Introducing Commutative and Associative Operators in Cryptographic Protocol Analysis*

Ivan Cibario Bertolotti[1], Luca Durante[1], Riccardo Sisto[2], and Adriano Valenzano[1]

[1] IEIIT – CNR
[2] Dipartimento di Automatica e Informatica, Politecnico di Torino, C.so Duca degli Abruzzi 24, I-10129 Torino, Italy,
{luca.durante,ivan.cibrario,riccardo.sisto,adriano.valenzano}@polito.it

Abstract. Many formal techniques for the verification of cryptographic protocols rely on the abstract definition of cryptographic primitives, such as shared, private, and public key encryption. This approach prevents the analysis of those protocols that explicitly use commutative and associative algebraic operators to build their messages such as, for example, the Diffie-Hellman key-exchange protocol. This paper investigates the possibility of handling operators which exhibit special properties by considering a stand-alone extension to the way most known popular techniques handle messages exchanged during the protocol sessions. Such an extension makes the new operators tractable by automatic model checking techniques. The properties examined in this paper are commutativity and associativity.

1 Introduction

The formal verification of cryptographic protocols is being extensively studied by many researchers, in view of the ever increasing importance and popularity of secure, distributed applications. Most commonly adopted techniques, such as proof techniques [3, 19, 21] and state exploration [7, 11, 17] rely on abstract concepts of encryption and decryption. For instance, in a shared key cryptosystem, it is usually not allowed that encryption and decryption keys, computed autonomously by different agents, still represent the same key although they have different syntactical forms, as is the case of the Diffie-Hellman key exchange protocol [9]. Similar issues arise when using primitives like `xor` or exponentiation modulo different moduli as in the RSA cryptosystem.

Such a limitation has the invaluable advantage that any syntactical difference between messages implies a semantical difference, i.e. there are not two or more different representations of the same message, but also hampers the ability to describe and formally verify any real-world cryptographic protocol that makes

* This work was partially supported by the Italian National Research Council, grant number CNRC00FE45, and by the Center of Excellence on Multimedia Radiocommunications (CERCOM).

use of more sophisticated operators, as pointed out in [2, 6, 12–14, 18, 20]; moreover, during peer review of this paper, the authors got to know that this problem was being addressed by other researchers, too [4, 8, 15, 16].

In this paper we present the theoretical framework needed to extend the intruder's knowledge representation strategy presented in [5], to include associative and commutative operators. As most other researchers do, that method relies on the *perfect encryption* assumptions, and on the so-called *Dolev-Yao intruder's model*, inspired by [10]; in addition, it guarantees that:

- encryption and decryption keys are not restricted to be atomic,
- public/private key cryptosystems, and related operators, are fully supported,
- the intruder's knowledge is always kept in a minimised form.

In particular, we show that the intruder's knowledge representation, as discussed in [5], can be updated to take into account the new operators by defining a canonical form for messages, i.e. any message built by means of the extended set of operators can always be syntactically transformed into its canonical form, preserving its semantic properties, so that the intruder's knowledge representation can be based only on messages in such a canonical form. In fact the proposed extension preserves uniqueness, canonicity and minimality of the intruder's knowledge representation.

Section 2 recalls the term algebra of [5], while the formalism and the main results of [5], concerning the intruder's knowledge representation, are briefly summarised in Sect. 3. Section 4 deals with commutative and associative operators, and the computation of the canonical form of messages containing them, while Sect. 5 shows how the intruder's knowledge representation can be made able to handle these operators. Finally, Sect. 6 discusses the Diffie-Hellman key exchange protocol [9] in the light of the approach presented in Sections 4 and 5, and Sect. 7 draws some conclusions.

2 Basic Term Algebra

Many formal methods used to describe cryptographic protocols rely on a *term algebra* for the description of messages. In the spi calculus term grammar adopted in [5], terms can be either atomic elements, i.e. names, including the special name 0 representing the integer constant zero, or compound terms built using the term composition operators listed in Table 1. Names may represent communication channels, atomic keys and key pairs, nonces (also called *fresh names*) and any other unstructured data. If a term σ occurs as a sub-expression of another term ρ, then σ is called a *sub-term* of ρ; moreover, any term σ is a sub-term of itself.

The informal meaning of the composition operators is as follows:

- (σ, ρ) is the *pairing* of σ and ρ. It is a compound term whose components are σ and ρ. Pairs can always be freely split into their components.
- $\mathrm{suc}(\sigma)$ is the *successor* of σ. This operator can also be used, more generally, as the abstract representation of an invertible function on terms.

Table 1. Basic term syntax

σ, ρ	terms	$\mathrm{H}(\sigma)$	hashing
a	name	$\{\sigma\}_\rho$	shared-key encryption
(σ, ρ)	pair	$\sigma^+,\ \sigma^-$	public/private part
0	zero	$\{[\sigma]\}_\rho$	public-key encryption
$\mathrm{suc}(\sigma)$	successor	$[\{\sigma\}]_\rho$	private-key signature
x	variable		

- $\mathrm{H}(\sigma)$ is the *hashing* of σ. $\mathrm{H}(\sigma)$ represents a function of σ that cannot be inverted.
- $\{\sigma\}_\rho$ is the ciphertext obtained by encrypting σ under key ρ using a shared-key cryptosystem.
- σ^+ and σ^- represent respectively the public and private half of a key pair σ. σ^+ cannot be deduced from σ^- and vice versa.
- $\{[\sigma]\}_\rho$ is the result of the public-key encryption of σ with ρ.
- $[\{\sigma\}]_\rho$ is the result of the signature (private key encryption) of σ with ρ.

Even though this algebra comes directly from spi calculus [1], it is general enough to cope with the needs of many formal techniques used to model cryptographic protocols.

3 Basic Intruder's Knowledge Representation

In this section we recall some notation and results from [5], useful to understand how the intruder's knowledge representation can be extended in order to manage also terms containing operators with special properties; it must be noted that here we do not take into account public and private key encryption, which is discussed in depth in [5].

Let $\mathcal{A}$ be the set of names, including the integer constant 0, and $\mathcal{T}$ the set of terms that can be built by combining the elements of $\mathcal{A}$ by means of the operators defined in Table 1.

The closure of a set of terms $\Sigma \subseteq \mathcal{T}$ is denoted $\widehat{\Sigma}$ and is defined as the set of all terms that can be built by combining the elements of Σ by means of the operators defined in Table 1 and their inverses. Formally, $\widehat{\Sigma}$ is the least set of terms such that, for each σ, σ_1 and $\sigma_2 \in \mathcal{T}$, the following closure rules hold:

$$\sigma \in \Sigma \Rightarrow \sigma \in \widehat{\Sigma} \tag{1}$$

$$\sigma \in \widehat{\Sigma} \Rightarrow \mathrm{suc}(\sigma) \in \widehat{\Sigma} \quad \text{(successor)} \tag{2}$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \Rightarrow (\sigma_1,\ \sigma_2) \in \widehat{\Sigma} \quad \text{(pairing)} \tag{3}$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \Rightarrow \{\sigma_1\}_{\sigma_2} \in \widehat{\Sigma} \quad \text{(sh. key encryption)} \tag{4}$$

$$\sigma \in \widehat{\Sigma} \Rightarrow \mathrm{H}(\sigma) \in \widehat{\Sigma} \quad \text{(hashing)} \tag{5}$$

$$\mathrm{suc}(\sigma) \in \widehat{\Sigma} \Rightarrow \sigma \in \widehat{\Sigma} \quad \text{(prec)} \tag{6}$$

$$(\sigma_1,\ \sigma_2) \in \widehat{\Sigma} \Rightarrow \sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \quad \text{(projection)} \tag{7}$$

$$\{\sigma_1\}_{\sigma_2} \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \quad \Rightarrow \qquad \sigma_1 \in \widehat{\Sigma} \qquad \text{(sh. key decryption)} \qquad (8)$$

In principle, if Σ is the set of messages the intruder has intercepted so far, $\widehat{\Sigma}$ is the set of all messages the intruder can generate at a given point. Since neither $\mathcal{A}$ nor $\mathcal{T}$ contain variables, the intruder's knowledge does not contain variables, too.

We say that a set of terms is *finite* if it contains a finite number of finite length elements. Given a finite set of terms Σ, we define the *minimal closure seed of* Σ, and denote it as $\overline{\Sigma}$, the subset of $\widehat{\Sigma}$ that satisfies the following predicates for each $a \in \mathcal{A}$, and for each σ, σ_1, $\sigma_2 \in \mathcal{T}$:

$$a \in \overline{\Sigma} \Leftrightarrow a \in \widehat{\Sigma} \qquad (9) \qquad\qquad \text{suc}(\sigma) \notin \overline{\Sigma} \qquad (12)$$

$$\{\sigma_1\}_{\sigma_2} \in \overline{\Sigma} \Leftrightarrow \sigma_2 \notin \widehat{\Sigma} \qquad (10) \qquad\qquad (\sigma_1, \sigma_2) \notin \overline{\Sigma} \qquad (13)$$

$$\text{H}(\sigma) \in \overline{\Sigma} \Leftrightarrow \sigma \notin \widehat{\Sigma} \qquad (11)$$

Before discussing the basic properties of $\overline{\Sigma}$, let us preliminarily define $\texttt{r}(\sigma, \Sigma)$ as the boolean value obtained by executing the following algorithm:

```
boolean r($\sigma$, Σ) {
  if $\sigma \in \Sigma$                  then return TRUE;
  else if $\sigma = \texttt{suc}(\sigma_1)$     then return r($\sigma_1$, Σ);
  else if $\sigma = (\sigma_1,\ \sigma_2)$    then return r($\sigma_1$, Σ) $\wedge$ r($\sigma_2$, Σ);
  else if $\sigma = \{\sigma_1\}_{\sigma_2}$      then return r($\sigma_1$, Σ) $\wedge$ r($\sigma_2$, Σ);
  else if $\sigma = \texttt{H}(\sigma_1)$       then return r($\sigma_1$, Σ);
  else ($\sigma \in \mathcal{A} \setminus$ Σ)       return FALSE;
}
```

Informally, this algorithm recursively checks whether σ can be deduced from the set Σ using rules (2)-(5) only, i.e. by using rules that build a new term by introducing an operator between two or more simpler terms. The basic properties of $\overline{\Sigma}$ are then expressed by the following theorems:

Theorem 1. *(Finiteness) For each finite set of terms $\Sigma \subseteq \mathcal{T}$, $\overline{\Sigma}$ is finite.*

Theorem 2. *(Minimality) Let $\Sigma \subseteq \mathcal{T}$ be a finite set of terms, and $\sigma \in \overline{\Sigma}$. Then $\widehat{(\overline{\Sigma} \setminus \{\sigma\})} \subset \widehat{\overline{\Sigma}}$.*

Theorem 3. *(Decidability) Let $\sigma \in \mathcal{T}$ be any finite term, $\Sigma \subseteq \mathcal{T}$ be a finite set of terms, and let us assume that $\overline{\Sigma}$ is known. Then, determining if $\sigma \in \widehat{\Sigma}$ is decidable.*

Let us now define a *reduction rule* as a triple $R = \langle \Sigma_I,\ C,\ \Sigma_O \rangle$, where Σ_I and Σ_O are sets of terms representing respectively premises and conclusions of closure rule C. Applying a reduction step R to a finite set of terms Σ means eliminating the premises from and adding the conclusions to Σ. This is written $\Sigma \xrightarrow{R} \Sigma'$, where $\Sigma' = (\Sigma \setminus \Sigma_I) \cup \Sigma_O$ is the resulting set.

Given a finite set of terms Σ, a *reduction of Σ* is a finite sequence of application of reduction rules R_i to finite sets of terms Σ_i, denoted:

$$\Sigma_0 \xrightarrow{R_0} \Sigma_1 \xrightarrow{R_1} \Sigma_2 \cdots \Sigma_{k-1} \xrightarrow{R_{k-1}} \Sigma_k \enspace ,$$

such that $\Sigma_0 = \Sigma$ and $R_i \in \mathcal{R}(\Sigma_i)$, where $\mathcal{R}(\Sigma_i)$ is the set of reduction rules whose pre-conditions are satisfied by Σ_i. Below, the notation $a \to b$ means that if the pre-condition a is true, then the reduction rule b can be applied in Σ_i, that is, $b \in \mathcal{R}(\Sigma_i)$. The set $\mathcal{R}(\Sigma_i)$ is the least set such that the following relations hold:

$$\mathrm{H}(\sigma) \in \Sigma_i \ \wedge \ \mathtt{r}(\sigma, \Sigma_i) \to \langle \{\mathrm{H}(\sigma)\}, (5), \emptyset \rangle \tag{14}$$

$$\mathrm{suc}(\sigma) \in \Sigma_i \to \langle \{\mathrm{suc}(\sigma)\}, (6), \{\sigma\} \rangle \tag{15}$$

$$(\sigma_1, \sigma_2) \in \Sigma_i \to \langle \{(\sigma_1, \sigma_2)\}, (7), \{\sigma_1, \sigma_2\} \rangle \tag{16}$$

$$\{\sigma_1\}_{\sigma_2} \in \Sigma_i \ \wedge \ \mathtt{r}(\sigma_2, \Sigma_i) \to \langle \{\{\sigma_1\}_{\sigma_2}\}, (8), \{\sigma_1\} \rangle \tag{17}$$

The definition of reduction allows to claim:

Proposition 1. *Given a finite set of terms Σ, there exists a finite reduction of Σ:*

$$\Sigma = \Sigma_0 \xrightarrow{R_0} \Sigma_1 \cdots \Sigma_{k-1} \xrightarrow{R_{k-1}} \Sigma_k \enspace ,$$

such that $\Sigma_k = \overline{\Sigma}$, and $\widehat{\Sigma_i} = \widehat{\Sigma} \ \forall \ i \in [0, k]$.

Starting with proposition 1 the following results hold:

Theorem 4. *(Closure preservation) For each finite set of terms $\Sigma \subseteq \mathcal{T}$, $\widehat{\overline{\Sigma}} = \widehat{\Sigma}$.*

Theorem 5. *(Computability) For each finite set of terms $\Sigma \subseteq \mathcal{T}$, $\overline{\Sigma}$ can be computed in a finite number of steps.*

Since $\overline{\Sigma}$ can be computed from Σ, theorem 3 can now be used to check whether $\sigma \in \widehat{\Sigma}$.

Theorems 1, 2 and 4 state that the closure seed representation of a finite set of terms Σ is finite and is the minimal set of terms having the same closure of Σ, where minimality means that any element cannot be built from the other ones by means of term composition operators only, i.e. there are no redundant elements.

Theorems 4 and 5 entail that if a new term ρ is added to a minimal closure seed $\overline{\Sigma}$, e.g. as a consequence of an output action, the new minimal closure seed $\overline{\overline{\Sigma} \cup \{\rho\}}$ can be incrementally computed by a reduction that starts from $\overline{\Sigma} \cup \{\rho\}$.

Formal proofs of results depicted above are found in [5].

4 The Canonical Term Representation

The introduction of operators with special properties has several subtle implications in the intruder's knowledge representation, where the necessary and sufficient condition for the equivalence of two terms simply relies on their syntactic identity. In fact, when a term algebra is extended with commutative and/or associative operators, two or more syntactic representations of the same term are allowed. For example, if $\odot$ is a commutative, binary operator, $\overline{\Sigma_1} = \{\sigma \odot \rho\}$ and $\overline{\Sigma_2} = \{\rho \odot \sigma\}$ express exactly the same knowledge and are both minimum and canonical according to the original definition of $\overline{\Sigma}$ presented in [5], but clearly they are *not* equal in a syntactical sense.

In order to correctly handle such operators, we introduce the notion of *canonical term representation* and we assume that all algorithms that act on terms, such as Σ minimisation and $\mathtt{r}(\cdot, \cdot)$ always act on canonical terms.

The canonical term representation leverages on the concept of *term equivalence class* induced by the operators' properties: two terms belong to the same equivalence class, induced by a given property, iff they can be made equal by applying that property. For example $\mathrm{H}(a \odot b)$ and $\mathrm{H}(b \odot a)$ are in the same equivalence class if $\odot$ is a commutative operator, because they can be made equal by applying the commutative property.

The canonical term representation has the important role of selecting, for each term equivalence class induced by the operators' properties, a unique element that will represent the class as a whole in all contexts. It can be proved, with ordinary effort, that this additional requirement is powerful enough to restore the uniqueness of $\overline{\Sigma}$ for all operators' properties considered in this paper.

From now on it is assumed that two terms that can be obtained from eachother by a suitable application of one or more operators' properties are indistinguishable from the intruder's point of view. For this reason, any additional observation the intruder can make on term generation besides the final value, such as timing or accuracy, is neglected.

4.1 Notational Conventions

Table 2 introduces the main notational conventions adopted in the following. Note also that:

- When multiple atoms, terms, and operators are needed in the same context, a unique, numeric subscript is used to distinguish them. For example, a_1 and a_2 are two distinct, and possibly different, atoms.
- When appropriate, the arity of the operator is explicitly denoted with a subscript, for example: $\odot_n$ is an n-ary operator. When more than one operator is needed in the same context, the *first* subscript singles out the operator, and the *second* one gives its arity. For example, $\odot_{1n}$ and $\odot_{2m}$ are two distinct, and possibly different operators; the first one has arity n, the second has arity m.
- This paper uses the infix and prefix operator notation interchangeably, that is, $a \odot b = \odot(a, b)$.

4.2 Total Order Relation on Terms

We assume that there is a total order relation $\leq_{\mathcal{P}}$ on the elements of $\mathcal{P}$, as it is the case when $\mathcal{P}$ is finite or is a countable infinity; similarly, we assume that

Table 2. Notational conventions

Symbol	Meaning
$\mathcal{P}$	is the set of term algebra operators; it contains both the standard term algebra operators of Table 1, and the additional operators described in Sections 4, 5 and 6.
$\mathcal{T}$	is the set of all terms that can be built by combining the elements of $\mathcal{A}$ with operators of $\mathcal{P}$.
a	is a generic, atomic term: $a \in \mathcal{A}$.
$\odot$	is a generic n-ary operator: $\odot \in \mathcal{P}$.
σ	is a generic term, either atomic or non-atomic: $\sigma \in \mathcal{T}$.
$\leq_{\mathcal{A}}$	is a total order relation on $\mathcal{A}$.
$\leq_{\mathcal{P}}$	is a total order relation on $\mathcal{P}$.
$\leq_{\mathcal{T}}$	is a total order relation on $\mathcal{T}$.

there is a total order relation $\leq_{\mathcal{A}}$ on the elements of $\mathcal{A}$, as it is the case when the total number of atomic terms ever used in a session of the protocol is finite or is a countable infinity.

Then, we are able to define a relation $\leq_{\mathcal{T}}$ on the elements of $\mathcal{T}$, by means of the following implications:

$$\sigma_1 \in \mathcal{A} \ \wedge \ \sigma_2 \in \mathcal{A} \ \wedge \ \sigma_1 \leq_{\mathcal{A}} \sigma_2 \Rightarrow \sigma_1 \leq_{\mathcal{T}} \sigma_2 \tag{18}$$

$$\sigma_1 \in \mathcal{A} \ \wedge \ \sigma_2 \notin \mathcal{A} \Rightarrow \sigma_1 \leq_{\mathcal{T}} \sigma_2 \tag{19}$$

$$\sigma_1 = \odot_{1n}(\ldots) \ \wedge \ \sigma_2 = \odot_{2m}(\ldots) \ \wedge \ n < m \Rightarrow \sigma_1 \leq_{\mathcal{T}} \sigma_2 \tag{20}$$

$$\begin{aligned} &\sigma_1 = \odot_{1n}(\sigma_{11} \ldots \sigma_{1n}) \ \wedge \ \sigma_2 = \odot_{2n}(\sigma_{21} \ldots \sigma_{2n}) \\ \wedge \ \exists j \mid \sigma_{1j} \neq \sigma_{2j} \ \wedge \ &\sigma_{1j} \leq_{\mathcal{T}} \sigma_{2j} \ \wedge \ \sigma_{1i} = \sigma_{2i} \ \ \forall i < j \Rightarrow \sigma_1 \leq_{\mathcal{T}} \sigma_2 \end{aligned} \tag{21}$$

$$\begin{aligned} &\sigma_1 = \odot_{1n}(\sigma_{11} \ldots \sigma_{1n}) \ \wedge \ \sigma_2 = \odot_{2n}(\sigma_{21} \ldots \sigma_{2n}) \\ &\wedge \ \sigma_{1i} = \sigma_{2i} \ \ \forall i \ \wedge \ \ \odot_{1n} \leq_{\mathcal{P}} \odot_{2n} \Rightarrow \sigma_1 \leq_{\mathcal{T}} \sigma_2 \end{aligned} \tag{22}$$

It is worth noting that implications (21) and (22) assume that the operands $\sigma_{11} \ldots \sigma_{1n}$ and $\sigma_{21} \ldots \sigma_{2n}$ of $\odot_{1n}$ and $\odot_{2n}$, respectively, are already in canonical form. In turn, the algorithm to compute the canonical representation of a term, given in Sect. 4.3, will make use of $\leq_{\mathcal{T}}$. Correctness and computability are guaranteed, because:

- The canonical representation algorithm always invokes $\leq_{\mathcal{T}}$ on terms that are already canonical *(correctness)*.
- To compute the canonical representation mentioned above, the canonical representation algorithm is always invoked recursively on sub-terms that have one operator less than their parent, and it is trivially computable on atoms *(computability by induction)*.

Theorem 6. *The relation $\leq_{\mathcal{T}}$ is a total order relation on $\mathcal{T}$.*

The proof of this theorem is based on proving that $\leq_{\mathcal{T}}$ is reflexive, antisymmetric, transitive, and that:

$$\forall \sigma_1, \sigma_2 \in \mathcal{T}, \sigma_1 \neq \sigma_2 \quad \sigma_1 \leq_{\mathcal{T}} \sigma_2 \ \vee \ \sigma_2 \leq_{\mathcal{T}} \sigma_1 \ .$$

4.3 Term Canonicalisation

Term canonicalisation has the purpose of determining a unique, canonical form for each $\sigma \in \mathcal{T}$; the canonical form is used when inserting a term into the intruder's knowledge, while checking whether the intruder is able to synthesise a term from a given knowledge.

Term canonicalisation selects one representative element from each equivalence class induced on $\mathcal{T}$ by operator properties according to the following informal rules:

- The canonical form of an atom is the atom itself (rule 23 below).
- The canonical form of the invocation of an operator without special properties is the invocation of the same operator on the same operands, put into canonical form (rule 24).
- The canonical form of the invocation of a commutative operator $\odot$ on a list of operands $\sigma_1 \dots \sigma_n$ is the invocation of the same operator on the operands put into canonical form and then sorted according to $\leq_{\mathcal{T}}$, to ensure the uniqueness of representation (rule 25).
- The canonical form of the invocation of an associative operator $\odot$ on a list of operands $\sigma_1 \dots \sigma_n$ is the invocation of the same operator on the operands taken into canonical form. If, after canonicalisation, some operands have the same operator $\odot$ as their top-level operator, the hierarchy of invocations of $\odot$ is *flattened* (rule 26).

Formally, the term canonicalisation function is the transitive closure $\mathcal{C}^*$ of function $\mathcal{C} : \mathcal{T} \to \mathcal{T}$, which can be defined according to the following set of rewrite rules, where $\mathcal{P}_C$ is the set of commutative operators, and $\mathcal{P}_A$ is the set of associative operators. The intersection of these sets may not be empty because operators may have multiple properties.

In the following, a rewrite rule like $\frac{\pi}{\sigma \overset{\mathcal{C}}{\longmapsto} \sigma'}$ should be read as: when the prerequisite predicate π is true, then term σ can be rewritten as σ' through a canonicalisation step. In σ', the notation $\mathcal{C}^*(\sigma_i)$ entails the recursive application of the canonicalisation rewrite rules to sub-term σ_i.

$$\frac{a \in \mathcal{A}}{a \overset{\mathcal{C}}{\longmapsto} a} \tag{23}$$

$$\frac{\odot \notin (\mathcal{P}_C \cup \mathcal{P}_A)}{\odot(\sigma_1 \dots \sigma_n) \overset{\mathcal{C}}{\longmapsto} \odot(\mathcal{C}^*(\sigma_1) \dots \mathcal{C}^*(\sigma_n))} \tag{24}$$

$$\frac{\begin{array}{c}\odot \in \mathcal{P}_C \ \wedge \ (\exists i \mid \sigma_i \neq \mathcal{C}^*(\sigma_i) \ \vee \ \exists i, j \mid \mathcal{C}^*(\sigma_i) \not\leq_{\mathcal{T}} \mathcal{C}^*(\sigma_j)) \ \wedge \\ \exists k_1 \dots k_n, k_i \neq k_l \ \ \forall i, l \mid \mathcal{C}^*(\sigma_{k_i}) \leq_{\mathcal{T}} \mathcal{C}^*(\sigma_{k_{i+1}}), i = 1 \dots n-1\end{array}}{\odot(\sigma_1 \dots \sigma_n) \overset{\mathcal{C}}{\longmapsto} \odot(\mathcal{C}^*(\sigma_{k_1}) \dots \mathcal{C}^*(\sigma_{k_n}))} \tag{25}$$

$$\frac{\odot \in \mathcal{P}_A \ \wedge \ \exists i \mid \mathcal{C}^*(\sigma_i) = \odot(\sigma_{i1} \dots \sigma_{im})}{\odot(\sigma_1 \dots \sigma_n) \overset{\mathcal{C}}{\longmapsto} \odot(\mathcal{C}^*(\sigma_1) \dots \mathcal{C}^*(\sigma_{i-1}) \mathcal{C}^*(\sigma_{i1}) \dots \mathcal{C}^*(\sigma_{im}) \mathcal{C}^*(\sigma_{i+1}) \dots \mathcal{C}^*(\sigma_n))} \tag{26}$$

Moreover, it is assumed that, on each invocation of the $\mathcal{C}^*$ function on term σ, *all* rewrite rules whose prerequisite are satisfied on σ are applied in sequence, until the set of applicable rules is exhausted, or the only applicable rule is either the atom-preserving rule (23), or the trivial canonicalisation rule (24) for operators with no special properties. The order of rule application does not matter, because any order leads to the same canonical term.

Theorem 7. *The function $\mathcal{C}^* : \mathcal{T} \to \mathcal{T}$ is computable for any finite-length $\sigma \in \mathcal{T}$.*

The proof is based on the observation that, given a term σ, only a finite number of rewrite rules in the form (25-26) are applicable to each sub-term of σ, and no sequence of rule applications on a term can lead to the term itself. In addition, rules (23-24) can be applied at most once to each atom of σ and to each sub-term of σ, respectively, but those sub-terms are in finite number because σ is finite.

5 Extended Intruder's Knowledge Representation

The intruder's knowledge closure rules, the predicates defining $\overline{\Sigma}$, the $\mathtt{r}(\cdot,\cdot)$ algorithm, and the $\overline{\Sigma}$ minimisation rules presented in Sect. 3, have to be extended to take into account the operator's properties and to ensure that all properties of $\overline{\Sigma}$ still hold: in fact, it can be proved that all properties mentioned in Sect. 3 (including *incremental computability*) still hold under the extensions proposed here.

5.1 Commutative Operators

Let us assume, without loss of generality, that $\odot$ is a binary, commutative operator. The ability to synthesise a compound term by means of operator $\odot$ can be captured by the following closure rule, to be added to rules (1-8):

$$\sigma_1 \in \widehat{\Sigma} \ \wedge \ \sigma_2 \in \widehat{\Sigma} \ \wedge \ \sigma_1 \leq_{\mathcal{T}} \sigma_2 \ \Rightarrow \ \sigma_1 \odot \sigma_2 \in \widehat{\Sigma} \ . \tag{27}$$

Accordingly, the following, additional $\overline{\Sigma}$ defining predicate must be added to (9-11):

$$\sigma_1 \odot \sigma_2 \in \overline{\Sigma} \ \Leftrightarrow \ (\sigma_1 \notin \widehat{\Sigma} \ \vee \ \sigma_2 \notin \widehat{\Sigma}) \ \wedge \ \sigma_1 \leq_{\mathcal{T}} \sigma_2 \ . \tag{28}$$

Informally, the predicate $\sigma_1 \notin \widehat{\Sigma} \vee \sigma_2 \notin \widehat{\Sigma}$ prevents the presence of compound terms $\sigma_1 \odot \sigma_2$ and $\sigma_2 \odot \sigma_1$ in $\overline{\Sigma}$ if they can be synthesised from their operands; predicate $\sigma_1 \leq_{\mathcal{T}} \sigma_2$ is made true by term canonicalisation, performed before inserting any term in the intruder's knowledge, and effectively prevents the presence of *both* $\sigma_1 \odot \sigma_2$ *and* $\sigma_2 \odot \sigma_1$ in $\overline{\Sigma}$, by explicitly selecting which one has to be preferred.

Similarly, closure rule (27) gives rise to the following additional $\overline{\Sigma}$ reduction rule to be added to (14-17):

$$\sigma_1 \odot \sigma_2 \in \Sigma_i \ \wedge \ \mathtt{r}(\sigma_1, \Sigma_i) \ \wedge \ \mathtt{r}(\sigma_2, \Sigma_i) \ \to \ \langle \{\sigma_1 \odot \sigma_2\}, (27), \emptyset \rangle \tag{29}$$

Rule (29) has a premise term that can be generated using other elements of Σ_i, and simply removes its premises. Therefore, it cannot introduce loops, preserves the closure $\widehat{\Sigma}$ and ensures that Σ_{i+1} only contains canonic terms if the same is true for Σ_i. Closure rule (27) is also the starting point to extend the function $\mathtt{r}(\cdot,\cdot)$ presented in Sect. 3:

```
boolean r($\sigma$, Σ) {
  ...
  else if $\sigma = \odot(\sigma_1, \sigma_2) \wedge \odot \in \mathcal{P}_{\mathtt{C}}$ then return r($\sigma_1$, Σ) $\wedge$ r($\sigma_2$, Σ) ;
  ...
}
```

5.2 Associative Operators

Let us assume, without loss of generality, that $\odot \in \mathcal{P}_A$ is a n-ary, associative operator, with $n \geq 2$. In addition, let us assume that the (degenerate) invocation of operator $\odot$ with one operand is the operand itself: $\odot(\sigma) = \sigma$.

The ability to synthesise a compound term by means of operator $\odot$, possibly leveraging the associative property, is captured by the following closure rule:

$$\odot\,(\sigma_{11}\ldots\sigma_{1m}) \in \widehat{\Sigma} \;\wedge\; \odot\,(\sigma_{21}\ldots\sigma_{2n}) \in \widehat{\Sigma} \;\Rightarrow\; \odot(\sigma_{11}\ldots\sigma_{1m}, \sigma_{21}\ldots\sigma_{2n}) \in \widehat{\Sigma} \tag{30}$$

by the additional $\overline{\Sigma}$ defining predicate:

$$\odot\,(\sigma_1\ldots\sigma_n) \in \overline{\Sigma} \;\Leftrightarrow\; \forall i \in [1, n-1] \;\; \odot\,(\sigma_1\ldots\sigma_i) \notin \widehat{\Sigma} \;\vee\; \odot\,(\sigma_{i+1}\ldots\sigma_n) \notin \widehat{\Sigma} \tag{31}$$

by the additional $\overline{\Sigma}$ reduction rule:

$$\left.\begin{array}{ll} & \odot(\sigma_1\ldots\sigma_n) \in \Sigma_i \\ \wedge & \exists i \in [1, n-1] \mid \mathtt{r}(\odot(\sigma_1\ldots\sigma_i), \Sigma_i) \\ \wedge & \mathtt{r}(\odot(\sigma_{i+1}\ldots\sigma_n), \Sigma_i) \end{array}\right\} \rightarrow \langle\{\odot(\sigma_1\ldots\sigma_n)\}, (30), \emptyset\rangle \tag{32}$$

and by the following extension to the $\mathtt{r}(\cdot,\cdot)$ function:

```
boolean r($\sigma$, Σ) {
  ...
  else if $\sigma = \odot(\sigma_1, \ldots \sigma_{\mathtt{n}}) \wedge \odot \in \mathcal{P}_{\mathtt{A}}$
  then return $\exists \mathtt{i} \in [1, \mathtt{n}-1] \mid$ r($\odot(\sigma_1, \ldots \sigma_{\mathtt{i}})$,Σ) $\wedge$ r($\odot(\sigma_{\mathtt{i+1}}, \ldots \sigma_{\mathtt{n}})$,Σ) ;
  ...
}
```

If $\odot$ is *both* associative and commutative, the rules outlined above must be complemented because if the lists: $(\sigma_1\ldots\sigma_i)$ and $(\sigma_{i+1}\ldots\sigma_n)$ are both sorted according to $\leq_{\mathcal{T}}$, this does *not* imply that their concatenation $(\sigma_1\ldots\sigma_i, \sigma_{i+1}\ldots\sigma_n)$ is also sorted according to the same relation.

So, for example, assuming that $\sigma_1 \leq_{\mathcal{T}} \sigma_2 \leq_{\mathcal{T}} \sigma_3$ it can be $\odot(\sigma_1, \sigma_2, \sigma_3) \in \widehat{\Sigma}$ even if $\sigma_1 \notin \widehat{\Sigma} \wedge \odot\,(\sigma_2, \sigma_3) \notin \widehat{\Sigma} \wedge \odot\,(\sigma_1, \sigma_2) \notin \widehat{\Sigma} \wedge \sigma_3 \notin \widehat{\Sigma}$ and closure rule (30) cannot be applied. In fact, this happens when $\sigma_2 \in \widehat{\Sigma} \wedge \odot\,(\sigma_1, \sigma_3) \in \widehat{\Sigma}$.

On the other hand, the same closure rule can introduce non-canonical terms in $\widehat{\Sigma}$, for the same reason. The refined closure rule for a commutative and associative operator therefore is:

$$\begin{aligned} &\odot(\sigma_{11}\ldots\sigma_{1m}) \in \widehat{\Sigma} \ \wedge \ \odot(\sigma_{21}\ldots\sigma_{2n}) \in \widehat{\Sigma} \Rightarrow \\ &\qquad \mathcal{C}^*(\odot(\sigma_{11}\ldots\sigma_{1m}, \sigma_{21}\ldots\sigma_{2n})) \in \widehat{\Sigma} \ , \end{aligned} \tag{33}$$

where the invocation of the canonicalisation operator $\mathcal{C}^*$ avoids non-canonical terms in $\widehat{\Sigma}$.

On the other hand, when defining the $\overline{\Sigma}$ reduction rules on terms in the form $\odot(\sigma_1\ldots\sigma_n)$, all possible partitions of $S = \{\sigma_1\ldots\sigma_n\}$ into two subsets must be considered:

$$\left.\begin{array}{ll} & \odot(\sigma_1\ldots\sigma_n) \in \Sigma_i \\ \wedge & \exists S_1, S_2 \mid S_1 \neq \emptyset \ \wedge \ S_2 \neq \emptyset \ \wedge \ S_1 \cap S_2 = \emptyset \\ \wedge & S_1 \cup S_1 = \{\sigma_1\ldots\sigma_n\} \ \wedge \ \mathtt{r}(\odot(S_1), \Sigma_i) \\ \wedge & \mathtt{r}(\odot(S_2), \Sigma_i) \end{array}\right\} \rightarrow \ \langle\{\odot(\sigma_1\ldots\sigma_n)\}, (33), \emptyset\rangle \tag{34}$$

In the above rule, with a little abuse of notation, we let $\odot(S) = \odot(\sigma_1\ldots\sigma_n)$, where $S = \{\sigma_1\ldots\sigma_n\}$ is a set. In the same manner, the extension to the $\mathtt{r}(\cdot,\cdot)$ function must be:

```
boolean r(σ, Σ) {
  ...
  else if $\sigma = \odot(\sigma_1, \ldots \sigma_n) \ \wedge \ \odot \in \mathcal{P}_A \cap \mathcal{P}_C$
  then return $\exists \ S_1, S_2 \mid S_1 \neq \emptyset \ \wedge \ S_2 \neq \emptyset \ \wedge \ S_1 \cap S_2 = \emptyset \ \wedge$
              $S_1 \cup S_2 = \{\sigma_1, \ldots \sigma_n\} \ \wedge$ r($\odot(S_1)$, Σ) $\wedge$ r($\odot(S_2)$, Σ) ;
  ...
}
```

6 Diffie-Hellman Key Exchange Protocol

In the Diffie-Hellman protocol [9], depicted in Fig. 1, two numbers G and N are publicly agreed on by the communicating principals A and B. A chooses $X = G^{n_1}$ *mod* N for some random n_1 and sends the result to B as message (1). B chooses $Y = G^{n_2}$ *mod* N for some random n_2 and sends the result to A as message (2). A computes $k = Y^{n_1}$ *mod* N and B computes $k = X^{n_2}$ *mod* N. The result of these two calculations is the same and is equal to the new session key k. This provides a means for exchanging keys but gives no guarantees of authenticity.

The last step (3) does not belong to the key-exchange protocol itself, but shows how the new key shared among agents can be used: A sends a message M encrypted by means of $k = (G^{n_2}$ *mod* $N)^{n_1}$ *mod* N to B which in turn can decrypt it by means of the same key $k = (G^{n_1}$ *mod* $N)^{n_2}$ *mod* N.

$$\begin{array}{l}
\hline
(1)\ A \rightarrow B : X = G^{n_1}\ mod\ N \\
(2)\ B \rightarrow A : Y = G^{n_2}\ mod\ N \\
(3)\ A \rightarrow B : \{M\}_{Y^{n_1}\ mod\ N} \\
\hline
\end{array}$$

Fig. 1. The Diffie-Hellman key exchange protocol

6.1 Representation of the Diffie-Hellman Operator

Here we define a new operator which allows to model computations made in the Diffie-Hellman key exchange protocol. Let us define:

$$\ll \sigma \gg_N^{\sigma_1 \ldots \sigma_n} = \sigma^{\sigma_1 \ldots \sigma_n}\ mod\ N\ . \tag{35}$$

The new operator has the following properties:

$$\ll \sigma \gg_N^{\sigma_1 \ldots \sigma_n} = \ll \sigma \gg_N^{\Pi(\sigma_1 \ldots \sigma_n)}\ , \tag{36}$$

where $\Pi(\sigma_1 \ldots \sigma_n)$ is a permutation, which captures the commutative property of the exponent product, and:

$$\ll \ll \sigma \gg_N^{\sigma_{11} \ldots \sigma_{1n}} \gg_N^{\sigma_{21} \ldots \sigma_{2m}} = \ll \sigma \gg_N^{\sigma_{11} \ldots \sigma_{1n} \sigma_{21} \ldots \sigma_{2m}}\ , \tag{37}$$

which reflects the ability to associate nested invocations of the operator in base position.

The generic term canonicalisation rewrite rules outlined in Sect. 4.3 can be specialised for the Diffie-Hellman operator, taking into account its well-known operator's algebraic properties recalled in (35-37). The specialised rewrite rules are:

$$\frac{\begin{array}{l}(\exists i \mid \sigma_i \neq \mathcal{C}^*(\sigma_i)\ \vee\ \exists i,j \mid \mathcal{C}^*(\sigma_i) \not\leq_{\mathcal{T}} \mathcal{C}^*(\sigma_j))\ \wedge \\ \exists k_1 \ldots k_n,\ \ k_i \neq k_l\ \ \forall i,l \mid \mathcal{C}^*(\sigma_{k_i}) \leq_{\mathcal{T}} \mathcal{C}^*(\sigma_{k_{i+1}}), i = 1 \ldots n-1\end{array}}{\ll \sigma \gg_N^{\sigma_1, \ldots \sigma_n} \overset{\mathcal{C}}{\longmapsto} \ll \sigma \gg_N^{\mathcal{C}^*(\sigma_{k_1}), \ldots \mathcal{C}^*(\sigma_{k_n})}} \tag{38}$$

$$\frac{\sigma = \ll \sigma' \gg_N^{\sigma_{11}, \ldots \sigma_{1n}}}{\ll \sigma \gg_N^{\sigma_{21}, \ldots \sigma_{2m}} \overset{\mathcal{C}}{\longmapsto} \ll \sigma' \gg_N^{\sigma_{11}, \ldots \sigma_{1n}, \sigma_{21} \ldots \sigma_{2m}}} \tag{39}$$

The intruder's ability to compute the Diffie-Hellman operator can be expressed by the following closure rules:

$$\sigma' \in \widehat{\Sigma}\ \wedge\ \sigma_1, \ldots \sigma_n \in \widehat{\Sigma}\ \wedge\ \sigma_i \leq_{\mathcal{T}} \sigma_{i+1}\ \ \forall i \in [1, n-1]\ \Rightarrow\ \ll \sigma' \gg_N^{\sigma_1, \ldots \sigma_n} \in \widehat{\Sigma} \tag{40}$$

$$\ll \sigma' \gg_N^{\sigma_{11}, \ldots \sigma_{1n}} \in \widehat{\Sigma}\ \wedge\ \sigma_{21}, \ldots \sigma_{2m} \in \widehat{\Sigma}\ \Rightarrow\ \mathcal{C}^*(\ll \sigma' \gg_N^{\sigma_{11}, \ldots \sigma_{1n}, \sigma_{21}, \ldots \sigma_{2m}}) \in \widehat{\Sigma} \tag{41}$$

Rule (40) captures the commutative property of the Diffie-Hellman operator with respect to $\sigma_1, \ldots \sigma_n$; similarly, rule (41) captures the ability to flatten nested invocations of the Diffie-Hellman operator in base position by grouping exponents $\sigma_{11}, \ldots \sigma_{1n}$ and $\sigma_{21}, \ldots \sigma_{2m}$ together.

In rule (41), the invocation of the canonicalisation operator $\mathcal{C}^*$ avoids the introduction of non-canonical terms in $\widehat{\Sigma}$; in rule (40), canonicalisation is unnecessary, because the rule's prerequisites implicitly ensure that the right-hand term is canonic.

According to the $\widehat{\Sigma}$ closure rules above, we can introduce the following additional $\overline{\Sigma}$ defining predicates; below, S_1 and S_2 represent all possible partitions of $S = \{\sigma_1, \dots \sigma_n\}$ and, with a little abuse of notation, we let $\ll\sigma\gg_N^{S_1} = \ll\sigma\gg_N^{\sigma_{11},\dots\sigma_{1k}}$

$$\ll\sigma\gg_N^{\sigma_1,\dots\sigma_n} \in \overline{\Sigma} \Leftrightarrow (\sigma \notin \widehat{\Sigma} \lor \exists i \mid \sigma_i \notin \widehat{\Sigma}) \land \sigma_j \leq_{\mathcal{T}} \sigma_{j+1} \; \forall j \in [1, n-1] \quad (42)$$

$$\ll\sigma\gg_N^{\sigma_1,\dots\sigma_n} \in \overline{\Sigma} \Leftrightarrow \forall S_1, S_2 \mid \begin{cases} S_1 = \{\sigma_{11}, \dots \sigma_{1k}\}, \;\; S_2 = \{\sigma_{21}, \dots \sigma_{2l}\}, \\ S_1 \neq \emptyset \land S_2 \neq \emptyset \land S_1 \cap S_2 = \emptyset \land \\ S_1 \cup S_2 = \{\sigma_1, \dots \sigma_n\} \land \\ (\ll\sigma\gg_N^{S_1} \notin \widehat{\Sigma} \lor \exists i \mid \sigma_{2i} \notin \widehat{\Sigma}) \end{cases} \quad (43)$$

Last, closure rules (40) and (41) induce the following $\overline{\Sigma}$ reduction rules:

$$\ll\sigma\gg_N^{\sigma_1,\dots\sigma_n} \in \Sigma_i \land \mathtt{r}(\sigma, \Sigma_i) \land \mathtt{r}(\sigma_j, \Sigma_i) \; \forall j \in [1, n] \rightarrow \langle \{\ll\sigma\gg_N^{\sigma_1,\dots\sigma_n}\}, (40), \emptyset \rangle \quad (44)$$

$$\left.\begin{array}{l} \ll\sigma\gg_N^{\sigma_1,\dots\sigma_n} \in \Sigma_i \land \exists S_1, S_2 \mid S_1 = \{\sigma_{11}, \dots \sigma_{1k}\}, \\ S_2 = \{\sigma_{21}, \dots \sigma_{2l}\}, S_1 \neq \emptyset \land S_2 \neq \emptyset \land S_1 \cap S_2 = \emptyset \\ \land S_1 \cup S_2 = \{\sigma_1, \dots \sigma_n\} \land \mathtt{r}(\ll\sigma\gg_N^{S_1}, \Sigma_i) \land \mathtt{r}(\sigma_{2j}, \Sigma_i) \; \forall j \in [1, l] \end{array}\right\} \rightarrow \langle \{\ll\sigma\gg_N^{\sigma_1,\dots\sigma_n}\}, (41), \emptyset \rangle \quad (45)$$

and the following extension to the $\mathtt{r}(\cdot, \cdot)$ function:

```
boolean r($\sigma$, Σ) {
  ...
  else if $\sigma = \ll\sigma'\gg_N^{\sigma_1,\dots\sigma_n}$
  then return ( r($\sigma'$, Σ)  ∧  r($\sigma_j$, Σ) ∀j )  ∨
              ($\exists S_1 = \{\sigma_{11}, \dots \sigma_{1n}\}, S_2 = \{\sigma_{21}, \dots \sigma_{2m}\} \mid S_1 \neq \emptyset \land S_2 \neq \emptyset \land$
              $S_1 \cap S_2 = \emptyset \land S_1 \cup S_2 = \{\sigma_1, \dots \sigma_n\} \land$
              r($\ll\sigma'\gg_N^{S_1}$, Σ)  ∧  r($\sigma_{2j}$, Σ) ∀j);
  ...
}
```

6.2 Example of Intruder's Knowledge Management

Table 3 shows how the intruder's knowledge grows up when the intruder intercepts the messages exchanged between A and B of Fig. 1: $\overline{\Sigma}$ is the minimised intruder's knowledge, ρ is the eavesdropped message, and column $\mathcal{C}$ contains the number of the canonicalisation rules needed to put ρ in canonical form.

Table 3. An example of canonicalisation

$\overline{\Sigma}$	ρ	$\mathcal{C}$
$\{c, G, N\}$	$\ll G\gg_N^{n_1}$	
$\{c, G, N, \ll G\gg_N^{n_1}\}$	$\ll G\gg_N^{n_2}$	
$\{c, G, N, \ll G\gg_N^{n_1}, \ll G\gg_N^{n_2}\}$	$\{M\}_{\ll\ll G\gg_N^{n_2}\gg_N^{n_1}}$	(39)
	$\{M\}_{\ll G\gg_N^{n_2 n_1}}$	(38)
	$\{M\}_{\ll G\gg_N^{n_1 n_2}}$	
$\{c, G, N, \ll G\gg_N^{n_1}, \ll G\gg_N^{n_2}, \{M\}_{\ll G\gg_N^{n_1 n_2}}\}$		

The first and second message of Fig. 1 are already in canonical form, thus no canonicalisation is needed. Moreover, they cannot be split into simpler sub-messages, thus the new minimal intruder's knowledge is obtained simply by adding these messages to the initial one.

The last message of Fig. 1 has been encrypted by a nested invocation of the new operator thus, by rule (39), the associative property is exploited and $\{M\}_{\ll G\gg_N^{n_2 n_1}}$ is obtained. Last, rule (38) gives the canonical form $\{M\}_{\ll G\gg_N^{n_1 n_2}}$ where each exponent precedes the next with respect to the total order relation among terms (we assume $n_1 \leq_{\mathcal{T}} n_2$). Table 4 shows what happens to the intruder's knowledge when n_1 belongs to the initial $\overline{\Sigma}$, too. Since the canonicalisation details have already been analysed in the previous example, here we assume that each message the intruder intercepts is already in canonical form ($\mathcal{C}^*(\rho)$), and we focus on the intruder's knowledge minimisation steps, induced by the presence of n_1 in the initial $\overline{\Sigma}$.

At each protocol step $\overline{\Sigma}$ (obtained from the previous step) and the intercepted message $\mathcal{C}^*(\rho)$ are listed in a row. Below, the sequence of reduction steps follows,

Table 4. An example of reductions

	$\overline{\Sigma} = \{c, G, N, n_1\}$		$\mathcal{C}^*(\rho) = \ll G\gg_N^{n_1}$	
i	Σ_i	$\mathcal{R}$	Σ_I	Σ_O
0	$\{c, G, N, n_1, \ll G\gg_N^{n_1}\}$	(44)	$\{\ll G\gg_N^{n_1}\}$	$\emptyset$
1	$\{c, G, N, n_1\}$			
	$\overline{\Sigma} = \{c, G, N, n_1\}$		$\mathcal{C}^*(\rho) = \ll G\gg_N^{n_2}$	
i	Σ_i	$\mathcal{R}$	Σ_I	Σ_O
0	$\{c, G, N, n_1, \ll G\gg_N^{n_2}\}$			
	$\overline{\Sigma} = \{c, G, N, n_1, \ll G\gg_N^{n_2}\}$		$\mathcal{C}^*(\rho) = \{M\}_{\ll G\gg_N^{n_1 n_2}}$	
i	Σ_i	$\mathcal{R}$	Σ_I	Σ_O
0	$\{c, G, N, n_1, \ll G\gg_N^{n_2}, \{M\}_{\ll G\gg_N^{n_1 n_2}}\}$	(17)	$\{\{M\}_{\ll G\gg_N^{n_1 n_2}}\}$	$\{M\}$
1	$\{c, G, N, n_1, \ll G\gg_N^{n_2}, M\}$			
	$\overline{\Sigma} = \{c, G, N, n_1, \ll G\gg_N^{n_2}, M\}$			

starting from $\Sigma_0 = \overline{\Sigma} \cup \{\mathcal{C}^*(\rho)\}$ and ending when a new minimised Σ_i has been obtained. For each step, the reduction rule used ($\mathcal{R}$) and the corresponding Σ_I and Σ_O sets are shown.

Starting from $\overline{\Sigma} = \{c, G, N, n_1\}$, the intruder captures the first message $\ll G \gg_N^{n_1}$ sent from A to B, thus $\Sigma_0 = \{c, G, N, n_1, \ll G \gg_N^{n_1}\}$ is obtained. $\ll G \gg_N^{n_1}$ can be synthesised starting from G and n_1 (N is embedded into the operator itself, but this is correct since N and G are *publicly agreed* between A and B), in fact rule (44) allows to remove it from Σ_0.

The next intercepted message is $\ll G \gg_N^{n_2}$; it is added to $\overline{\Sigma}$ without any reduction, since it can neither be synthesised from simpler messages, nor it allows to decode some message already in $\overline{\Sigma}$. In fact, premises of both rules (44) and (45) fail.

The last intercepted message is $\{M\}_{\ll G \gg_N^{n_1 n_2}}$. Since $\mathtt{r}(\Sigma_0, \ll G \gg_N^{n_1 n_2})$ is true, then rule (17) allows to remove $\{M\}_{\ll G \gg_N^{n_1 n_2}}$ from, and add M to the intruder's knowledge. In fact $\mathtt{r}(\ll G \gg_N^{n_1 n_2}, \Sigma_0)$ corresponds to the last extension to $\mathtt{r}(\cdot, \cdot)$ made before the examples. By defining $S_1 = \{n_2\}$ and $S_2 = \{n_1\}$, we have that both $\mathtt{r}(\ll G \gg_N^{S_1}, \Sigma_0)$ and $\mathtt{r}(n_1, \Sigma_0)$ are true.

7 Conclusions

A compact and efficient intruder's knowledge representation is a key point for all formal techniques that use model checking to verify cryptographic protocols. Current techniques rely on specification languages and intruder's knowledge models that allow to deal only with abstract term composition operators, without any mathematical property. This prevents to model protocols that heavily use commutative and associative operators to build their messages. In this paper we have shown how to overcome such a limitation by enabling an existing intruder's knowledge representation technique to successfully handle also messages built by means of associative and commutative operators. Then, theoretical results have been applied to build the set of rules needed to handle the knowledge of a Dolev-Yao intruder that eavesdrops on a session of the Diffie-Hellman key exchange protocol.

As a further enhancement, we plan to extend this technique in order to enable it to manage operators with operand self-cancellation properties, such as the exclusive-or.

References

1. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols. The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.
2. Michele Boreale and Maria Grazia Buscemi. A framework for the analysis of security protocols. In *Proceedings of CONCUR'01*. Springer-Verlag, 2002.
3. Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for cryptographic processes. In *Proceedings of 14th IEEE LICS*, pages 157–166. IEEE Computer Society Press, 1999.

4. Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR, 2003. To appear in *Proc. of 18th IEEE LICS'2003*.
5. Ivan Cibrario Bertolotti, Luca Durante, Riccardo Sisto, and Adriano Valenzano. A new knowledge representation strategy for cryptographic protocol analysis. In *Proceedings of TACAS'03*, volume 2619 of *Lecture Notes in Computer Science*, pages 284–298. Springer-Verlag, April 2003.
6. E. M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proceedings of IFIP PROCOMET*, pages 87–106, London, 1998. Chapman & Hall.
7. E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, October 2000.
8. H. Comon-Lundh and V. Shmatikov. Constraint solving and insecurity decision in presence of exclusive or, June 2003. To appear in *Proc. of 18th IEEE LICS'2003.*
9. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.
10. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
11. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
12. Gavin Lowe. Casper: a compiler for the analysis of security protocols. In *Proceedings of 10th IEEE CSFW*, pages 18–30. IEEE Computer Society Press, June 1997.
13. Gavin Lowe. Towards a completeness result for model checking security protocols. *Journal of Computer Security*, 7(2–3):89–146, 1999.
14. W. Marrero, E. M. Clarke, and S. Jha. A model checker for authentication protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
15. Catherine Meadows and Paliath Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Proceedings of WITS'02*, 2002.
16. Jonathan Millen and Vitaly Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation, June 2003. To appear in *Proc. of 16th IEEE CSFW.*
17. Jonathan K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, 13(2):274–288, February 1987.
18. David Monniaux. Abstracting cryptographic protocols with tree automata. In *Sixth International Static Analysis Symposium (SAS'99)*, number 1694 in Lecture Notes in Computer Science, pages 149–163. Springer Verlag, 1999.
19. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
20. Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceedings of 14th IEEE CSFW*, pages 174–187. IEEE Computer Society Press, June 2001.
21. Steve Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, September 1998.