] can be set within our framework by assuming a conventional 'absurd' action $\perp$ that it is nowhere used in agent expressions. Thus, the formula $\perp \hookleftarrow \alpha$ means that action α should never take place.

Example 2.7 (Diffie-Hellman, continued). *The property that the protocol P in Example 2.4 should not leak the datum d can be expressed also by saying that the adversary will never be capable of synthesising d, without prior knowledge of it. This can be formalised by extending P with a 'guardian' process $\mathsf{g}(t).\,\mathbf{0}$ that at any time can pick up one message from the network and then stop: $S = P \parallel \mathsf{g}(t).\,\mathbf{0}$. Then we check whether this guardian can ever pick d from the network. This means checking whether the initial configuration $\mathcal{C}_{DH} = \langle \epsilon,\ S\rangle$ (ϵ = empty trace) satisfies the property $Secret(d) = \perp \hookleftarrow \mathsf{g}\langle d\rangle$, i.e. whether $\mathcal{C} \models Secret(d)$. Note that, by definition of deduction relation (Def. 2.2), $\epsilon \vdash \alpha$.*

3 Symbolic Semantics

We equip the frame $\mathcal{F}_{DH}$ with a *symbolic* evaluation relation ($\downarrow_s$), which is in agreement with its concrete counterpart ($\downarrow$). Intuitively, $\zeta \downarrow_\theta \eta$ means that ζ evaluates to η under all instances ρ of θ. The main advantage of the symbolic evaluation relation with respect to the concrete one is that infinitely many pairs (ζ, η) such that $\zeta \downarrow \eta$ can be represented by means of a single judgement $\zeta_0 \downarrow_\theta \eta_0$, for appropriate ζ_0, θ, η_0.

Formally, let us denote by $\mathrm{mgu}(t_1, t_2)$ a chosen *most general unifier* (mgu) of t_1 and t_2, that is, a unifier θ of t_1 and t_2 such that any other unifier is a composition of substitutions θ and θ', written $\theta\theta'$, for some θ'. Also, for t_1, t_1', t_2, t_2' terms, $\mathrm{mgu}(t_1 = t_1', t_2 = t_2')$ stands for $\theta\,\mathrm{mgu}(t_2\theta, t_2'\theta)$, where $\theta = \mathrm{mgu}(t_1, t_1')$, if such mgu's exist. The symbolic evaluation relation $\downarrow_s$ of $\mathcal{F}_{DH}$ is presented in Table 4: it is defined as the reflexive and transitive closure of the relation $\overset{\theta}{\rightsquigarrow}_s$.

Lemma 3.1. *Relation $\overset{\theta}{\rightsquigarrow}_s$ is image-finite and terminating. Hence, $\downarrow_s$ is image-finite.*

The notions of symbolic traces and symbolic configurations are defined below. The main difference from their concrete counterparts (Def. 2.5) is that no condition on input messages is required.

Definition 3.2 (symbolic traces and configurations). *A* symbolic trace *is a string $s \in Act^*$ such that: (a) $\mathrm{en}(s) = \emptyset$, and (b) for each s_1, s_2, α and x, if $s =$*

$$
\begin{array}{rrcll}
(\textsc{Inp}) & \langle s,\ \mathsf{a}(x).\,P\rangle & \longrightarrow & \langle s\cdot \mathsf{a}\langle M\rangle,\ P[^{M}\!/_{x}]\rangle & s \vdash M \\
(\textsc{Out}) & \langle s,\ \overline{\mathsf{a}}\langle\zeta\rangle.\,P\rangle & \longrightarrow & \langle s\cdot \overline{\mathsf{a}}\langle M\rangle,\ P\rangle & \zeta \downarrow M \\
(\textsc{Let}) & \langle s,\ \mathsf{let}\, y=\zeta\,\mathsf{in}\,P\rangle & \longrightarrow & \langle s,\ P[^{\eta}\!/_{y}]\rangle & \zeta \downarrow \eta \\
(\textsc{Match}) & \langle s,\ [\zeta=\eta]P\rangle & \longrightarrow & \langle s,\ P\rangle & \zeta\downarrow\xi,\ \eta\downarrow\xi \\
(\textsc{New}) & \langle s,\ (\mathsf{new}\,a)\,P\rangle & \longrightarrow & \langle s,\ P\rangle & a=\mathrm{new}_{\mathcal{LN}}(V) \\
(\textsc{Par}) & \multicolumn{3}{c}{\dfrac{\langle s,\ P\rangle \longrightarrow \langle s',\ P'\rangle}{\langle s,\ P\parallel Q\rangle \longrightarrow \langle s',\ P'\parallel Q\rangle}} &
\end{array}
$$

plus symmetric version of (PAR).
In (NEW), V is the set of free names in the source configuration.

Table 3: Rules for the concrete transition relation ($\longrightarrow$)

$$
\begin{array}{rll}
(\textsc{Dec}_{\mathrm{s}}) & \mathsf{dec}_{\eta}(\zeta) \overset{\theta}{\rightsquigarrow}_{s} x_1\,\theta & \theta=\mathrm{mgu}(\zeta=\{x_1\}_{x_2},\eta=x_2) \\
(\textsc{Mult}_{\mathrm{s}}) & \mathsf{mult}(\zeta_1,\zeta_n) \overset{\theta}{\rightsquigarrow}_{s} (x_{i_1}\times\cdots\times x_{i_n})\,\theta & \left\{\begin{array}{l} 1\le k<n\le l, \\ \theta=\mathrm{mgu}(\zeta_1=x_1\times\cdots\times x_k, \\ \zeta_2=x_{k+1}\times\cdots\times x_n)\end{array}\right. \\
(\textsc{Inv1}_{\mathrm{s}}) & \mathsf{inv}(\zeta) \overset{\theta}{\rightsquigarrow}_{s} (\mathsf{inv}'(x_{i_1})\times\cdots\times\mathsf{inv}'(x_{i_n}))\,\theta & \left\{\begin{array}{l} 1\le n\le l, \\ \theta=\mathrm{mgu}(\zeta=x_1\times\cdots\times x_n)\end{array}\right. \\
(\textsc{Inv2}_{\mathrm{s}}) & \mathsf{inv}'(\zeta) \overset{\theta}{\rightsquigarrow}_{s} x_1\,\theta & \theta=\mathrm{mgu}(\zeta,{x_1}^{-1}) \\
(\textsc{Inv3}_{\mathrm{s}}) & \mathsf{inv}'(\zeta) \overset{\epsilon}{\rightsquigarrow}_{s} \zeta^{-1} \qquad (\textsc{Inv4}_{\mathrm{s}}) \quad \mathsf{inv}'(\zeta)\times\eta \overset{\theta}{\rightsquigarrow}_{s} \mathsf{unit} & \theta=\mathrm{mgu}(\zeta,\eta) \\
(\textsc{Unit1}_{\mathrm{s}}) & \mathsf{unit}\times\zeta \overset{\epsilon}{\rightsquigarrow}_{s} \zeta \qquad (\textsc{Unit2}_{\mathrm{s}}) \quad \mathsf{unit} \overset{\epsilon}{\rightsquigarrow}_{s} 1 & \\
(\textsc{Exp1}_{\mathrm{s}}) & \mathsf{exp}\,(x,\zeta) \overset{\theta}{\rightsquigarrow}_{s} \mathsf{exp}\,(\alpha,\mathsf{mult}(x_1,\zeta)) & \theta=[^{\mathsf{exp}\,(\alpha,x_1)}\!/_{x}] \\
(\textsc{Exp2}_{\mathrm{s}}) & \mathsf{exp}\,(\mathsf{exp}\,(\xi,\eta),\zeta) \overset{\epsilon}{\rightsquigarrow}_{s} \mathsf{exp}\,(\xi,\mathsf{mult}(\eta,\zeta)) & \\
(\textsc{Root1}_{\mathrm{s}}) & \mathsf{root}\,(x,\zeta) \overset{\theta}{\rightsquigarrow}_{s} \mathsf{exp}\,(\alpha,\mathsf{mult}(x_1,\mathsf{inv}(\zeta))) & \theta=[^{\mathsf{exp}\,(\alpha,x_1)}\!/_{x}] \\
(\textsc{Root2}_{\mathrm{s}}) & \mathsf{root}\,(\mathsf{exp}\,(\xi,\eta),\zeta) \overset{\epsilon}{\rightsquigarrow}_{s} \mathsf{exp}\,(\xi,\mathsf{mult}(\eta,\mathsf{inv}(\zeta))) & \\
(\textsc{Ctx}_{\mathrm{s}}) & \dfrac{\zeta \overset{\theta}{\rightsquigarrow}_{s} \zeta'}{C[\zeta] \overset{\theta}{\rightsquigarrow}_{s} C\theta[\zeta']} &
\end{array}
$$

SYMBOLIC EVALUATION $\zeta\downarrow_{\theta}\eta$ iff $\zeta \overset{\theta_1}{\rightsquigarrow}_{s}\cdots\overset{\theta_n}{\rightsquigarrow}_{s}\eta$ and $\theta=\theta_1\cdots\theta_n$
Variables $x_1,\ \cdots,\ x_n$ are fresh according to some arbitrary but fixed rule.

Table 4: Symbolic Evaluation Relation ($\downarrow_s$) for $\mathcal{F}_{DH}$

$s_1 \cdot \alpha \cdot s_2$ *and* $x \in \mathrm{v}(\alpha) - \mathrm{v}(s_1)$ *then* α *is an input action. Symbolic traces are ranged over by* $\sigma, \sigma', \ldots$. *A* symbolic configuration, *written* $\langle \sigma,\ A \rangle_{\mathrm{s}}$, *is a pair composed by a symbolic trace* σ *and an agent* A, *such that* $\mathrm{en}(A) = \emptyset$ *and* $\mathrm{v}(A) \subseteq \mathrm{v}(\sigma)$.

The symbolic semantics is based on a symbolic transition relation $\longrightarrow_{\mathrm{s}}$, defined in Table 5. There, a function $\mathrm{new}_{\mathcal{V}}(\cdot)$ is assumed such that, for any given $V \subseteq_{\mathrm{fin}} \mathcal{V}$, $\mathrm{new}_{\mathcal{V}}(V)$ is a variable not in V. Moreover, $\mathcal{C} \xrightarrow{\theta}_{\mathrm{s}} \mathcal{C}'$ stands for $\mathcal{C} \longrightarrow_{\mathrm{s}} \mathcal{C}'$, where θ is the substitution applied to $\mathcal{C}'$ in the reduction step.

Note that, unlike the concrete semantics, input variables are *not* instantiated immediately (rule ($\textsc{Inp}_{\mathrm{s}}$)). Rather, the input message is represented as a free variable and constraints on this variable are added as soon as they are needed, and recorded via mgu's. This may occur due to rules ($\textsc{Out}_{\mathrm{s}}$), ($\textsc{Let}_{\mathrm{s}}$) and ($\textsc{Match}_{\mathrm{s}}$).

Example 3.3. *The simple example below shows how, after a first step, variable* x *is instantiated to name* b *by a (*$\textsc{Match}_{\mathrm{s}}$*)-reduction:*

$$\begin{array}{lll} \langle \epsilon,\ \mathsf{a}(x).\,[x=b]P \rangle_{\mathrm{s}} & \longrightarrow_{\mathrm{s}} & \langle \mathsf{a}\langle x\rangle,\ [x=b]P \rangle_{\mathrm{s}} \\ & \longrightarrow_{\mathrm{s}} & \langle \mathsf{a}\langle b\rangle,\ P[b/x] \rangle_{\mathrm{s}}. \end{array}$$

As a more elaborate example, let $P = \overline{\mathsf{a}}\langle k\rangle.\,\mathsf{a}(x).\,\mathsf{let}\, z = \mathsf{root}\,(x,k)\,\mathsf{in}\, P'$. *After an output action and an input action, the symbolical evaluation of* $\mathsf{root}\,(x,k)$ *produces a global substitution* $\theta = [\mathsf{exp}\,(\alpha, x_1)/x]$ *(*x_1 *fresh), to be applied to the whole configuration, and a local substitution* $\theta' = [\mathsf{exp}\,(\alpha, x_1 \times k_{-1})/z]$. *I.e.*

$$\langle \epsilon,\ P \rangle_{\mathrm{s}} \longrightarrow^{*}_{\mathrm{s}} \langle \sigma\theta,\ P'\theta\theta' \rangle_{\mathrm{s}}, \quad \textit{with } \sigma = \overline{\mathsf{a}}\langle k\rangle \cdot \mathsf{a}\langle x\rangle.$$

Whenever $\langle \sigma,\ A \rangle_{\mathrm{s}} \longrightarrow^{*}_{\mathrm{s}} \langle \sigma',\ A' \rangle_{\mathrm{s}}$ for some A', we say that $\langle \sigma,\ A \rangle_{\mathrm{s}}$ *symbolically generates* σ', and write $\langle \sigma,\ A \rangle_{\mathrm{s}} \searrow_{\mathrm{s}} \sigma'$. The relation $\longrightarrow_{\mathrm{s}}$ is finitely-branching since $\downarrow_s$ is. Therefore, each configuration generates a finite number of symbolic traces. It is important to stress that many symbolic traces are in fact nonsense – sequences of actions that cannot be instantiated to any concrete trace. For instance, consider process $P = \mathsf{a}(y).\,\mathsf{let}\, x = \mathsf{dec}_k(y)\,\mathsf{in}\,\overline{\mathsf{a}}\langle x\rangle.\,\mathbf{0}$. The initial configuration $\langle \epsilon,\ P \rangle_{\mathrm{s}}$ symbolically generates the trace $\mathsf{a}\langle\{x_0\}_k\rangle \cdot \overline{\mathsf{a}}\langle x_0\rangle$, which is inconsistent, because the environment cannot generate the value k in $\{x_0\}_k$ (i.e. $\epsilon \not\vdash k$, hence $\epsilon \not\vdash \{x_0\}_k$). The problem of detecting these inconsistent traces, that might give rise to 'false positives' when checking protocol properties, will be faced in the next section.

Theorem 3.5 below establishes a correspondence between the concrete and the symbolic transition relations. It relies on the notion of consistency, defined below. Recall that marked variables are intended to carry messages known by the environment. For any $\hat{x}$ and any trace σ, we denote by $\sigma\backslash\hat{x}$ the longest prefix of σ not containing $\hat{x}$.

Definition 3.4 (consistency). *Let* $\sigma \in Act^*$ *and* ρ *be a ground substitution. We say that* ρ satisfies σ *if* $\sigma\rho$ *is a ground trace and, for each* $\hat{x} \in \mathrm{v}(\sigma)$, *it holds that* $(\sigma\backslash\hat{x})\rho \vdash \rho(\hat{x})$. *In this case we say that* $\sigma\rho$ *is a* solution *of* σ, *and that* σ *is* consistent.

Note that, because of the requirements on input messages, any trace (Def. 2.5) is obviously consistent: e.g., map all variables to some environmental names.

Theorem 3.5 (concrete vs. symbolic semantics). $\mathcal{C}$ *be an initial configuration and* s *be a ground trace. Then* $\mathcal{C} \searrow s$ *if and only if there exists* σ *such that* $\mathcal{C} \searrow_{\mathrm{s}} \sigma$ *and* s *is a solution of* σ.

4 A Verification Method

A crucial point of the method we present is checking consistency of symbolic traces. As mentioned in the previous section, a symbolic trace σ need not have solutions (ground instances that are traces). The next result allows us to check consistency.

Proposition 4.1. *Let* σ *be a symbolic trace. Then there exists a finite set of traces* $\mathbf{Refinement}(\sigma)$, *which are instances of* σ *and have the following property: for any* s, s *is a solution of* σ *if and only if* s *is a solution of some* $\sigma' \in \mathbf{Refinement}(\sigma)$.

Proposition 4.1 implies that σ is consistent if and only if $\mathbf{Refinement}(\sigma) \neq \emptyset$. Roughly, the set $\mathbf{Refinement}(\sigma)$ is computed by repeatedly unifying each input message in σ to terms that can be synthetized out of previous messages in σ. We refer to the Appendix for details. Here we shall content ourselves with giving a couple of examples; the second example shows the important role of the root extraction operation.

Example 4.2.

1. *Consider* $\sigma = \overline{\mathsf{c}}\langle\{m\}_k\rangle \cdot \overline{\mathsf{c}}\langle\{n\}_k\rangle \cdot \mathsf{c}\langle\{x\}_k\rangle$. *Then* $\mathbf{Refinement}(\sigma) = \{\, \sigma[m/x], \sigma[n/x] \,\}$.
2. *Consider* $\sigma' = \overline{\mathsf{c}}\langle h\rangle \cdot \mathsf{c}\langle x\rangle \cdot \overline{\mathsf{c}}\langle \mathsf{exp}\,(\alpha, k \times h)\rangle \cdot \overline{\mathsf{c}}\langle\{m\}_{\mathsf{exp}\,(\alpha, k\times x)}\rangle \cdot \mathsf{c}\langle m\rangle$. *Clearly,* σ' *is consistent: e.g., map* x *to* h. *And, indeed,* $\mathbf{Refinement}(\sigma') = \{\sigma''\}$ *where* $\sigma'' = \sigma'[\hat{x}/x]$. *It is notable that the root extraction operation, though not mentioned*

$$
\begin{array}{lrcll}
(\textsc{Inp}_{\mathrm{s}}) & \langle \sigma,\ \mathsf{a}(x).\,A\rangle_{\mathrm{s}} & \longrightarrow_{\mathrm{s}} & \langle \sigma \cdot \mathsf{a}\langle x\rangle,\ A\rangle_{\mathrm{s}} & \\
(\textsc{Out}_{\mathrm{s}}) & \langle \sigma,\ \overline{\mathsf{a}}\langle\zeta\rangle.\,A\rangle_{\mathrm{s}} & \longrightarrow_{\mathrm{s}} & \langle \sigma\theta \cdot \overline{\mathsf{a}}\langle M\rangle,\ A\theta\rangle_{\mathrm{s}} & \zeta \downarrow_\theta M \\
(\textsc{Let}_{\mathrm{s}}) & \langle \sigma,\ \mathsf{let}\, x = \zeta\, \mathsf{in}\, A\rangle_{\mathrm{s}} & \longrightarrow_{\mathrm{s}} & \langle \sigma\theta,\ A\theta[\xi/x]\rangle_{\mathrm{s}} & \zeta \downarrow_\theta \xi \\
(\textsc{Match}_{\mathrm{s}}) & \langle \sigma,\ [\zeta = \eta]A\rangle_{\mathrm{s}} & \longrightarrow_{\mathrm{s}} & \langle \sigma\theta,\ A\theta\rangle_{\mathrm{s}} & \left\{ \begin{array}{l} \zeta \downarrow_{\theta_1} \xi_1,\ \eta\,\theta_1 \downarrow_{\theta_2} \xi_2, \\ \theta_3 = \mathrm{mgu}(\xi_1\theta_2, \xi_2), \\ \theta = \theta_1\theta_2\theta_3 \end{array} \right. \\
(\textsc{New}_{\mathrm{s}}) & \langle \sigma,\ (\mathsf{new}\, a)\, A\rangle_{\mathrm{s}} & \longrightarrow_{\mathrm{s}} & \langle \sigma,\ A\rangle_{\mathrm{s}} & a = \mathrm{new}_{\mathcal{LN}}(V) \\
(\textsc{Par}_{\mathrm{s}}) & \multicolumn{3}{c}{\dfrac{\langle \sigma,\ A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma',\ A'\rangle_{\mathrm{s}}}{\langle \sigma,\ A \parallel B\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma',\ A' \parallel B\rangle_{\mathrm{s}}}} &
\end{array}
$$

plus symmetric version of ($\textsc{Par}_{\mathrm{s}}$).
(i) $x = \mathrm{new}_{\mathcal{V}}(V)$, being V the set of free variables in the source configuration;
(ii) $\mathrm{msg}(\sigma)\theta \subseteq \mathcal{M}$;
(iii) in rule ($\textsc{Par}_{\mathrm{s}}$), $B' = B\theta$ where $\langle \sigma,\ A\rangle_{\mathrm{s}} \xrightarrow{\theta}_{\mathrm{s}} \langle \sigma',\ A'\rangle_{\mathrm{s}}$.

Table 5: Rules for symbolic transition relation ($\longrightarrow_{\mathrm{s}}$)

in σ'', is essential to prove that σ'' is a trace. In fact, the environment is capable of learning m only by computing the key of the encrypted message as $\mathsf{exp}\,(\mathsf{root}\,(\mathsf{exp}\,(\alpha, k \times h), h), \hat{x})$.

Let $\alpha \hookleftarrow \beta$ be a property and $\mathcal{C}$ an initial configuration. The verification method $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$, presented in Table 6, checks whether $\mathcal{C} \models \alpha \hookleftarrow \beta$ or not. Moreover, if the property is not satisfied, $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ computes a trace violating the property, that is, an attack on $\mathcal{C}$.

To understand how the method works, it is convenient to consider the simple case $\alpha = \bot$, i.e. the verification of $\mathcal{C} \models \bot \hookleftarrow \beta$. This means verifying that in the *concrete* semantics, no instance of action β is ever executed starting from $\mathcal{C}$. By the correspondence between symbolic and concrete semantics (Theorem 3.5), this amounts to checking that for each σ symbolically generated by $\mathcal{C}$, no solution of σ contains an instance of β. The method proceeds as follows. First, one checks whether there is a mgu θ of γ and β, for every action γ of σ. If, for every σ, such a θ does not exist, or if it exists but $\sigma\theta$ is not consistent (this means that the check $\exists\, \sigma' \in \mathbf{Refinement}(\sigma\theta)$ at step 5 fails), then the property holds true, otherwise it does not, and the trace σ' violating the property is reported.

The correctness of the method in the general case is stated by Theorem 4.3 below. Note that the method always terminates, given that $\longrightarrow_{\mathrm{s}}$ is finite-branching and, hence, $\mathbf{Mod}_{\mathcal{C}}$ is finite.

Theorem 4.3 (correctness and completeness). *Let $\mathcal{C}$ be an initial configuration and α and β be actions with* $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$.

(1) *If* $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ *returns* $(\mathsf{No},\ \sigma')$ *then* $\mathcal{C} \not\models \alpha \hookleftarrow \beta$. *In particular, for any injective ground substitution* $\rho : \mathrm{v}(\sigma') \to \mathcal{EN}$, $\mathcal{C} \searrow \sigma'\rho$ *and* $\sigma'\rho \not\models \alpha \hookleftarrow \beta$.

(2) *If* $\mathcal{C} \not\models \alpha \hookleftarrow \beta$ *then* $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ *returns* $(\mathsf{No},\ \sigma')$ *and for any injective ground substitution* $\rho : \mathrm{v}(\sigma') \to \mathcal{EN}$, $\mathcal{C} \searrow \sigma'\rho$ *and* $\sigma'\rho \not\models \alpha \hookleftarrow \beta$.

Remark 4.4. *In practice, rather than generating the whole set of symbolic traces at once (step 1) and then checking the property, it is more convenient to work 'on-the-fly' and comparing every last symbolic action γ taken by the configuration against action β of the property $\alpha \hookleftarrow \beta$. The refinement procedure* $\mathbf{Refinement}(\cdot)$ *is invoked only when β and γ are unifiable.*

5 Example: Symbolic Analysis of the Diffi e-Hellman Protocol

In this section we apply the verification method described in Section 4 to analyse the Diffie-Hellman protocol. We adopt the formalisation of the protocol presented in Examples 2.4 and 2.7.

The Diffie-Hellman protocol is subject to attacks from active adversaries. In terms of our model, discovering an attack on the protocol amounts to finding a ground trace s such that the initial configuration

$\mathbf{M}(\mathcal{C}, \alpha \leftrightarrows \beta)$

1. compute $\mathbf{Mod}_{\mathcal{C}} = \{\sigma \mid \mathcal{C} \searrow_{\mathsf{s}} \sigma\}$;
2. **foreach** $\sigma \in \mathbf{Mod}_{\mathcal{C}}$ **do**
3. **foreach** action γ in σ **do**
4. **if** $\exists\, \theta = \mathrm{mgu}(\beta, \gamma)$ **and**
5. $\exists\, \sigma' \in \mathbf{Refinement}(\sigma\theta)$ **where** $\sigma' = \sigma\theta\theta'$ **and**
6. $\alpha\theta\theta'$ does not occur prior to $\beta\theta\theta'$ in σ'
7. **then return**$(\mathsf{No},\ \sigma')$;
8. **return**(Yes);

Table 6: The verification method

$\mathcal{C}_{DH} = \langle \epsilon,\ S \rangle \searrow s$ and $s \not\models \mathit{Secret}(d) = \bot \leftrightarrows \mathsf{g}\langle d \rangle$. And, indeed, such an s exists and is as follows:

$$\overline{\mathsf{a1}}\langle \mathsf{exp}\,(\alpha, n_A) \rangle \cdot \mathsf{a2}\langle \mathsf{exp}\,(\alpha, \underline{n}_I) \rangle \cdot \overline{\mathsf{a3}}\langle \{d\}_H \rangle \cdot \mathsf{g}\langle d \rangle,$$

where $\underline{n}_I$ is any environmental name and $K = \mathsf{exp}\,(\alpha, \underline{n}_I \times n_A)$.

Intuitively, the above trace corresponds to an attack in which the environment intercepts $\mathsf{exp}\,(\alpha, n_A)$, generates a private name $\underline{n}_I$ and handles $\mathsf{exp}\,(\alpha, \underline{n}_I)$ to A. Then, A computes $K = \mathsf{exp}\,(\alpha, \underline{n}_I \times n_A)$ and erroneously believes K is a shared key known by A and B. Finally, A sends over the network the secret datum d encrypted under the corrupted shared key K and, so, d is revealed to the environment.

Now, we have to show how this attack is discovered by our verification method. First, consider the following symbolic execution starting from $\mathcal{C}_{DH}$:

$$\begin{aligned}
&\mathcal{C}_{DH} \longrightarrow_{\mathsf{s}} \longrightarrow_{\mathsf{s}} \\
&\longrightarrow_{\mathsf{s}} \langle \overline{\mathsf{a1}}\langle \mathsf{exp}\,(\alpha, n_A) \rangle \cdot \mathsf{a2}\langle x \rangle, \\
&\qquad \mathsf{let}\ z = \mathsf{exp}\,(x, n_A)\ \mathsf{in}\ \overline{\mathsf{a3}}\langle \{d\}_z \rangle.\, \mathbf{0} \parallel B \parallel \mathsf{g}(t).\, \mathbf{0} \rangle_{\mathsf{s}} \\
&\xrightarrow{\theta_0}_{\mathsf{s}} \langle \overline{\mathsf{a1}}\langle \mathsf{exp}\,(\alpha, n_A) \rangle \cdot \mathsf{a2}\langle \mathsf{exp}\,(\alpha, x_0) \rangle, \\
&\qquad \overline{\mathsf{a3}}\langle \{d\}_K \rangle.\, \mathbf{0} \parallel B\theta_0\theta_1 \parallel \mathsf{g}(t).\, \mathbf{0} \rangle_{\mathsf{s}} \qquad (*) \\
&\longrightarrow_{\mathsf{s}} \langle \overline{\mathsf{a1}}\langle \mathsf{exp}\,(\alpha, n_A) \rangle \cdot \mathsf{a2}\langle \mathsf{exp}\,(\alpha, x_0) \rangle \cdot \overline{\mathsf{a3}}\langle \{d\}_K \rangle, \\
&\qquad \mathbf{0} \parallel B\theta_0\theta_1 \parallel \mathsf{g}(t).\, \mathbf{0} \rangle_{\mathsf{s}} \\
&\longrightarrow_{\mathsf{s}} \langle \overline{\mathsf{a1}}\langle \mathsf{exp}\,(\alpha, n_A) \rangle \cdot \mathsf{a2}\langle \mathsf{exp}\,(\alpha, x_0) \rangle \cdot \overline{\mathsf{a3}}\langle \{d\}_K \rangle \cdot \mathsf{g}\langle t \rangle, \\
&\qquad \mathbf{0} \parallel B\theta_0\theta_1 \parallel \mathbf{0} \rangle_{\mathsf{s}} \\
&\quad = \langle \sigma,\ \mathbf{0} \parallel B\theta_0\theta_1 \parallel \mathbf{0} \rangle_{\mathsf{s}},
\end{aligned}$$

where $K = \mathsf{exp}\,(\alpha, x_0 \times n_A)$. In step $(*)$, rule (LET$_{\mathsf{s}}$) is applied, with $\mathsf{exp}\,(x, n_A) \downarrow_{\theta_0} \mathsf{exp}\,(\alpha, x_0 \times n_A)$, $\theta_0 = [\mathsf{exp}\,(\alpha, x_0)/x]$ (x_0 fresh), and $\theta_1 = [\mathsf{exp}\,(\alpha, x_0)/z]$.

Now, we execute the method step by step:

1. The symbolic model $\mathbf{Mod}_{\mathcal{C}_{DH}}$ is computed (in practice, symbolic traces are generated and checked 'on-the-fly').

2. The symbolic trace σ defined above is considered.

3,4. Action $\gamma = \mathsf{g}\langle t \rangle$ is found that unifies with $\beta = \mathsf{g}\langle d \rangle$, via $\theta = [d/t]$.

5. The set $\mathbf{Refinement}(\sigma\theta) = \{\sigma'\}$ is computed where $\sigma' = \sigma\theta\theta'$, and $\theta' = [\hat{x}_0/x_0]$. As stated by Theorem 4.1, σ' is a consistent trace. Note, in particular, that if we let $\sigma' = \sigma'' \cdot \mathsf{g}\langle d \rangle$, then $\sigma'' \vdash d$. Indeed, there exists $\zeta = \mathsf{dec}_{\xi}(\{d\}_{\eta}) \in \mathcal{H}(\sigma'')$, with $\eta = \mathsf{exp}\,(\alpha, \hat{x}_0 \times n_A)$, $\xi = \mathsf{exp}\,(\mathsf{exp}\,(\alpha, n_A), \hat{x}_0)$, and $\zeta \rightsquigarrow_s \rightsquigarrow_s \mathsf{dec}_{\eta}(\{d\}_{\eta}) \rightsquigarrow_s d$.

6. Action $\bot$ does not appear in σ', hence,

7. $(\mathsf{No},\ \sigma')$ is returned. Note that s is retrieved as $\sigma'[\underline{n}_I/\hat{x}_0]$.

Of course, here we have proceeded by hand, but the symbolic traces generated by $\mathcal{C}_{DH}$ are of the order of hundreds, hence we expect that an implementation of the method would detect this attack in a fraction of second.

It is straightforward to extend the frame of Section 2 with operations such as pairing, public key encryption and decryption, hashing and digital signature; we refer the reader to [7] for a detailed treatment of those operations. In particular, digital signature allows one to describe the authenticated version of the Diffie-Hellman protocol, where the public values exchanged are signed by the two partners, so that the man-in-the-middle attack is thwarted.

6 Conclusions and future work

We have presented a model for the analysis of protocols built around shared-key encryption and modular exponentiation. The model is less precise than others (notably [20]), its main limitation being a bound on the number of factors that may appear in any exponent. However, for such restricted model, we offer a decision

method. The model and the method smoothly carry over when including other common enciphering, signing and hashing primitives. We also believe the method is effective in practice, because the symbolic model is compact, and the refinement procedure at its heart is only invoked on demand and on single symbolic traces. We are in the process of integrating the technique presented here into the STA analysis tool ([6, 23]).

Our technical development has been confined to multiplication and exponentiation, but the methodology presented suggests directions for extensions to other low-level primitives. It certainly seems promising for the case of RSA encryption, which is based on exponentiation too.

Acknowledgements. The authors wish to thank the anonymous referees for their helpful comments.

References

[1] M. Abadi, C. Fournet. Mobile Values, New Names, and Secure Communication. In *Conf. Rec. of POPL'01*, 2001.

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

[3] R.M. Amadio, S. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. of Concur'00*, LNCS 1877, Springer-Verlag, 2000. Full version: RR 3915, INRIA Sophia Antipolis.

[4] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. of 14th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001.

[5] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proc. of ICALP'01*, LNCS 2076, Springer-Verlag, 2001.

[6] M. Boreale, M. Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Security Protocols. In *Proc. of SAC'02*, ACM Press, 2002.

[7] M. Boreale and M. Buscemi. A Framework for the Analysis of Security Protocol. In *Proc. of CONCUR '02*, LNCS 2421. Springer-Verlag, 2002.

[8] Y. Chevalier, R. Kuesters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with Xor. In *Proc. of LICS '03*, IEEE Computer Society Press, 2003.

[9] H. Comon, V. Cortier, J. Mitchell. Tree automata with one memory, set constraints and ping-pong protocols. In *Proc. of ICALP'01*, LNCS 2076, Springer-Verlag, 2001.

[10] H. Comon-Lundh and V. Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or. In *Proc. LICS '03*, IEEE Computer Society Press, 2003.

[11] W. Diffie, M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, 1976.

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

[13] A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.

[14] D. Kapur, P. Narendran, and L. Wang. An E-unification Algorithm for Analyzing Protocols that Use Modular Exponentiation. In *Proc. of RTA '03*, LNCS 2706, Springer-Verlag, 2003.

[15] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proc. of TACAS'96*, LNCS 1055, Springer-Verlag, 1996.

[16] G. Lowe. A Hierarchy of Authentication Specifications. In *Proc. of 10th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1997.

[17] W. Marrero, E.M. Clarke, S. Jha. Model checking for security protocols. Technical Report TR-CMU-CS-97-139, Carnegie Mellon University, 1997.

[18] A Menezes, P.C. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

[19] J. Millen, V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of 8th ACM Conference on Computer and Communication Security*, ACM Press, 2001.

[20] J. Millen and V. Shmatikov. Symbolic Protocol Analysis with Products and Diffie-Hellman Exponentiation. In *Proc. of 16th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 2003.

[21] J.C. Mitchell, M. Mitchell, U Stern. Automated Analysis of Cryptographic Protocols Using Murφ. In *Proc. of Symp. Security and Privacy*, IEEE Computer Society Press, 1997.

[22] O. Pereira and J.J Quisquater. A Security Analysis of the Cliques Protocols Suites. In *Proc. of the 14th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001.

[23] STA: a tool for trace analysis of cryptographic protocols. ML object code and examples. Available at `http://www.dsi.unifi.it/~boreale/tool.html`.

[24] V. Vanack`ere. The TRUST Protocol Analyser, Automatic and Efficient Verification of Cryptographic Protocols. In *Proc. of Verify '02*, 2002.

A Computing Refinement

Here we move within the general framework of [7][2]. First, we introduce the concept of *basis*, which intuitively consists of the 'building blocks' of all messages deducible from a given σ. Given generic products F and G, we write $G \subset F$ to indicate that factors of G are strictly included in those of F.

Definition A.1 (a basis for $\mathcal{F}_{DH}$). *For each symbolic trace σ:*

$$\begin{aligned}\mathbf{b}_{DH}(\sigma) = \{M \mid (\sigma \vdash M) \text{ and } \big(\ & M \in \mathcal{LN} \cup \{\alpha, 1\} \\ & \text{or } (M = F \text{ and } \sigma \not\vdash G, \forall\, G \subset F) \\ & \text{or } (M = \mathsf{exp}(\alpha, F) \text{ and } \sigma \not\vdash G, \forall\, G \subset F)) \\ & \text{or } (M = \{M\}_K \text{ and } \sigma \not\vdash K)\ \big)\ \}.\end{aligned}$$

In practice, for a given σ, the set $\mathbf{b}_{DH}(\sigma)$ can be effectively computed by an iterative procedure which repeatedly applies destructors (dec, root, mult, inv, inv′) to messages in σ, until some fixed point is reached. This procedure always terminates and yields a finite set of messages.

Definition A.2 ($\mathbf{Refinement}(\sigma)$). *We let* refinement, *written $\succ$, be the least binary relation over symbolic traces given by the two rules below. In (*$\textsc{Ref}_1$*), it is assumed: that σ' is the longest prefix of σ which is a trace, that $\sigma = \sigma' \cdot \mathsf{a}\langle M\rangle \cdot \sigma''$, for some σ'', that $N, N' \notin \mathcal{V} \cup \widehat{\mathcal{V}}$, and that $\theta \neq \epsilon$.*

$$(\textsc{Ref}_1)\ \frac{N' \in \mathbf{b}(\sigma') \quad M = C[N] \quad \theta = \mathrm{mgu}(N, N')}{\sigma \ \succ\ \sigma\theta\theta_0}$$

$$(\textsc{Ref}_2)\ \frac{x \in \mathrm{v}(M)}{\sigma \ \succ\ \sigma[\hat{x}/x]}$$

where $\theta_0 = [\, x/\hat{x} \mid \hat{x} \in \mathrm{v}(\sigma)$ and $|(\sigma\theta)\backslash\hat{x}| < |\sigma\backslash\hat{x}|\,]$.

For any symbolic trace σ, we let $\mathbf{Refinement}(\sigma) = \{\, \sigma' \mid \sigma \succ {}^*\sigma'$ *and σ' is a trace* $\}$.

Rule ($\textsc{Ref}_1$) implements the basic step of refinement: the subterm N of M gets unified, via θ, with an element of $\mathbf{b}(\sigma')$. By rule ($\textsc{Ref}_2$) a variable x can become marked: this amounts to constraining the possible values of x to be messages known by the environment. (For technical reasons, marked variables sometimes need to be 'unmarked' back to plain variables, and this is achieved in ($\textsc{Ref}_1$) via the renaming θ_0.)

Lemma A.3. $\mathbf{Refinement}(\sigma)$ *can be effectively computed, and is finite.*

Proof. It follows by two facts: (a) $\succ$ is an image-finite relation, and (b) infinite sequences of refinement steps cannot arise. As to the latter point, note that, since each ($\textsc{Ref}_1$)-step eliminates at least one variable, any sequence of refinement steps can contain only finitely many ($\textsc{Ref}_1$)-steps. After the last such step, rule ($\textsc{Ref}_2$) can only be applied a finite number of times. □

[2] With a small change in terminology: in [7], consistent symbolic traces are called 'solved forms', while $\mathbf{Refinement}(\cdot)$ is denoted by $\mathbf{SF}(\cdot)$