Symbolic Analysis of Crypto-Protocols based on Modular Exponentiation*

Michele Boreale[1] and Maria Grazia Buscemi[2]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Italy.
[2] Dipartimento di Informatica, Università di Pisa, Italy.
boreale@dsi.unifi.it, 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 (e.g. [13]) 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 ([2, 3, 8, 14]), have emerged over the past few years. Implementations of these techniques (e.g. [4, 16]) 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 [11] assumptions - even though the protocol's state space is infinite. Second, symbolic methods usually rely on representations of data that 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 [5], we devise a complete analysis method for protocols that depend on modular exponentiation operations, like the Diffie-Hellman key-exchange [10]. We expect that our methodology may be adapted to other low-level primitives (like RSA encryption).

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

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 $\mathcal{Z}_p^* = \{1, \ldots, p-1\}$. Assume A and B want to establish a shared secret key. 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 mod p):

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

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 $K = \mathsf{exp}\,(\mathsf{exp}\,(\alpha, n_A), n_B) = \mathsf{exp}\,(\alpha, n_A \times n_B)$. Now A and B share K, and A can use it to, say, encrypt a secret datum d and send it to B:

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

The protocol's security depends on the difficulty of the discrete logarithm problem: computing y is computationally infeasible 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 [12] 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 non-determinism. As an example, upon evaluation of the expression $\mathsf{exp}\,(\mathsf{exp}\,(\alpha, n), m)$, an attacker will non-deterministically 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 since *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 [5], 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 2, 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 2, which asserts the correctness and completeness of the method with respect to the concrete model. Remarkably, the presence of the modular root operation plays a crucial role in the completeness proof. Directions for further research are discussed in Section 5. An extended version of the present paper is available as [6]. Complete proofs will appear in a forthcoming full version.

Very recent work by Millen and Shmatikov [15] 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. 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.* [7] and, independently, by Comon-Lundh and Shmatikov [9].

2 The model

We recall here the concept of *frame* from [5], and tailor it to the case of modular exponentiation and multiplication. 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 place-holders 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 ζ, thus yielding an expression $C[\zeta]$.

Definition 1 (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*. 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: 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.

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 $\rightsquigarrow$

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

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

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

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), along with the rules, ensure termination of both $\rightsquigarrow$ and the induced narrowing relation, introduced in Sec. 3.

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 the conditions are just not guaranteed to produce any message. Restriction (1) might be relaxed at the cost of introducing a class of operations mult_l, for each $l \geq 0$, but for simplicity we shall stick to the above model in this paper.

The deduction relation below 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. $\mathcal{P}_f(X)$ denotes the set of finite subsets of X.

Definition 2 (deduction relation). *For $S \subseteq \mathcal{M}$, the set $\mathcal{H}(S)$ is inductively defined by the following rules:*

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

The deduction relation $\vdash \subseteq \mathcal{P}_f(\mathcal{M}) \times \mathcal{M}$ *is defined as: $S \vdash M$ if and only if there exists $\zeta \in \mathcal{H}(S)$ such that $\zeta \downarrow M$.*

For instance, let $S = \{ \ n_A, \mathsf{exp}\,(\alpha, n_B) \ \}$. Then, $S \vdash \mathsf{exp}\,(\alpha, n_A \times n_B)$ and $S \vdash \mathsf{exp}\,(\alpha, n_B \times n_A)$. As another example, let $S = \{ \ \{m\}_{\mathsf{exp}\,(\alpha, k \times l)}, \ \mathsf{exp}\,(\alpha, k \times h), \ h, \ l \ \}$. 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. $\zeta \downarrow m$.

We now present a calculus which is a variant of the applied pi [1]. We consider a set $\mathcal{L}$ of *labels* which is ranged over by $\mathsf{a}, \mathsf{b}, \ldots$ and assume a unique public channel; thus input and output labels $(\mathsf{a}, \mathsf{b}, \ldots)$ are simply 'tags' attached to process actions for ease of reference. The syntax of *agents* is as follows:

$$A, B \ ::= \ \mathbf{0} \ \mid \ \mathsf{a}(x).\,A \ \mid \ \overline{\mathsf{a}}\langle\zeta\rangle.\,A \ \mid \ \mathsf{let}\,x = \zeta\,\mathsf{in}\,A \ \mid \ [\zeta = \eta]A \ \mid \ A \parallel B \ \mid \ (\mathsf{new}\,a)\,A.$$

The occurrences of variable x are bound in input and let operators. Notions of *free variables* ($\mathrm{v}(A) \subseteq \mathcal{V}$), *substitution* ($[\zeta/u]$), and *$\alpha$-equivalence* arise as expected. We denote by $\mathrm{v}(A)$ the set of free variables of A. An agent A is a *process* if $\mathrm{v}(A) = \emptyset$; $P, Q, \ldots$ range over the set of processes $\mathcal{P}$.

Example 1 (the Diffie-Hellman key exchange). The process P below is a formalisation of a one-session version of the Diffie-Hellman protocol:

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

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

The states of a protocol model are pairs $\langle s, \ P\rangle$, where s records the current environment's knowledge and P is a process term. 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 of strings of actions Act^* is ranged over by $s, s', \ldots$. String concatenation is written '$\cdot$'. $\mathrm{act}(s)$ and $\mathrm{msg}(s)$ are the sets of actions and messages, respectively, appearing in s, and $s \vdash M$ stands for $\mathrm{msg}(s) \vdash M$.

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 3 (traces and configurations). *A* trace *is a string $s \in Act^*$ such that $\forall s_1$, 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 composed by 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 semantics of the calculus is given in terms of a transition relation, which is also referred to as 'concrete' (as opposed to the 'symbolic' one discussed in the next section). Given the evaluation relation ($\downarrow$), the concrete transition relation $\longrightarrow$ is standard. Hence, here we just present the two most relevant transition rules and refer the reader to [5] for a complete treatment.

$$\begin{array}{lll} (\textsc{Inp})\ \langle s,\ \mathsf{a}(x).\, P\rangle \longrightarrow \langle s \cdot \mathsf{a}\langle M\rangle,\ P[{}^{M}\!/\!_{x}]\rangle & & s \vdash M \\ (\textsc{Out})\ \langle s,\ \overline{\mathsf{a}}\langle \zeta\rangle.\, P\rangle \longrightarrow \langle s \cdot \overline{\mathsf{a}}\langle M\rangle,\ P\rangle & & \zeta \downarrow M \end{array}$$

Rule ($\textsc{Inp}$) makes the transition relation infinitely-branching, as M ranges over the infinite set $\{M\ :\ s \vdash M\}$. Rule ($\textsc{Out}$) allows to lift the non-determinism of $\rightsquigarrow$ to processes; this is used to render commutativity and associativity of product.

The security properties that can be formalised within our model are 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'. These correspondence assertions are defined below.

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 maps variables to messages; we let ρ range over ground substitutions.

Definition 4 (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 and secrecy properties. We set secrecy in the style of [2] within our framework by assuming a conventional 'absurd' action $\perp$ that it is nowhere used in agent expressions. Thus, $\perp \hookleftarrow \alpha$ means that action α should never take place.

Example 2 (Diffie-Hellman, continued). The property that the protocol P in Example 1 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, i.e. whether $\mathcal{C}_{DH} = \langle \epsilon,\ S\rangle \models \mathit{Secret}(d)$, with $\mathit{Secret}(d) = \perp \hookleftarrow \mathsf{g}\langle d\rangle$ and ϵ being the empty trace.

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 as a single judgement $\zeta_0 \downarrow_\theta \eta_0$, for appropriate ζ_0, θ, η_0. The symbolic evaluation relation $\downarrow_s$ of $\mathcal{F}_{DH}$ is presented in Table 2: it is defined as the reflexive and transitive closure of the relation $\overset{\theta}{\rightsquigarrow}_s$.

Table 2. Symbolic Evaluation Relation ($\downarrow_s$) for $\mathcal{F}_{DH}$

($\textsc{Dec}_\mathrm{s}$)	$\mathsf{dec}_\eta(\zeta) \overset{\theta}{\rightsquigarrow}_s x_1\,\theta$			$\theta = \mathrm{mgu}(\zeta = \{x_1\}_{x_2}, \eta = x_2)$
($\textsc{Mult}_\mathrm{s}$)	$\mathsf{mult}(\zeta_1, \zeta_n) \overset{\theta}{\rightsquigarrow}_s (x_{i_1} \times \cdots \times x_{i_n})\,\theta$			$\begin{cases} 1 \le k < n \le l, \\ \theta = \mathrm{mgu}(\zeta_1 = x_1 \times \cdots \times x_k, \\ \zeta_2 = x_{k+1} \times \cdots \times x_n) \end{cases}$
($\textsc{Inv1}_\mathrm{s}$)	$\mathsf{inv}(\zeta) \overset{\theta}{\rightsquigarrow}_s (\mathsf{inv}'(x_{i_1}) \times \cdots \times \mathsf{inv}'(x_{i_n}))\,\theta$			$\begin{cases} 1 \le n \le l, \\ \theta = \mathrm{mgu}(\zeta = x_1 \times \cdots \times x_n) \end{cases}$
($\textsc{Inv2}_\mathrm{s}$)	$\mathsf{inv}'(\zeta) \overset{\theta}{\rightsquigarrow}_s x_1\,\theta$			$\theta = \mathrm{mgu}(\zeta, {x_1}^{-1})$
($\textsc{Inv3}_\mathrm{s}$)	$\mathsf{inv}'(\zeta) \overset{\epsilon}{\rightsquigarrow}_s \zeta^{-1}$	($\textsc{Inv4}_\mathrm{s}$)	$\mathsf{inv}'(\zeta) \times \eta \overset{\theta}{\rightsquigarrow}_s \mathsf{unit}$	$\theta = \mathrm{mgu}(\zeta, \eta)$
($\textsc{Unit1}_\mathrm{s}$)	$\mathsf{unit} \times \zeta \overset{\epsilon}{\rightsquigarrow}_s \zeta$	($\textsc{Unit2}_\mathrm{s}$)	$\mathsf{unit} \overset{\epsilon}{\rightsquigarrow}_s 1$	
($\textsc{Exp1}_\mathrm{s}$)	$\mathsf{exp}\,(x, \zeta) \overset{\theta}{\rightsquigarrow}_s \mathsf{exp}\,(\alpha, \mathsf{mult}(x_1, \zeta))$			$\theta = [\mathsf{exp}\,(\alpha, x_1)/x]$
($\textsc{Exp2}_\mathrm{s}$)	$\mathsf{exp}\,(\mathsf{exp}\,(\xi, \eta), \zeta) \overset{\epsilon}{\rightsquigarrow}_s \mathsf{exp}\,(\xi, \mathsf{mult}(\eta, \zeta))$			
($\textsc{Root1}_\mathrm{s}$)	$\mathsf{root}\,(x, \zeta) \overset{\theta}{\rightsquigarrow}_s \mathsf{exp}\,(\alpha, \mathsf{mult}(x_1, \mathsf{inv}(\zeta)))$			$\theta = [\mathsf{exp}\,(\alpha, x_1)/x]$
($\textsc{Root2}_\mathrm{s}$)	$\mathsf{root}\,(\mathsf{exp}\,(\xi, \eta), \zeta) \overset{\epsilon}{\rightsquigarrow}_s \mathsf{exp}\,(\xi, \mathsf{mult}(\eta, \mathsf{inv}(\zeta)))$			
($\textsc{Ctx}_\mathrm{s}$)	$\dfrac{\zeta \overset{\theta}{\rightsquigarrow}_s \zeta'}{C[\zeta] \overset{\theta}{\rightsquigarrow}_s C\theta[\zeta']}$	$\textsc{Symbolic Eval.}$:	$\zeta \downarrow_\theta \eta$ 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 chosen fresh according to some arbitrary but fixed rule.

Lemma 1. *$\overset{\theta}{\rightsquigarrow}_s$ is image-finite and terminating. Hence, $\downarrow_s$ is image-finite.*

Definition 5 (symbolic traces and configurations). *A* symbolic trace *is a string $s \in Act^*$ s.t.: (a) $\mathrm{en}(s) = \emptyset$, and (b) for each s_1, s_2, α and x, if $s = s_1 \cdot \alpha \cdot s_2$ 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 given in terms of a symbolic transition relation $\longrightarrow_\mathrm{s}$ which is standard (see [5]), once the symbolic evaluation relation ($\downarrow_s$) has been defined. Here we simply present the symbolic versions of the input and output rules for comparison with their concrete counterparts:

$$\begin{array}{ll} (\textsc{Inp}_{\mathsf{s}})\ \langle \sigma,\ \mathsf{a}(x).\,A\rangle_{\mathsf{s}} \longrightarrow_{\mathsf{s}} \langle \sigma \cdot \mathsf{a}\langle x\rangle,\ A\rangle_{\mathsf{s}} & \\ (\textsc{Out}_{\mathsf{s}})\ \langle \sigma,\ \overline{\mathsf{a}}\langle \zeta\rangle.\,A\rangle_{\mathsf{s}} \longrightarrow_{\mathsf{s}} \langle \sigma\theta \cdot \overline{\mathsf{a}}\langle M\rangle,\ A\theta\rangle_{\mathsf{s}} & \zeta \downarrow_{\theta} M \end{array}$$

In rule $(\textsc{Inp}_{\mathsf{s}})$, input variables are *not* instantiated immediately. 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 rule $(\textsc{Out}_{\mathsf{s}})$. For 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., $\langle \epsilon,\ P\rangle_{\mathsf{s}} \longrightarrow_{\mathsf{s}}^{*} \langle \sigma\theta,\ P'\theta\theta'\rangle_{\mathsf{s}}$, with $\sigma = \overline{\mathsf{a}}\langle k\rangle \cdot \mathsf{a}\langle x\rangle$.

Whenever $\langle \sigma,\ A\rangle_{\mathsf{s}} \longrightarrow_{\mathsf{s}}^{*} \langle \sigma',\ A'\rangle_{\mathsf{s}}$ for some A', we say that $\langle \sigma,\ A\rangle_{\mathsf{s}}$ *symbolically generates* σ', and write $\langle \sigma,\ A\rangle_{\mathsf{s}} \searrow_{\mathsf{s}} \sigma'$. The relation $\longrightarrow_{\mathsf{s}}$ is finitely-branching since $\downarrow_s$ is. Hence, 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, let $P = \mathsf{a}(y).\,\mathsf{let}\, x = \mathsf{dec}_k(y)\,\mathsf{in}\,\overline{\mathsf{a}}\langle x\rangle.\,\mathbf{0}$. The initial configuration $\langle \epsilon,\ P\rangle_{\mathsf{s}}$ symbolically generates $\mathsf{a}\langle \{x_0\}_k\rangle \cdot \overline{\mathsf{a}}\langle x_0\rangle$, which is inconsistent, as the environment cannot generate the value k in $\{x_0\}_k$ (i.e. $\epsilon \not\vdash k$). 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.

The next theorem 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. We denote by $\sigma\backslash\hat{x}$ the longest prefix of σ not containing $\hat{x}$.

Definition 6 (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 $\sigma\rho$ is a* solution *of σ and σ is* consistent.

Theorem 1 (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 $\mathcal{C} \searrow_{\mathsf{s}} \sigma$ 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. Remark that a symbolic trace σ needs not have solutions (ground instances that are traces). The next result allows us to check consistency.

Proposition 1. *Let σ be a symbolic trace. Then there exists a finite set of traces* **Refinement**(σ)*, 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 $\sigma' \in$* **Refinement**(σ)*.*

Proposition 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 [5] for details; here we just give an example. Let $\sigma' = \bar{\mathsf{c}}\langle h\rangle \cdot \mathsf{c}\langle x\rangle \cdot \bar{\mathsf{c}}\langle \mathsf{exp}\,(\alpha, k \times h)\rangle \cdot \bar{\mathsf{c}}\langle \{m\}_{\mathsf{exp}\,(\alpha, k\times x)}\rangle \cdot \mathsf{c}\langle m\rangle$. 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 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 $\mathsf{exp}\,(\mathsf{root}\,(\mathsf{exp}\,(\alpha, k \times h), h), \hat{x})$.

The verification method $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ below 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 $\mathcal{C}$.

```
   $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$
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 $\gamma$ in $\sigma$ do
4.                  if  $\exists\,\theta = \mathrm{mgu}(\beta, \gamma)$  and  $\exists\,\sigma' \in \mathbf{Refinement}(\sigma\theta)$  where
5.                      $\sigma' = \sigma\theta\theta'$ and  $\alpha\theta\theta'$ does not occur prior to $\beta\theta\theta'$ in $\sigma'$
6.                  then return(No, $\sigma'$);
7. return(Yes);
```

To understand how the method works, 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 Theorem 1, this amounts to checking that for each σ symbolically generated by $\mathcal{C}$, no solution of σ contains an instance of β. First, one checks whether there is a mgu θ of γ and β, for every γ in σ. If, for every σ, such a θ does not exist or if it exists but $\sigma\theta$ is not consistent (check at step 4), then the property holds true, otherwise it does not, and the trace σ' violating the property is reported.

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

The method has been applied to analyse the Diffie-Hellman protocol and it has detected the usual man-in-the-middle attack (see [6]).

5 Conclusions and future work

We have presented a model and a method for the analysis of protocols built around shared-key encryption and modular exponentiation. We are confident that our approach smoothly carries 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 our technique into the STA analysis tool ([4]).

Our technical development has been confined to multiplication and exponentiation, but the methodology presented suggests directions for extensions to other low-level primitives.

Acknowledgements. We thank the anonymous referees for helpful comments.

References

1. M. Abadi, C. Fournet. Mobile Values, New Names, and Secure Communication. In *Conf. Rec. of POPL'01*, 2001.
2. 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.
3. M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proc. of ICALP'01*, LNCS 2076, Springer-Verlag, 2001.
4. M. Boreale, M. Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Security Protocols. In *Proc. of SAC'02*, ACM Press, 2002.
5. M. Boreale and M. Buscemi. A Framework for the Analysis of Security Protocol. In *Proc. of CONCUR '02*, LNCS 2421. Springer-Verlag, 2002.
6. M. Boreale and M. Buscemi. On the Symbolic Analysis of Low-Level Cryptographic Primitives: Modular Exponentiation and the Diffie-Hellman Protocol. To appear in *Proc. of FCS'03*, 2003.
7. 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.
8. 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.
9. 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.
10. W. Diffie, M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, 1976.
11. D. Dolev, A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29(2):198-208, 1983.
12. 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.
13. G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proc. of TACAS'96*, LNCS 1055, Springer-Verlag, 1996.
14. 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.
15. 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.
16. V. Vanackère. The TRUST Protocol Analyser, Automatic and Efficient Verification of Cryptographic Protocols. In *Proc. of Verify '02*, 2002.