# A Note on First-order Logic and Security Protocols

Reynald Affeldt    Hubert Comon-Lundh

Research Center for Information Security (RCIS),
National Institute of Advanced Industrial Science and Technology (AIST)
{reynald.affeldt,h.comon-lundh}@aist.go.jp

**Abstract**

We make a simple observation on the relationship between two formalisms for security protocols. The first one is used for the verification of a bounded number of sessions and can be modeled in first-order logic with rigid variables. The second one is an approximation for the verification of an unbounded number of sessions and is typically used in ProVerif. We show that actually the bounded number of sessions case can be simply represented in a first-order logic setting, without introducing false attacks.

This simple observation allows to translate results, which are obtained in one framework into results in the other framework.

## 1 Introduction

It is convenient and simple to model the security of cryptographic protocols within first-order logic. It is indeed possible then to use general purpose theorem provers such as SPASS (see first experiments for security protocols in [17]). There are also successful verification tools such as ProVerif [4], which are based on first-order logic.

However, one drawback of such a formalization is that it requires some approximations. First, global properties such as freshness require a heavy encoding to be faithfully represented in first-order logic (see e.g., [10]), which is not amenable to further automation.

Second, pieces of messages that can be replaced with any message (since they cannot be analyzed by the recipient) are abstracted by variables. Such variables are naturally universally quantified in first-order logic. However, if an attacker can indeed replace these messages with an arbitrary forged message (hence a universal quantification), he should be allowed to do it only once for every variable. More precisely, the attacker can choose the substitution, but has to commit on this value. On the other hand, in a first-order logic formulation, since $\forall x.\phi(x)$ is equivalent to $(\forall x.\phi(x)) \wedge (\forall y.\phi(y))$, the attacker may use two distinct substitutions for the same variable. Hence, in general, the attacker model in first-order logic corresponds to a stronger attacker than the real one. We will give concrete examples in Sect. 2.

It follows that attacks that are found in a first-order logic formulation of security might be false attacks. This has been well-known for a long time and it is the reason why several more accurate formalisms have been designed, for instance using MSR or linear logic [5].

Instead of proving security for an arbitrary number of sessions, much work focuses on finding an attack for a bounded number of sessions (e.g., [15]). In this setting, there is no need for approximation: the insecurity problem can be translated into deducibility constraints, after guessing an interleaving of actions. Translating this approach in first-order logic is not straightforward, for the same reason as above: we must express that a variable, though universally quantified, can be instantiated only once.

A simple way to fix the problems and remove the false attacks due to universal quantification is to use *rigid variables*: while universally quantified, standard variables can be instantiated as many times as we wish, rigid variables get only one instance [3]. This is exactly what we need for modeling security protocols. We need however one set of rigid variables for each session. Therefore, in practice, this can only be applied for a bounded number of sessions. This is exactly what is done in [13]: the authors introduce rigid clauses to model the protocol rules when the number of sessions is fixed. Then they design a proof calculus for such clauses and show a termination result.

To the best of our knowledge, there is no formulation of the security problem for a bounded number of sessions within first-order logic, which avoids false attacks. This is what we do in the present note. Actually, we show that there is a simple translation of rigid variables in first-order logic, which preserves the satisfiability of formulas. It follows for instance that we can capture rigid-validity, hence security for a bounded number of sessions, within first-order logic. Then we can use first-order theorem provers for finding attacks in a bounded number of sessions, without generating false attacks. This can be useful, when trying to reconstruct attacks from candidate attacks found by a theorem prover: this is a simple alternative to [2]. It can also be used to search simultaneously, with the same tool, for attacks and proofs.

Using this simple translation of rigid variables in first-order logic, we may considerably simplify the proof rules for rigid clauses of [13]. We can also extend the calculus, allowing for clauses mixing rigid and non-rigid variables as well as equalities (however only flexible variables are allowed in equalities). Finally, we can translate back and forth results from first-order logic to first-order logic with rigid variables. For instance, from [9], we derive a decision result for a bounded number of sessions for protocols including exclusive-or. The other way around, from the decision results on security protocols for a bounded number of sessions, we derive decision results (in co-NP) for fragments of first-order logic.

We recall some protocol models in Sect. 2, through examples. In Sect. 3, we show the simple translation from rigid clauses to first-order clauses. In Sect. 4 we sketch some possible applications.

## 2 First-order Models of Protocols

We give the main ideas about first-order models of protocols (see e.g., [4] for further details) and examples to illustrate false attacks.

### 2.1 A Standard Model

Let $I$ be a predicate symbol that captures the intruder's knowledge. The attacker capabilities are described by a set of Horn clauses:

$$\begin{array}{lrcl}
(I0) & I(x), I(y) & \to & I(\langle x, y\rangle) \\
(I1) & I(\langle x, y\rangle) & \to & I(x) \\
(I2) & I(x), I(y) & \to & I([x]_y) \\
(I3) & I([x]_y), I(y) & \to & I(x) \\
(I4) & & \to & I(pk(x)) \\
(I5) & I(x), I(pk(y)) & \to & I(\{x\}_{pk(y)}) \\
(I6) & I(\{x\}_{pk(y)}), I(y) & \to & I(x) \\
 & & \cdots &
\end{array}$$

The function symbol $pk$ represents the public key corresponding to some private key. Public-key encryption and symmetric encryption are respectively represented by $\{\cdot\}_{\cdot}$ and $[\cdot]_{\cdot}$. The pairing function $\langle\cdot,\cdot\rangle$ will sometimes be omitted or considered as variadic to ease reading. The rules $(I1)$, $(I6)$, etc. can also be replaced with explicit destructors

$$\begin{array}{lrcl}
(I1') & I(x) & \to & I(\pi_1(x)) \\
(I6') & I(x), I(y) & \to & I(\mathrm{pdec}(x, y)) \\
 & & \cdots &
\end{array}$$

together with equations $\pi_1(\langle x, y\rangle) = x$, $\mathrm{pdec}(\{x\}_{pk(y)}, y) = x$, etc. We will use either of the two formalisms without mention.

Protocol clauses are modeled considering the protocol as an oracle used by the attacker. Freshness is approximated using a function symbol, which depends on the terms seen at this stage. The following sections illustrate this with examples.

## 2.2 An Example of False Attack

Let us consider a protocol from [7], which we write first in the (somewhat ambiguous) Alice-and-Bob notation:

$$\begin{array}{lcl} A \to B & : & \{pk_A, N\}_{pk_B} \\ B \to A & : & \{N, K\}_{pk_A} \\ A \to B & : & [S]_K \\ B \to A & : & N \end{array}$$

and, more formally, in an extended $\pi$-calculus notation (as described e.g., in [1]):

$$\begin{array}{l} P_A(pk_A, sk_A, pk_B) \overset{def}{=} c(x_{pk_B}).\nu s.\nu N.\overline{c}\langle\{\langle pk_A, N\rangle\}_{x_{pk_B}}\rangle. \\ \quad c(x_{nk}).\text{let } (non, k) = \text{pdec}(x_{nk}, sk_A) \text{ in if } non = N \text{ then if } x_{pk_B} = pk_B \text{ then } \overline{c}\langle[s]_k\rangle \\ P_B(sk_B) \overset{def}{=} c(x_m).\text{let } (pka, non) = \text{pdec}(x_m, sk_B) \text{ in } \nu k.\overline{c}\langle\{\langle non, k\rangle)\}_{pka}\rangle. \\ \quad c(x_{sk}).\text{let } s = \text{sdec}(x_{sk}, k) \text{ in } \overline{c}\langle non\rangle \\ P_0 \overset{def}{=} !(\nu sk_A.\text{let } pk_A = pk(sk_A) \text{ in } \overline{c}\langle pk_A\rangle.\nu sk_B.\text{let } pk_B = pk(sk_B) \text{ in } \overline{c}\langle pk_B\rangle. \\ \qquad !P_A(pk_A, sk_A, pk_B) \mid !P_B(sk_B)) \\ \quad \mid \quad !(\nu sk_A.\text{let } pk_A = pk(sk_A) \text{ in } \overline{c}\langle pk_A\rangle.\overline{c}\langle sk_A\rangle.\nu sk_B.\text{let } pk_B = pk(sk_B) \text{ in } \overline{c}\langle pk_B\rangle.\overline{c}\langle sk_B\rangle. \\ \qquad !P_A(pk_A, sk_A, pk_B) \mid !P_B(sk_B)) \end{array}$$

The security property (which we do not display here) states that any secret $s$ generated by an honest identity $A$ for an honest identity $B$ (given as the first input in the process $P_A$) is never disclosed.

In first-order logic, the protocol is modeled as

$$\begin{array}{lrcl} (\Delta 1) & I(pk(x)), I(pk(y)) & \to & I(\{pk(x), N(x,y)\}_{pk(y)}) \\ (\Delta 2) & I(\{x,y\}_{pk(z)}) & \to & I(\{y, K(x,y,z)\}_x) \\ (\Delta 3) & I(\{N(x,y), z\}_{pk(x)}) & \to & I([S(x,y,z)]_z) \\ (\Delta 4) & I(\{w\}_{K(x,y,z)}) & \to & I(y) \end{array}$$

We avoid here the explicit destructors, while they are used in the process description, for readability reasons. (See appendix A for a ProVerif encoding.)

Basically, the intruder uses the protocol as an oracle: for each protocol rule, if the intruder can construct a message matching the expected pattern, then it gets the corresponding reply message. For example, the clause ($\Delta 1$) represents the first action of process $P_A$ (session initialization), the clause ($\Delta 2$) represents the first action of process $P_B$ (upon reception from $A$ of $\{x, y\}_{pk(sk_B)}$ which is supposed to be $\{pk(sk_A), N\}_{pk(sk_B)}$ the reply of $B$ is the message $\{y, K\}_x$), etc.

Finally, the security property can be modeled as $\neg I(S(sk_A, sk_B, z))$ and $I(sk_C)$ where $C$ is a constant representing a corrupted identity. If the protocol is insecure, then the set of clauses is unsatisfiable: there is a derivation of $I(S(sk_A, sk_B, t))$ for some $t$.

In the clauses above, the freshness of generated data $N$, $K$, and $S$ is abstracted using universally quantified variables. This may be a cause of false attacks as, for instance, every session between $A$ and $B$ will use the same representation of $N$. For a bounded number of sessions, this problem does not occur as different symbols can be used for nonces occurring in different sessions.

There are however other sources of false attacks. In the above example, the protocol is (supposedly) secure, while there is a simple derivation of the empty clause: from a honest session of the protocol (i.e., using clauses ($\Delta 1$) to ($\Delta 4$) once), we derive $I(N(x,y))$. Now, for any $z$ such that $I(z)$ we derive $I(\{N(x,y), z\}_{pk(x)})$ using the intruder capabilities. Next, using clause ($\Delta 3$) we get $I([S]_z)$ and, from this clause and $I(z)$, we derive $I(S)$ by decrypting.

The problem here is that the nonce is first kept secret but eventually revealed. A first-order model leads to a false attack by wrongly inferring that the intruder could have the nonce at an early stage: when the nonce $N$ is revealed, the rule ($\Delta 3$) is replayed and the intruder gets back $[S]_{K'}$ for a key $K'$ of his choice, which he can decrypt. This cannot occur in the process model, as the agents would have moved forward their internal state, preventing the replay of rule ($\Delta 3$). This kind of problem occurs even for a single session, as shown by the example.

### 2.3 Trying to Refine the Model: there are still False Attacks

The false attack above comes from the ability (in the model) to play again a rule of the protocol after completing it. One may think that this can be fixed by adding some state information at each step of the protocol. While this is quite difficult for an unbounded number of sessions, there is an easy (though expensive) encoding for a bounded number of sessions.

First, we can get rid of freshness encoding by simply modeling nonces with distinct constants. It is also possible (though expensive; we will avoid this in the encoding of next section), to guess an interleaving of actions and use a different predicate symbol at each step: instead of a single predicate symbol $I$, we could use symbols $I_0, \dots, I_n$ representing the intruder knowledge after $n$ steps. Then the protocol clauses increase the index of the predicate:

$$I_k(t) \to I_{k+1}(u) \quad \text{for } k = 0, \dots, n-1$$

for a rule stating that, at stage $k$, upon receiving an instance of $t$, the agent sends the corresponding instance of $u$. The security is stated as $\neg I_n(s)$ where $s$ is the supposed secret. And we have to add the clauses expressing the increasingness of the intruder knowledge:

$$I_k(x) \to I_{k+1}(x) \quad \text{for } k = 0, \dots, n-1$$

Finally, clauses reflecting intruder capabilities are replicated $n$ times with the indices $1, \dots, n$.

With such an encoding, the above false attack can be prevented. However, this is not sufficient in general. Here is an (cook-up) example showing again a false attack in this new setting:

**Example 2.1**

$$\begin{array}{lcl} A \to B & : & [A, N_0]_{K_{AB}} \\ B \to A & : & [B, N_0, N_1]_{K_{AB}}, [B, N_0, N_2]_{K_{AB}} \\ A \to B & : & N_1 \end{array}$$

*$A$, relying on a long term shared secret $K_{AB}$, wants to establish a short term secret key. $B$ generates two nonces $N_1, N_2$ and sends them separately. $A$ acknowledges both nonces by sending back one of them. The new short-term secret is $N_1 \oplus N_2$.*

In a single-session model, there is no attack: the intruder can get either $N_1$ or $N_2$, but not both. However, in a clausal formulation as explained above, we would have two clauses:

$$\begin{array}{rcl} I_1([A, x]_{K_{AB}}) & \to & I_2(\langle [B, x, N_1]_{K_{AB}}, [B, x, N_2]_{K_{AB}} \rangle) \\ I_2(\langle [B, N_0, x]_{K_{AB}}, [B, N_0, y]_{K_{AB}} \rangle) & \to & I_3(x) \end{array}$$

From $I_2(\langle [B, x, N_1]_{K_{AB}}, [B, x, N_2]_{K_{AB}} \rangle)$ we infer $I_2(\langle [B, x, N_2]_{K_{AB}}, [B, x, N_1]_{K_{AB}} \rangle)$. Then using two instances of the second clause, we get immediately $I_3(N_1)$ and $I_3(N_2)$, hence the secret. This is however a false attack: the last rule should not be played twice.

## 3 Rigid Clauses vs. Classical Clauses

The best way to prevent the last false attack is to use rigid variables. Encoding a bounded number of sessions does not require then to introduce new predicate symbols, nor to guess the interleaving of messages.

Let us first introduce the less standard rigid variables and rigid clauses and then show how to get rid of them.

## 3.1 Rigid Clauses

Let $\mathcal{F}$ and $\mathcal{P}$ be resp. a set of function symbols and a set of predicate symbols with their arities.

We use the classical vocabulary of first-order logic. To fix notations, if $\mathcal{S}$ is a first-order structure and $P$ is a predicate symbol, $[\![P]\!]^{\mathcal{S}}$ is the interpretation of $P$ in $\mathcal{S}$.

Formulas are clauses whose variables are either *rigid* (written in upper-case) or *flexible* (written in lower-case). Both types of variables are universally quantified, but rigid variables can only get one instance.

**Example 3.1** *Consider the following set of clauses (taken from [13]):*

$$\{I(a),\ I(b),\ \neg I(X) \vee I(f(X)),\ \neg I(f(a)) \vee \neg I(f(b))\}$$

*If $X$ was considered as an ordinary first-order variable, this set of clauses would be unsatisfiable: from the three first clauses we can infer both $I(f(a))$ and $I(f(b))$. The derivation of an empty clause however requires two instances of the third clause, which is forbidden for rigid variables. We can choose the instance $X = a$ or the instance $X = b$, but not both.*

*The above set clauses is satisfiable in our intended interpretation of rigid variables since both sets of ground clauses*

$$\{I(a),\ I(b),\ \neg I(a) \vee I(f(a)),\ \neg I(f(a)) \vee \neg I(f(b))\}$$

*and*

$$\{I(a),\ I(b),\ \neg I(b) \vee I(f(b)),\ \neg I(f(a)) \vee \neg I(f(b))\}$$

*are satisfiable.*

The next example shows that resolution procedures cannot be easily extended to clauses containing rigid variables.

**Example 3.2** *Consider the following set of clauses:*

$$\{I(X),\ \neg I(f(x)) \vee I_0,\ \neg I(g(x))\}$$

*It is unsatisfiable: the first and the third clauses resolve to the empty clause.*

*Consider however that we start by resolving the two first clauses. This yields the new set of clauses*

$$\{I(f(Y)),\ I_0,\ \neg I(f(x)) \vee I_0,\ \neg I(g(x))\}$$

*where $Y$ is a new rigid variable resulting from the unification $X \stackrel{?}{=} f(x)$. We can still choose $Y$, but we committed to an assignment of $X$ to a term headed with $f$. Now the set of clauses is satisfiable.*

*For a complete resolution procedure, we would have to restart from the beginning, with another choice of clauses to resolve.*

The last example shows that, unlike classical first-order clauses, resolution does not yield a logically equivalent set of clauses. Therefore, resolution theorem proving would have to be reconsidered; this is the reason for complications in [13].

Let us now formalize the model-theoretic part of such calculi.

Let $\mathcal{S}$ be an $\mathcal{F}, \mathcal{P}$-structure whose underlying $\mathcal{F}$-algebra is $\mathcal{A}$. Let $\mathcal{C}$ be a set of clauses whose free variables are split into $\overline{X}$ (rigid free variables) and $\overline{y}$ (flexible free variables). If $\sigma$ is an assignment of $\overline{X}$ in $\mathcal{A}$, we have $\mathcal{S}, \sigma \models \forall \overline{y}.\mathcal{C}$ defined as usual in first-order logic.

**Definition 1** *Let $\mathcal{C}$ be a set of clauses whose free variables are split into $\overline{X}$ (rigid free variables) and $\overline{y}$ (flexible free variables).*

*$\mathcal{C}$ is* satisfiable *if there is a $\mathcal{F}$-algebra $\mathcal{A}$ such that, for any $\mathcal{A}$-assignment $\sigma$ of $\overline{X}$, there is a structure $\mathcal{S}$ whose underlying algebra is $\mathcal{A}$ such that $\mathcal{S}, \sigma \models \forall \overline{y}.\mathcal{C}$.*

In other words, models of formulas with rigid variables are collections of structures, one for each assignment of the rigid variables.

**Example 3.3** *On example 3.1, for any of the two assignments $X \mapsto a$ and $X \mapsto b$ there is a model: $\{I(a), I(b), I(f(a)), \neg I(f(b))\}$ in the first case and $\{I(a), I(b), I(f(b)), \neg I(f(a))\}$ in the second case.*

**Example 3.4** *The one session case of Example 2.1 can be translated into the following rigid clauses (keeping the intruder rules with flexible variables):*

$$\begin{array}{rcl} & \rightarrow & I([A, N_0]_{K_{AB}}) \\ I([A, X]_{K_{AB}}) & \rightarrow & I(\langle [B, X, N_1]_{K_{AB}}, [B, X, N_2]_{K_{AB}} \rangle) \\ I(\langle [B, N_0, Y]_{K_{AB}}, [B, N_0, Z]_{K_{AB}} \rangle) & \rightarrow & I(Y) \end{array}$$

*Which, together with $\neg I(\langle N_1, N_2 \rangle)$ is satisfiable. In contrast, if the above variables are considered as flexible, it is unsatisfiable (yielding a false attack, as in Example 2.1).*

There are also some traps, for instance, $\forall x.\phi(x) \wedge \psi(x) \models\!\!\!\mid \forall x, y.\phi(x) \wedge \psi(y)$ while $\forall X.\phi(X) \wedge \psi(X) \not\models \forall X, Y.\phi(X) \wedge \psi(Y)$, as shown by the following example.

**Example 3.5** *Consider $\phi(X) = \psi(X) = P(X) \wedge (\neg P(a) \vee \neg P(b))$. On the one hand, $\forall X.\phi(X) \wedge \psi(X)$ is satisfiable. Consider the algebra with two constants $a$ and $b$. For the assignment $X \mapsto a$ (resp. $X \mapsto b$), the structure $\mathcal{S}$ such that $P(a)$ holds (resp. $P(b)$ holds) satisfies $\phi(a) \wedge \psi(a)$ (resp. $\phi(b) \wedge \psi(b)$). On the other hand, $\forall X, Y.\phi(X) \wedge \psi(Y)$ is not satisfiable. Consider any algebra containing the constants $a$ and $b$. For the assignment $X \mapsto a; Y \mapsto b$, there cannot be any structure that satisfies $\phi(a) \wedge \psi(b)$.*

## 3.2 Translation of Rigid Clauses into First-order Clauses

As shown in the Examples 3.2 and 3.4, (dis)proving might be a difficult task when there are rigid variables. There is however a simple observation, yielding a transformation of rigid clauses into flexible ones: as can be seen from the definition of satisfiability, the interpretation of predicates depends on the assignment of rigid variables. Hence, a simple Skolemization argument suffices to eliminate this dependence and leads back to first-order clauses:

**Theorem 2** *For any finite set of clauses $\mathcal{C}$ there is an effectively computable set of clauses $\mathcal{C}'$, which does not contain any rigid variable, and such that $\mathcal{C}$ is satisfiable iff $\mathcal{C}'$ is satisfiable.*

**Proof:** $\mathcal{C}'$ is constructed from $\mathcal{C}$ as follows. If $X_1, \ldots, X_n$ are the rigid variables of $\mathcal{C}$, for each $P \in \mathcal{P}$ of arity $m$ we let $P'$ be a predicate symbol of arity $n + m$. If $\neg P_1(\overline{s_1}) \vee \cdots \vee \neg P_{n_1}(\overline{s_{n_1}}) \vee Q_1(\overline{t_1}) \vee \cdots \vee Q_{n_2}(\overline{t_{n_2}}) \in \mathcal{C}$ we let

$$\neg P'_1(x_1, \ldots, x_n, \overline{s'_1}) \vee \cdots \vee \neg P'_{n_1}(x_1, \ldots, x_n, \overline{s'_{n_1}}) \vee Q'_1(x_1, \ldots, x_n, \overline{t'_1}) \vee \cdots \vee Q'_{n_2}(x_1, \ldots, x_n, \overline{t'_{n_2}})$$

be a clause $C' \in \mathcal{C}'$ where $x_1, \ldots, x_n$ are distinct variables, which do not occur free in the clause $C$ and $s'_1, \ldots, s'_{n_1}, t'_1, \ldots, t'_{n_2}$ are the terms obtained from their unprimed version by replacing each $X_i$ with the corresponding $x_i$.

If $\mathcal{C}$ is satisfiable, then there is an $\mathcal{F}$-algebra $\mathcal{A}$ such that, for any $\mathcal{A}$-assignment $\sigma$ of $X_1, \ldots, X_n$ there is a structure $\mathcal{S}_\sigma$ such that, for every clause $C \in \mathcal{C}$, we have $\mathcal{S}_\sigma, \sigma \models \forall \overline{y}.C$. Consider then the structure $\mathcal{S}'$ (whose underlying algebra is $\mathcal{A}$) such that

$$(a_1, \ldots, a_n, b_1, \ldots, b_m) \in [\![P']\!]^{\mathcal{S}'} \text{ iff } (b_1, \ldots, b_m) \in [\![P]\!]^{\mathcal{S}_{\{X_1 \mapsto a_1; \ldots; X_n \mapsto a_n\}}}$$

For any clause $C$ of $\mathcal{C}$, we claim that $\mathcal{S}' \models \forall \overline{x}, \overline{y}.C'$: for any assignment $\sigma'$ of the variables $x_1, \ldots, x_n$ respectively and for any assignment $\theta$ of the other variables $\overline{y}$ of the clause, we let $\sigma$ be the assignment of the rigid variables defined by $\sigma(X_i) = \sigma'(x_i)$ for every $i$. By hypothesis,

$\mathcal{S}_\sigma, \theta, \sigma \models C$. It follows that, for some literal $L \in C$, $\mathcal{S}_\sigma, \theta, \sigma \models L$. Assume for instance that $L$ is a positive literal (the other case is similar): $L = P(u_1, \ldots, u_m)$ and $(\llbracket u_1 \rrbracket^{\theta,\sigma,\mathcal{A}}, \ldots, \llbracket u_m \rrbracket^{\theta,\sigma,\mathcal{A}}) \in \llbracket P \rrbracket^{\mathcal{S}_\sigma}$. This is equivalent, by definition, to

$$(a_1, \ldots, a_n, \llbracket u_1 \rrbracket^{\theta,\sigma,\mathcal{A}}, \ldots, \llbracket u_m \rrbracket^{\theta,\sigma,\mathcal{A}}) \in \llbracket P' \rrbracket^{\mathcal{S}'}$$

which, again by construction, yields

$$\mathcal{S}', \sigma', \theta \models C'$$

Conversely, if $\mathcal{C}'$ is satisfiable, then let $\mathcal{S}'$ be a structure which satisfies all clauses of $\mathcal{C}'$. Consider an arbitrary assignment $\sigma$ of rigid variables occurring in $\mathcal{C}$. We let $\mathcal{S}_\sigma$ be the structure defined by

$$\llbracket P \rrbracket^{\mathcal{S}_\sigma} = \{(b_1, \ldots, b_m) \mid (X_1\sigma, \ldots, X_n\sigma, b_1, \ldots, b_m) \in \llbracket P' \rrbracket^{\mathcal{S}'}\}$$

As before, $S_\sigma, \sigma \models \forall \overline{y}.\mathcal{C}$ iff $\mathcal{S}' \models \forall \overline{x}, \overline{y}.\mathcal{C}'$.

$\square$

This extends to clauses with equality, provided that every equality clause does not contain any rigid variable.

## 4 Examples of Possible Applications

### 4.1 Automatic Proofs for a Bounded Number of Sessions

Thanks to the effective procedure given in the proof of Theorem 2, we can use resolution for mechanizing proofs in a bounded number of sessions. We believe that the resulting procedure is simpler, both conceptually and from the complexity point of view, than the one given in [13]. In addition, it works as well if we have clauses mixing rigid and flexible variables and also if we have (flexible) equations. Though, in the latter case, there is no guarantee for termination.

**Example 4.1** *Let us come back to Example 3.4. We translate now the rigid clauses into first-order clauses:*

$$\begin{array}{rcl}
 & \rightarrow & I(x, y, z, [A, N_0]_{K_{AB}}) \\
I(x, y, z, [A, x]_{K_{AB}}) & \rightarrow & I(x, y, z, \langle [B, x, N_1]_{K_{AB}}, [B, x, N_2]_{K_{AB}} \rangle) \\
I(x, y, z, \langle [B, N_0, y]_{K_{AB}}, [B, N_0, z]_{K_{AB}} \rangle) & \rightarrow & I(x, y, z, y) \\
I(x, y, z, \langle N_1, N_2 \rangle) & \rightarrow &
\end{array}$$

*Using an appropriate strategy (see next section), resolution terminates in a few steps, yielding in particular the literals $I(N_0, N_1, N_2, N_1)$ and $I(N_0, N_2, N_1, N_2)$ (which, without the three first components, were used to mount a false attack). On the other side, the goal is decomposed into $\neg I(x, y, z, N_1) \vee \neg I(x, y, z, N_2)$ and leads, using the two inferred literals, to clauses $\neg I(N_0, N_2, N_1, N_1)$ and $\neg I(N_0, N_1, N_2, N_2)$. But the empty clause does not belong to the saturated set: there is no one-session attack.*

### 4.2 Decidable Fragments of First-order Logic

Concerning termination, the delicate problem is the design of appropriate strategies, which we do not address thoroughly in this note.

If we translate back in terms of strategies the constraint solving techniques used for the decidability and complexity proofs for a bounded number of sessions (see [8]), we will get a decision result for formulas in the following clausal form[1]:

[1]The part of $I$'s arguments that model rules ordering is put in subscript position to ease reading.

**Theorem 3** *Let $f$ be a function symbol. Assume that all clauses are of one of the following forms:*

1. $I_{\overline{z}}(\overline{x}, y_0), \ldots, I_{\overline{z}}(\overline{x}, y_{n-1}) \rightarrow I_{\overline{z}}(\overline{x}, f(y_0, \ldots, y_{n-1}))$ *with* $\overline{x}$, $\overline{y}$, $\overline{z}$ *pairwise disjoint*
2. $I_{\overline{z}[i \leftarrow k]}(\overline{x}, s) \rightarrow I_{\overline{z}[i \leftarrow k+1]}(\overline{x}, t)$ *with* $Var(t) \subseteq Var(s) \subseteq \overline{x}$ *and* $\overline{x} \cap \overline{z} = \emptyset$
3. $I_{\overline{z}}(\overline{x}, t)$ *with* $Var(t) \subseteq \overline{x}$ *and* $\overline{x} \cap \overline{z} = \emptyset$
4. $\neg I_{\overline{z}}(\overline{x}, s)$ *with* $Var(s) \subseteq \overline{x}$ *and* $\overline{x} \cap \overline{z} = \emptyset$

*where* $\overline{z}[i \leftarrow k]$ *represents the variable-vector* $\overline{z}$ *whose* $i^{th}$ *element is replaced by* $k$. *Then the satisfiability modulo the axioms of encryption/decryption (resp. satisfiability modulo exclusive-or [6, 11], resp. satisfiability modulo Abelian groups [16]) is co-NP-complete.*

This shows a new decidable fragment of first-order logic. It is related to both the extended Skolem class and the $\mathcal{E}^+$ class [14], but it is not subsumed by any of the two classes. So, as a perspective, if we manage to characterize the appropriate strategy, out of the previous theorem, we could get a new non-trivial decidable class of first-order logic.

In the case of the axioms of encryption/decryption, the strategy consisting in binary resolution with the free selection defined just below and deletion of redundant clauses is complete and terminating. The free selection function in question is defined as follows. Consider a well-founded, liftable ordering containing the subterm ordering. For any Horn clause of the form $I_{\overline{u}_0}(\overline{s}_0, t_0), \ldots, I_{\overline{u}_{n-1}}(\overline{s}_{n-1}, t_{n-1}) \rightarrow I_{\overline{u}_n}(\overline{s}_n, t_n)$:

- If there is only one maximal literal, then return the corresponding atom.
- Otherwise, if there is a maximal, negative literal where $t_i$ is not a variable, then return the corresponding atom.
- Otherwise, if the positive literal is maximal and $t_n$ is not a variable, then return it.
- Otherwise, return any atom.

Of course, known decidable fragments of first-order logic (e.g., the one defined in [9]) can be used to infer decision algorithms for a bounded number of sessions. But this is less likely to give interesting results as it will impose necessary restrictions on the protocol.

### 4.3 Enhancing First-order Provers for Security Protocols

Another possible use of the translation given in Theorem 2 is to combine in a single first-order theorem prover the advantages of the approximations and of the bounded number of sessions: using the same engine and specification it is possible to look first for attacks/safety in an exact way for a given number of sessions and then use an approximation for more sessions. Alternatively, in case a candidate attack is found, we could simply check the falsity of the attack using additional clauses.

## 5 Conclusion

We do not claim to contribute here by a deep or difficult result. We believe however that our simple observation is useful for encoding the problem of security for a bounded number of sessions into a first-order clausal form.

This opens also some perspectives in automated deduction: there is a hidden strategy, which yields a decidable fragment of first-order logic and deserves some attention.

## References

[1] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, 2005.

[2] X. Allamigeon and B. Blanchet. Reconstruction of Attacks against Cryptographic Protocols. In *18th IEEE Computer Security Foundations Workshop*, pages 140–154, 2005.

[3] P. B. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28(2):193–214, 1981.

[4] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop*, pages 82–96, 2001.

[5] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *12th IEEE Computer Security Foundations Workshop*.

[6] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *18th IEEE Symp. on Logic in Computer Science (LICS 2003)*, pages 261–270.

[7] A. Cohen. Combined CPV-TLV Security Protocol Verifier. Master's thesis, New York University, 2004.

[8] H. Comon-Lundh. Résolution de contraintes et recherche d'attaque pour un nombre borné de sessions. Lecture notes, Feb. 2004. Available at: `http://www.lsv.ens-cachan.fr/~comon/CRYPTO/bounded.ps`.

[9] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *14th Int. Conf. on Rewriting Techniques and Applications (RTA 2003)*, volume 2706 of *Lecture Notes in Computer Science*, pages 148–164. Springer.

[10] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. *Science of Computer Programming*, 50(1–3):51–71, 2004.

[11] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th IEEE Symp. on Logic in Computer Science (LICS 2003)*, pages 271–.

[12] V. Cortier, M. Rusinowitch, and E. Zalinescu. A resolution strategy for verifying cryptographic protocols with cbc encryption and blind signatures. In *7th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP 2005)*, pages 12–22. ACM.

[13] S. Delaune, H. Lin, and C. Lynch. Protocol verification via rigid/flexible resolution. In *14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2007)*, volume 4790 of *Lecture Notes in Artificial Intelligence*, pages 242–256. Springer.

[14] C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tamet. Resolution decision procedure. In *Handbook of Automated Reasoning*, chapter 25. Elsevier, 2001.

[15] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 1-3(299):451–475, 2003.

[16] V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *13th European Symp. on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 355–369. Springer.

[17] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *16th Conference on Automated Deduction (CADE 1999)*, volume 1632 of *Lecture Notes in Computer Science*, pages 314–328. Springer.

## A A False Attack in ProVerif

Here follows a formalization in ProVerif of the first protocol of Sect. 2.2 whose verification yields a false attack. The clauses `rule 1`, `rule 2`, etc. below correspond respectively to the clauses ($\Delta 1$), ($\Delta 2$), etc. in Sect. 2.2. With respect to Sect. 2.2, the ProVerif encoding makes some sound simplifications, which are not explained here.

```
pred I/1 elimVar,decompData.
fun pk/1. fun penc/2. fun senc/2.
query I:S[skA[],skB[],z]. not I:skA[]. not I:skB[].
reduc

(* the attacker *)
I:x & I:y              -> I:senc(x,y)      ;
I:x & I:senc(y, x)     -> I:y              ;
                          I:pk(x)          ;
I:x & I:pk(y)          -> I:penc(x, pk(y)) ;
I:y & I:penc(x, pk(y)) -> I:x              ;

(* the protocol *)
I:senc(w, k[x, y, z])      -> I:y                          ; (* rule 4 *)
I:penc((N[x,y], z), pk(x)) -> I:senc(S[x,y,z], z)          ; (* rule 3 *)
I:penc((x, y), pk(z))      -> I:penc((y, k[x,y,z]), x)     ; (* rule 2 *)
I:pk(x) & I:pk(y)          -> I:penc((pk(x), N[x,y]), pk(y)) . (* rule 1 *)
```

Analysis by ProVerif yields a false attack represented by a derivation of `I:S[skA[],skB[],z]` for some `z` provided by the attacker. In this derivation, one can observe the execution of a honest session of the protocol (using `rule 1` to `rule 4` once) leading to `I:N[skA[],skB[]]`, the latter being eventually used in a replay of `rule 3`.

```
rule 8 I:S[skA[],skB[],z_308]
  hypothesis I:z_308
  rule 3 I:senc(S[skA[],skB[],z_308],z_308)
    rule 6 I:penc((N[skA[],skB[]],z_308),pk(skA[]))
      2-tuple I:(N[skA[],skB[]],z_308)
        rule 4 I:N[skA[],skB[]]
          rule 3 I:senc(S[skA[],skB[],k[pk(skA[]),N[skA[],skB[]],skB[]]],
                        k[pk(skA[]),N[skA[],skB[]],skB[]])
            rule 2 I:penc((N[skA[],skB[]],k[pk(skA[]),N[skA[],skB[]],skB[]]),pk(skA[]))
              rule 1 I:penc((pk(skA[]),N[skA[],skB[]]),pk(skB[]))
                rule 7 I:pk(skA[])
                rule 7 I:pk(skB[])
        hypothesis I:z_308
      rule 7 I:pk(skA[])

RESULT goal reachable: I:S[skA[],skB[],z_5]
```