Verification of Security Protocols with Lists: From Length One to Unbounded Length

Miriam Paiola and Bruno Blanchet

INRIA, École Normale Supérieure, CNRS, Paris
{paiola,blanchet}@di.ens.fr

Abstract. We present a novel, simple technique for proving secrecy properties for security protocols that manipulate lists of unbounded length, for an unbounded number of sessions. More specifically, our technique relies on the Horn clause approach used in the automatic verifier ProVerif: we show that if a protocol is proven secure by our technique with lists of length one, then it is secure for lists of unbounded length. Interestingly, this theorem relies on approximations made by our verification technique: in general, secrecy for lists of length one does not imply secrecy for lists of unbounded length. Our result can be used in particular to prove secrecy properties for group protocols with an unbounded number of participants and for some XML protocols (web services) with ProVerif.

1 Introduction

Security protocols are protocols that rely on cryptographic primitives such as encryption and signatures for securing communication between several parties. They aim at ensuring security properties such as secrecy or authentication. Historically, attacks were often found against protocols that were thought correct. Furthermore, security flaws cannot be detected by testing since they appear only in the presence of an attacker. The confidence in these protocols can then be increased by a formal analysis that proves the desired security properties. To ease formal verification, one often uses the symbolic, so-called Dolev-Yao model [8], which abstracts from the details of cryptographic primitives and considers messages as terms. In this work, we also rely on this model.

The formal verification of security protocols with fixed-size data structures has been extensively studied. However, the formal verification of protocols that manipulate more complex data structures, such as lists, has been less studied and presents additional difficulties: these complex data structures add another cause of undecidability.

In this work, we present a technique for proving secrecy properties for security protocols that manipulate lists of unbounded length. This technique is based on the Horn clause approach used in the automatic verifier ProVerif [1,4]. ProVerif is an automatic protocol verifier that takes as input a protocol, translates it into a representation in Horn clauses, and uses a resolution algorithm to determine whether a fact is derivable from the clauses. One can then infer security properties of the protocol. For instance, we use a fact $\mathrm{att}(M)$ to mean that the attacker may have the message M. If $\mathrm{att}(s)$ is not derivable from the clauses, then s is secret. The main goal of this approach is to prove security properties of protocols without bounding the number of sessions of the protocol.

Like other protocol verifiers, ProVerif can analyze protocols with lists if we fix the lengths of the lists a priori. However, if the protocol is verified only for some lengths, attacks may exist for other values. So our goal is to prove the protocols for lists of any length. To reach this goal, we extend the language of Horn clauses, introducing a new kind of clauses, generalized Horn clauses, to be able to represent lists of any length. We consider a class of protocols that manipulate list elements in a uniform way. Because of this uniformity, one might intuitively think that secrecy for lists of length one implies secrecy for lists of any length. We show that this intuition is not exactly true: in general, secrecy for lists of length one does not imply secrecy for lists of any length, as demonstrated in Sect. 4.2. However, we show that, for a certain class of Horn clauses, if secrecy is proved by our Horn clause technique for lists of length one, then secrecy also holds for lists of unbounded length. This result relies on the sound abstractions made by the translation into Horn clauses. Additionally, we provide an approximation algorithm that can transform generalized Horn clauses into clauses of the class on which our result holds. All facts derivable from the initial clauses are also derivable from the clauses generated by the approximation algorithm, so that we can prove secrecy on the latter clauses, and conclude secrecy for the initial clauses. Our result therefore provides an easy way of obtaining a strong security guarantee: we prove using ProVerif that $\mathrm{att}(s)$ is not derivable from the clauses for lists of length one, and we can then immediately conclude that secrecy holds for lists of unbounded length, with an unbounded number of sessions.

Applications of our results include in particular proving secrecy properties for some group protocols that manipulate unbounded lists, with an unbounded number of participants. In this paper, we focus mainly on group protocols and illustrate our work on the Asokan-Ginzboorg protocol [2]. We prove secrecy of the session key exchanged in this protocol by verifying with ProVerif its version with lists of length one and the size of the group equal to one. Another possible application is the treatment of XML protocols such as web services, XML documents being modeled using possibly nested lists.

Related Work. The first approach considered for proving protocols with recursive data structures was interactive theorem proving: Paulson [17] and Bryans et al [5] study a recursive authentication protocol for an unbounded number of participants, using Isabelle/HOL for [17], and rank functions and PVS for [5]. However, this approach requires considerable human effort.

Meadows et al [15] used the NRL protocol analyzer (NPA), based on a combination of model checking and theorem-proving techniques, to verify the Group Domain of Interpretation (GDOI) protocol suite. NPA could not handle the infinite data structures required for modeling general group protocols, so a single key was used instead of a key hierarchy. Several problems including type flaw attacks were found in the protocol and fixed in later versions of GDOI. The early verification of the A.GDH-2 protocol using NPA [14] seems to have missed attacks [18], although the tool supports the Diffie-Hellman exponentiation [16].

Steel and Bundy [20] have used CORAL, a tool for finding counterexamples to incorrect inductive conjectures, to model protocols for group key agreement and group key management, without any restrictions on the scenario. They have discovered new attacks against several group protocols, but cannot prove that protocols are correct.

Kremer, Mercier, and Treinen [11] verify secrecy for group protocols with modular exponentiation and XOR, for any number of participants and an unbounded number of sessions, but only for a passive adversary (eavesdropper).

Several works consider the case of a bounded number of sessions. Pereira and Quisquater [18] discovered several attacks on the CLIQUES protocol suite [21], which extends the Diffie-Hellman key agreement method to support dynamic group operations (A-GDH). They converted the problem of the verification of security properties to the resolution of linear equation systems. In [19], they proved a first generic insecurity result for authentication protocols showing that it is impossible to design a correct authenticated group key agreement protocol based on the A-GDH. Truderung [22] showed a decidability result (in NEXPTIME) for secrecy in recursive protocols. This result was extended to a class of recursive protocols with XOR [13] in 3-NEXPTIME. Chridi et al [6,7] present an extension of the constraint-based approach in symbolic protocol verification to handle a class of protocols (Well-Tagged protocols with Autonomous keys) with unbounded lists in messages. They prove that the insecurity problem for Well-Tagged protocols with Autonomous keys is decidable for a bounded number of sessions.

We consider a class of protocols that includes the one of [6,7] but, instead of proving decidability for a bounded number of sessions, we provide a technique that can prove protocols for an unbounded number of sessions and any number of protocol participants, using abstractions.

Outline. The next section recalls the technique used by ProVerif. In Sect. 3, we formally define generalized Horn clauses, and their semantics by giving their translation into Horn clauses. Additionally, we introduce our running example and motivate the introduction of this new type of clauses. In Sect. 4, we show our main theorem: for a class of generalized Horn clauses, if $\mathrm{att}(s)$ is not derivable for lists of length one, then it is also not derivable for lists of any length. In Sect. 5, we provide an approximation algorithm for transforming generalized Horn clauses into clauses that satisfy the hypothesis of our main theorem. The proofs can be found in the long version of the paper available at `http://www.di.ens.fr/~paiola/publications/PaiolaBlanchetPOST12.html`.

2 A Reminder on ProVerif

ProVerif translates the initial protocol into a set of Horn clauses. The syntax of these clauses is defined in Fig. 1. The patterns represent messages that are

$p ::=$	patterns
x, y, z, v, w	variable
$a[p_1, \dots, p_n]$	name
$f(p_1, \dots, p_n)$	constructor application
$F ::= \mathrm{att}(p)$	facts
$R ::= F_1 \wedge \dots \wedge F_n \Rightarrow F$	Horn clause

Fig. 1. Syntax of Horn clauses

exchanged between participants of the protocol. A variable can represent any pattern. Names represent atomic values, such as keys and nonces. Each participant can create new names. Instead of creating a fresh name at each run of the protocol, the created names are considered as functions of the messages previously received by the principal that creates it, represented by the pattern $a[p_1, \dots, p_n]$. Hence names are distinguished only when they are created after receiving different messages. As shown in, e.g., [1], this is a sound approximation. When a name has no arguments, we write a instead of $a[\,]$. We use v, w, x, y, z for variables and other identifiers $a, b, c, e, L, pw, r, s, \dots$ for names.

The fact $\mathrm{att}(p)$ means that the attacker may have the pattern (message) p. A clause $F_1 \wedge \dots \wedge F_n \Rightarrow F$ means that if all facts F_i are true then the conclusion F is also true. We use R for a clause, H for its hypothesis, and C for its conclusion. The hypothesis of a clause is considered as a multiset of facts. A clause with no hypothesis $\Rightarrow F$ is written simply F.

Cryptographic primitives are represented by functions and perfect cryptography is assumed. There are two kinds of functions: constructors and destructors. A constructor f is a function that explicitly appears in the patterns that represent messages and builds new patterns of the form $f(p_1, \dots, p_n)$. Destructors manipulate patterns. A destructor g is defined by a set $\mathit{def}(g)$ of rewrite rules of the form $g(p_1, \dots, p_n) \rightarrow p$ where $p_1, \dots, p_n, p$ are patterns with only variables and constructors and the variables of p appear in $p_1, \dots, p_n$. Using constructors and destructors, one can represent data structures and cryptographic operations. For instance, $\mathit{senc}(x, y)$ is the constructor that represents the symmetric key encryption of the message x under the key y. The corresponding destructor $\mathit{sdec}(x', y)$ returns the decryption of x' if x' is a message encrypted under y. The rewrite rule that defines sdec is

$$\mathit{sdec}(\mathit{senc}(x, y), y) \rightarrow x.$$

A protocol is represented by three sets of Horn clauses:

1. initial knowledge of the attacker: we have a fact $\mathrm{att}(p)$ for each p initially known by the attacker.
2. abilities of the attacker:
   – $\mathrm{att}(a)$
   – for each constructor f of arity n:
   $\mathrm{att}(x_1) \wedge \dots \wedge \mathrm{att}(x_n) \Rightarrow \mathrm{att}(f(x_1, \dots x_n))$

– for each destructor g,
  for each rule $g(p_1, \ldots, p_n) \to p$ in $\mathit{def}(g)$:
    $\mathrm{att}(p_1) \wedge \cdots \wedge \mathrm{att}(p_n) \Rightarrow \mathrm{att}(p)$

The first clause represents the ability of the attacker to create fresh names a: all fresh names that the adversary may create are represented by the single name a. The other clauses mean that if the attacker has some messages, then he can apply constructors and destructors to them.

3. the protocol itself: for each message p of the protocol sent by agent A, we create the clause $\mathrm{att}(p_1) \wedge \cdots \wedge \mathrm{att}(p_n) \Rightarrow \mathrm{att}(p)$, where $p_1, \ldots, p_n$ are patterns representing the messages received by A before sending message p. Indeed, if the attacker has $p_1, \ldots, p_n$, then it can send them to A and intercept A's reply p.

This representation of protocols by Horn clauses is approximate, in particular because Horn clauses that represent the protocol itself can be applied any number of times instead of exactly once per session. However, it is sound: if $\mathrm{att}(p)$ cannot be derived from the clauses, then the protocol preserves the secrecy of p. (This is proved by [1, Theorem 7.2.3] when the clauses are generated from a pi calculus model of the protocol.)

ProVerif determines whether $\mathrm{att}(p)$ is derivable from the clauses using resolution with free selection [3]: we combine pairs of clauses by resolution; the literals upon which we perform resolution are chosen by a selection function. Next, we define when a given fact is derivable from a given set of clauses.

Definition 1 (Subsumption). *We say that $R_1 = H_1 \Rightarrow C_1$ subsumes $R_2 = H_2 \Rightarrow C_2$, and we write $R_1 \sqsupseteq R_2$, if and only if there exists a substitution σ such that $\sigma C_1 = C_2$ and $\sigma H_1 \subseteq H_2$ (multiset inclusion).*

We say that R_1 subsumes R_2 when R_2 can be obtained by adding hypotheses to a particular instance of R_1. In this case, all facts that can be derived by R_2 can also be derived by R_1, so R_2 can be eliminated.

Definition 2 (Derivability). *Let F be a closed fact, that is, a fact without variable. Let $\mathcal{R}$ be a set of clauses. F is derivable from $\mathcal{R}$ if and only if there exists a derivation of F from $\mathcal{R}$, that is, a finite tree defined as follows:*

1. *Its nodes (except the root) are labeled by clauses $R \in \mathcal{R}$;*
2. *Its edges are labeled by closed facts;*
3. *If the tree contains a node labeled R with one incoming edge labeled by F_0 and n outgoing edges labeled by $F_1, \ldots, F_n$, then $R \sqsupseteq F_1 \wedge \cdots \wedge F_n \Rightarrow F_0$.*
4. *The root has one outgoing edge labeled by F. The unique son of the root is named the subroot.*

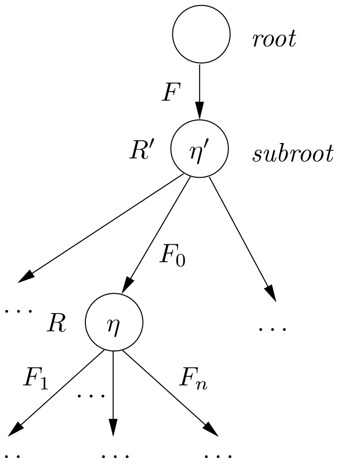

Fig. 2. Derivation of F

This definition is illustrated in Fig. 2. In a derivation, if there is a node labeled by R with one incoming edge labeled by F_0 and n outgoing edges $F_1, \dots, F_n$ then F_0 can be derived by $F_1, \dots, F_n$ by the clause R. Therefore there exists a derivation of F from $\mathcal{R}$ if and only if F can be derived from clauses in $\mathcal{R}$ (in classical logic).

3 Abstract Representation of Protocols by Generalized Horn Clauses

This section is devoted to the abstract representation of protocols by *generalized Horn clauses.* After introducing a running example and motivating our choices, we give the syntax and semantics of generalized Horn clauses.

3.1 Running Example

As a running example, we use a version of the Asokan-Ginzboorg protocol [2] for key agreement, also used in [7,20]. Let the set of players be $\{a_i, i = 1, \dots, N\}$ for $N \geq 1$ and L be the leader. The protocol describes the establishment of a session key between the leader and the other N participants.

$$\begin{array}{lll}
(1) & L \rightarrow ALL: & (L, \{\!|e|\!\}_{pw}) \\
(2) & a_i \rightarrow L: & (a_i, \{\!|(r_i, s_i)|\!\}_e) \\
(3) & L \rightarrow a_i: & \{\!|(s_1, \dots, s_N, s')|\!\}_{r_i} \\
(4) & a_i \rightarrow L: & (a_i, \{\!|(s_i, h(s_1, \dots, s_N, s'))|\!\}_K), \\
 & & \text{for some } i, \text{ where } K = f(s_1, \dots, s_N, s')
\end{array}$$

At the beginning, the participants share the knowledge of a password pw and of two $N+1$-input hash functions f and h. (In this paper, we ignore dictionary attacks against pw and consider pw as a strong key.) First, the leader sends to all other participants his identity paired with a fresh key e encrypted with the password pw. Each participant a_i for $i \in \{1, \dots, N\}$ decrypts $\{\!|e|\!\}_{pw}$ and then creates a fresh key r_i and a fresh nonce s_i which will be his contribution to the final session key. Then he sends $\{\!|(r_i, s_i)|\!\}_e$ paired with his identity. When L receives this message, he decrypts it and assumes that it has been created by a_i. After receiving all N messages, the leader creates his contribution s' to the final key and sends to each participant a_i for $i \in \{1, \dots, N\}$ the list of all contributions encrypted with the key r_i that a_i previously sent. If step 3 is completed successfully, each participant can compute the session key $K = f(s_1, \dots, s_N, s')$. In the end, the leader randomly picks one of the other players and asks him for step 4.

3.2 Need for Generalizing Horn Clauses

We would like to model the example protocol of Sect. 3.1 by Horn clauses and use ProVerif to verify it. Since we consider a parametric group size, we encounter several problems. First, we have to deal with lists whose length is not fixed but is the size N of the group, such as $s_1, \dots, s_N$ in message 3 of the example. Next, we need conjunctions of N facts (and N is again not fixed) to represent that some agents receive one message from each group member. For example, when translating message 3 into Horn clauses, the leader L expects messages 2 of the form $(a_i, \{\!|(v_i, w_i)|\!\}_e)$ from each a_i. (The leader cannot verify the incoming values of r_i, s_i so they become variables v_i, w_i.) Then L replies with message 3 $\{\!|(w_1, \dots, w_N, s')|\!\}_{v_i}$ where s' is a fresh name generated by L, modeled as a function of the previously received messages $s'[v_1, w_1, \dots, v_N, w_N]$. The attacker can send the incoming messages and intercept L's reply, so we find the clause

$$\begin{aligned}&\mathrm{att}((a_1, \mathit{senc}((v_1, w_1), e))) \wedge \cdots \wedge \mathrm{att}((a_N, \mathit{senc}((v_N, w_N), e))) \Rightarrow \\ &\quad \mathrm{att}(\mathit{senc}((w_1, \dots, w_N, s'[v_1, w_1, \dots, v_N, w_N]), v_i)).\end{aligned} \tag{1}$$

where *senc* is the encryption function. We solve those two problems by adding two new constructs to the syntax of Horn clauses: $\mathit{list}(i \le N, p_i)$ for the list of elements p_i with index i in the set $\{1, \dots, N\}$, that is, $\mathit{list}(i \le N, p_i)$ stands for $\langle p_1, \dots, p_N \rangle$ (inspired by the *mpair* construct of [7]) and $\bigwedge_{i_1 \le N, \dots, i_h \le N} F$ for the conjunction of facts F with indices $i_1, \dots, i_h$ in $\{1, \dots, N\}$.

3.3 Syntax

This section formally defines the syntax and semantics of *generalized Horn clauses.*

$p^G, s, t ::=$	patterns
$\quad x_{i_1,\dots,i_h}$	variable ($h \ge 0$)
$\quad f(p_1^G, \dots, p_l^G)$	function application
$\quad a_i[p_1^G, \dots, p_l^G]$	indexed names
$\quad \mathit{list}(i \le M, p^G)$	list constructor
$F^G ::= \bigwedge_{i_1 \le M_1, \dots, i_h \le M_h} \mathrm{att}(p^G)$	facts
$R^G ::= F_1^G \wedge \cdots \wedge F_n^G \Rightarrow \mathrm{att}(p^G)$	generalized Horn clause

Fig. 3. Syntax of our protocol representation

The syntax of these new clauses is defined in Fig. 3. The patterns p^G that represent messages are enriched with several new constructs. The variables may have indices $x_{i_1,\dots,i_h}$. The pattern for function application $f(p_1^G, \dots, p_l^G)$ includes not only constructor application but also names $a[p_1^G, \dots, p_l^G]$ where a is a name without index. The indexed name $a_i[p_1^G, \dots, p_l^G]$ represents a name created by the group member number i. We added a particular constructor $\mathit{list}(i \le M, p^G)$ to represent lists of length M, where M is an unknown bound.

In the Asokan-Ginzboorg protocol, we can write, for example, at message 3: $\mathit{senc}((\mathit{list}(j \le N, s_j), s'), r_i)$ for $\mathit{senc}((s_1, \dots, s_N, s'), r_i)$. The last element s'

is not included in the list $\mathit{list}(j \leq N, s_j)$, to distinguish s' that has just been created by the leader from s_i with $i = 1, \ldots, N$ that has just been received by him: $s_1, \ldots, s_N$ are treated in a uniform way while s' is treated differently.

We extend facts to model the possibility of having a conjunction of facts depending on indices, so that the syntax for facts becomes $\bigwedge_{i_1 \leq M_1, \ldots, i_h \leq M_h} \mathrm{att}(p^G)$. For example, intuitively, $\bigwedge_{i \leq M} \mathrm{att}(p^G)$ represents $\mathrm{att}(p^G\{i \mapsto 1\}) \wedge \cdots \wedge \mathrm{att}(p^G\{i \mapsto M\})$, where $p^G\{i \mapsto i'\}$ denotes p^G in which i has been replaced with i'. The conjunction $\bigwedge_{i_1 \leq M_1, \ldots, i_h \leq M_h}$ with $h = 0$ is omitted: the fact is then simply $\mathrm{att}(p^G)$.

The generalized Horn clause $F_1^G \wedge \cdots \wedge F_n^G \Rightarrow \mathrm{att}(p^G)$ means that, if the facts $F_1^G, \ldots, F_n^G$ hold, then the fact $\mathrm{att}(p^G)$ also holds. The conclusion of a clause does not contain a conjunction $\bigwedge_{i_1 \leq M_1, \ldots, i_h \leq M_h}$: we can simply leave the indices of $\mathrm{att}(p^G)$ free to mean that $\mathrm{att}(p^G)$ can be concluded for any value of these indices.

3.4 Representation of the Protocol

The representation of the abilities of the attacker includes the clauses given in Sect. 2. For our running example, $\mathrm{att}(a_i)$ and $\mathrm{att}(L)$ represent that the attacker initially knows a_i and L, and the clauses

$$\begin{array}{lll}
\mathrm{att}(a) & & \\
\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}(\mathit{senc}(x,y)) & \mathrm{att}(\mathit{senc}(x,y)) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}(x) & \\
\mathrm{att}(x) \Rightarrow \mathrm{att}(f(x)) & \mathrm{att}(x) \Rightarrow \mathrm{att}(h(x)) & \\
\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}((x,y)) & \mathrm{att}((x,y)) \Rightarrow \mathrm{att}(x) & \mathrm{att}((x,y)) \Rightarrow \mathrm{att}(y)
\end{array}$$

represent that the attacker can create fresh names, encrypt and decrypt messages, apply hash functions, compose and decompose pairs.

In addition, we have clauses for *list*, which generalize clauses for pairs:

$$\bigwedge_{i \leq M} \mathrm{att}(x_i) \Rightarrow \mathrm{att}(\mathit{list}(j \leq M, x_j)) \tag{2}$$

$$\mathrm{att}(\mathit{list}(j \leq M, x_j)) \Rightarrow \mathrm{att}(x_i) \tag{3}$$

Let us now give the clauses that represent the protocol itself. We suppose that each principal always plays the same role in the protocol; we could build a more complex model in which the same principal can play several roles by adding clauses. The leader L sends the first message $(L, \{\!|e|\!\}_{pw})$ and the attacker intercepts it, so we have the fact:

$$\mathrm{att}((L, \mathit{senc}(e, pw))).$$

Each agent a_i with $i = 1, \ldots, N$ expects a message 1 of the form $(L, \{\!|y|\!\}_{pw})$. (a_i cannot verify the value of the key e, so it becomes a variable y.) Agent a_i replies with message 2: $(a_i, \{\!|(r_i, s_i)|\!\}_y)$, where the new names r_i and s_i are encoded as functions of the key y just received. If the attacker sends the first message

$(L, \{\!|y|\!\}_{pw})$ to a_i, a_i replies with $(a_i, \{\!|(r_i, s_i)|\!\}_y)$, and the attacker can intercept this reply, so we obtain the clause:

$$\mathrm{att}((L, \mathit{senc}(y, pw))) \Rightarrow \mathrm{att}((a_i, \mathit{senc}((r_i[y], s_i[y]), y))) \tag{4}$$

For the output of message 3, the leader replies with $\{\!|(w_1, \ldots, w_N, s')|\!\}_{v_i}$ where s' is a fresh name generated by L modeled as a function of the previously received messages $s'[v_1, w_1, \ldots, v_N, w_N]$: the clause (1) was already given in Sect. 3.2; we adapt it using *list* and conjunctions over the set of participants:

$$\begin{aligned}&\textstyle\bigwedge_{j \le N} \mathrm{att}((a_j, \mathit{senc}((v_j, w_j), e))) \Rightarrow \\ &\qquad \mathrm{att}(\mathit{senc}((\mathit{list}(j \le N, w_j), s'[\mathit{list}(j \le N, (v_j, w_j))]), v_i))\end{aligned} \tag{5}$$

Finally, if a_i has received a message 1 of the form $(L, \{\!|y|\!\}_{pw})$ and a message 3 of the form $\{\!|(z_1, \ldots, z_N, z')|\!\}_{r_i[y]}$, encoded as $\{\!|(\mathit{list}(j \le N, z_j), z')|\!\}_{r_i[y]}$,[1] then a_i computes the session key $K = f((\mathit{list}(j \le N, z_j), z'))$ and one a_i sends to the leader message 4: $(a_i, \{\!|(s_i[y], h((\mathit{list}(j \le N, z_j), z')))|\!\}_K)$.

$$\begin{aligned}&\mathrm{att}((L, \mathit{senc}(y, pw))) \wedge \mathrm{att}(\mathit{senc}((\mathit{list}(j \le N, z_j), z'), r_i[y])) \Rightarrow \\ &\qquad \mathrm{att}((a_i, \mathit{senc}((s_i[y], h((\mathit{list}(j \le N, z_j), z'))), K))) \\ &\qquad\qquad \text{where } K = f((\mathit{list}(j \le N, z_j), z'))\end{aligned} \tag{6}$$

We want to prove the secrecy of the session key K. However, this key depends on data received by protocol participants, so we cannot simply test the derivability of $\mathrm{att}(K)$. We can use the following trick: to test the secrecy of the key K that participant a_i has, we consider that a_i sends the encryption $\{\!|s''|\!\}_K$ of a secret s'' under K. If K is secret, the adversary will not be able to decrypt the message, so s'' will remain secret. Therefore, we add the clause

$$\begin{aligned}&\mathrm{att}(\mathit{senc}((\mathit{list}(j \le N, z_j), z'), r_i[y])) \Rightarrow \\ &\qquad \mathrm{att}(\mathit{senc}(s'', f((\mathit{list}(j \le N, z_j), z'))))\end{aligned}$$

to model the output of $\{\!|s''|\!\}_K$, and we test the derivability of $\mathrm{att}(s'')$. We have also used a similar clause to prove the secrecy of the key K that L has.

3.5 Type System for the New Clauses

In Fig. 4, we define a simple type system for the generalized Horn clauses. The goal of this type system is to guarantee that all variables use indices that vary

[1] In the protocol, the participant a_i can check whether the component z_i of the list is his own contribution $s_i[y]$, but cannot check the other components. Our representation of lists does not allow us to model such a test: in fact, we cannot substitute a_i directly because, in the construct for lists $\mathit{list}(j \le N, z_j)$, all elements z_j need to have the same form. Moreover, we have built examples of protocols with such tests, for which our result does not hold: intuitively, the test breaks the uniform treatment of the elements of lists, so proving secrecy by the Horn clause technique for lists of length one does not imply secrecy for lists of unbounded length. We shall prove secrecy without the test on z_i; this implies a fortiori secrecy with this test, because the clause without test subsumes the one with the test. In general, removing these tests may obviously lead to false attacks.

$$\frac{i : [1,M] \in \Gamma}{\Gamma \vdash i : [1,M]}\text{(EnvIndex)} \qquad \frac{x\_ : [1,M_1] \times \cdots \times [1,M_h] \in \Gamma}{\Gamma \vdash x\_ : [1,M_1] \times \cdots \times [1,M_h]}\text{(EnvVar)}$$

$$\frac{\Gamma \vdash x\_ : [1,M_1] \times \cdots \times [1,M_h] \qquad \Gamma \vdash i_1 : [1,M_1] \ldots \Gamma \vdash i_h : [1,M_h]}{\Gamma \vdash x_{i_1,\ldots,i_h}}\text{(Var)}$$

$$\frac{\Gamma \vdash p_1^G \ldots \Gamma \vdash p_h^G}{\Gamma \vdash f(p_1^G,\ldots,p_h^G)}\text{(Fun)}$$

$$\frac{\Gamma \vdash p_1^G \ldots \Gamma \vdash p_h^G \qquad \Gamma \vdash i : [1,N]}{\Gamma \vdash a_i[p_1^G,\ldots,p_h^G]}\text{(Name)} \qquad \frac{\Gamma, i : [1,M] \vdash p^G}{\Gamma \vdash \mathit{list}(i \leq M, p^G)}\text{(List)}$$

$$\frac{\Gamma, i_1 : [1,M_1],\ldots,i_h : [1,M_h] \vdash p^G}{\Gamma \vdash \bigwedge_{i_1 \leq M_1,\ldots,i_h \leq M_h} \mathrm{att}(p^G)}\text{(Fact)}$$

$$\frac{\Gamma \vdash F_1^G \ldots \Gamma \vdash F_n^G \qquad \Gamma \vdash F^G}{\Gamma \vdash F_1^G \wedge \cdots \wedge F_n^G \Rightarrow F^G}\text{(Clause)}$$

Fig. 4. Type system for generalized Horn clauses

in the appropriate interval. We shall see in Sect. 4 that this type system is also very helpful in order to establish our main result.

Definition 3. *An index i is bound if:*

- *it appears as an index of a conjunction defining a fact, so, for instance, in the fact $\bigwedge_{i_1 \leq M_1,\ldots,i_h \leq M_h} \mathrm{att}(p^G)$, $i_1,\ldots,i_h$ are bound in $\mathrm{att}(p^G)$.*
- *it appears as an index for a list constructor, that is, in the pattern $\mathit{list}(i \leq M, p^G)$, i is bound in p^G.*

Indices that are not bound are free.

For simplicity, we suppose that the bound indices of clauses have pairwise distinct names, and names distinct from the names of free indices. This can easily be guaranteed by renaming the bound indices if needed.

In the type system, the type environment Γ is a list of type declarations:

- $i : [1,M]$ means that i is of type $[1,M]$, that is, intuitively, the value of index i can vary between 1 and the value of the bound M.
- $x\_ : [1,M_1] \times \cdots \times [1,M_h]$ means that the variable x expects indices of types $[1,M_1]$, ..., $[1,M_h]$.

The type system defines the judgments:

- $\Gamma \vdash i : [1,M]$, which means that i has type $[1,M]$ in environment Γ, by rule (EnvIndex);
- $\Gamma \vdash x\_ : [1,M_1] \times \cdots \times [1,M_h]$, which means that x expects indices of types $[1,M_1]$, ..., $[1,M_h]$ according to environment Γ, by rule (EnvVar);
- $\Gamma \vdash p^G$, $\Gamma \vdash F^G$, $\Gamma \vdash R^G$, which mean that p^G, F^G, R^G, respectively, are well typed in environment Γ.

Most type rules are straightforward. For instance, the rule (Var) means that $x_{i_1,\ldots,i_h}$ is well typed when the types expected by x for its indices match the types of $i_1, \ldots, i_h$. The rule (Name) deserves an additional explanation: we have no information in Γ to set the type of the index of name a, and hence the index of a can have any type. A priori, it is obviously expected that the index of a certain name a always has the same type. However, the additional freedom given by the type rule will be useful in the rest of the paper: the transformations of Sect. 5 can create clauses in which the same name a has indices of different types. The formal meaning of such clauses can be defined by assuming that the name a exists for indices up to the value of the largest bound.

It is easy to verify that the clauses of Sect. 3.4 are well typed in our type system. Clause (2) is well typed in the environment $x\_ : [1, M]$, (3) in the environment $x\_ : [1, M], i : [1, M]$, and the other clauses in the environment in which all free indices have type $[1, N]$ and the variables expect indices of type $[1, N]$.

3.6 Translation from Generalized Horn Clauses to Horn Clauses

A generalized Horn clause represents several Horn clauses: for each value of the bounds M and of the free indices i that occur in a generalized Horn clause R^G, R^G corresponds to a certain Horn clause. This section formally defines this correspondence.

$\overline{p} ::=$	patterns
$\quad x_{\bar{\imath}_1,\ldots,\bar{\imath}_h}$	variable
$\quad a_{\bar{\imath}}[\overline{p}_1, \ldots, \overline{p}_h]$	name
$\quad f(\overline{p}_1, \ldots, \overline{p}_h)$	constructor application
$\quad \langle \overline{p}_1, \ldots, \overline{p}_h \rangle$	list
$\overline{F} ::= \mathrm{att}(\overline{p})$	facts
$\overline{R} ::= \overline{F}_1 \wedge \ldots \wedge \overline{F}_n \Rightarrow \overline{F}$	Horn clauses

Fig. 5. Syntax of Horn clauses

The syntax of Horn clauses obtained by translation of generalized Horn clauses is given in Fig. 5. This syntax is similar to that of initial Horn clauses (Fig. 1) except that variables and names can now have indices $\bar{\imath}$, which are integer values, and that we include a pattern $\langle \overline{p}_1, \ldots, \overline{p}_h \rangle$ for representing lists (which will be generated by translation of *list*).

Definition 4. *Given a generalized Horn clause R^G well typed in Γ, an environment T for R^G is a function that associates to each bound M a fixed integer M^T and to each free index i that appears in R^G, an index $i^T \in \{1, \ldots, M^T\}$, if $\Gamma \vdash i : [1, M]$.*

Given an environment T and values $\bar{\imath}_1, \ldots, \bar{\imath}_h$, we write $T[i_1 \mapsto \bar{\imath}_1, \ldots, i_h \mapsto \bar{\imath}_h]$ for the environment that associates to indices $i_1, \ldots, i_h$ the values $\bar{\imath}_1, \ldots, \bar{\imath}_h$ respectively and that maps all other indices as in T.

Given an environment T, a generalized Horn clause R^G is translated into the standard Horn clause R^{GT} defined next. We denote respectively $p^{GT}, F^{GT}, \dots$ the translation of $p^G, F^G, \dots$ using the environment T.

The translation of a pattern p^G is defined as follows:

- $(x_{i_1,\dots,i_h})^T = x_{i_1^T,\dots,i_h^T}$.
- $f(p_1^G, \dots, p_l^G)^T = f(p_1^{GT}, \dots, p_l^{GT})$.
- $a_i[p_1^G, \dots, p_l^G]^T = a_{i^T}[p_1^{GT}, \dots, p_l^{GT}]$.
- $\mathit{list}(i \leq M, p^G)^T = \langle p^{GT[i\mapsto 1]}, \dots, p^{GT[i\mapsto M^T]}\rangle$.

The translation of *list* is a list; we stress that this translation uses a list symbol $\langle \dots \rangle$ different from the tuple symbol $(\dots)$: *list* is the only construct that can introduce the list symbol $\langle \dots \rangle$. This is important to make sure that confusions between tuples that may occur in the protocol and *list* do not occur for particular list lengths. In the implementation of the protocol, one must also make sure to use distinct encodings for *list* and for tuples.

The translation of a fact $F^G = \bigwedge_{i_1 \leq M_1,\dots,i_h \leq M_h} \mathrm{att}(p^G)$ is

$$F^{GT} = \mathrm{att}(p_1) \wedge \dots \wedge \mathrm{att}(p_k)$$

where $\{p_1, \dots, p_k\} = \{p^{GT'} \mid T' = T[i_1 \mapsto \bar{\imath}_1, \dots, i_h \mapsto \bar{\imath}_h]$ where $\bar{\imath}_j \in \{1, \dots, M_j^T\}$ for all j in $\{1, \dots, h\}\}$, and $(F_1^G \wedge \dots \wedge F_n^G)^T = F_1^{GT} \wedge \dots \wedge F_n^{GT}$.

Finally, we define the translation of the generalized Horn clause $R^G = H^G \Rightarrow \mathrm{att}(p^G)$ as $R^{GT} = H^{GT} \Rightarrow \mathrm{att}(p^{GT})$.

For instance, the translation of the clause (5) in the environment $T = \{N \mapsto 1, i \mapsto 1\}$ is $\mathrm{att}((a_1, \mathit{senc}((v_1, w_1), e))) \Rightarrow \mathrm{att}(\mathit{senc}((\langle w_1\rangle, s'[\langle (v_1, w_1)\rangle]), v_1))$.

When $\mathcal{R}^G$ is a set of generalized Horn clauses, we define $\mathcal{R}^{G\mathcal{T}} = \{R^{GT} \mid R^G \in \mathcal{R}^G,\ T$ is an environment for $R^G\}$. In terms of abstract interpretation, the sets of generalized Horn clauses ordered by inclusion constitute the abstract domain, the sets of Horn clauses ordered by inclusion the concrete domain, and $\mathcal{R}^{G\mathcal{T}}$ is the concretization of $\mathcal{R}^G$. The set $\mathcal{R}^{G\mathcal{T}}$ includes clauses translated for any values of the bounds. In our running example, for instance, this allows one to consider several sessions of the protocol that have different group sizes N, and interactions between such sessions.

4 From Any Length to Length One

In this section, we define a mapping from lists of any length to lists of length one, and show that derivability for lists of any length implies derivability for lists of length one, for a particular class of Horn clauses.

4.1 Main Result

Given a generalized Horn clause R^G, there exists only one environment T for R^G such that all bounds are equal to 1. Hence by now we use $R^{G\mathit{1}}$ for the only possible translation of R^G when all bounds are 1. We define $\mathcal{R}^{G1} = \{R^{G1} \mid R^G \in \mathcal{R}^G\}$.

Next, we define a translation from clauses in which bounds can have any value, following the syntax of Fig. 5, to clauses in which the bounds are fixed to 1:

$$\mathbb{I}(\overline{p}) = \begin{cases} \{x_{\underbrace{1,\dots,1}_{h}}\} & \text{if } \overline{p} = x_{\overline{\imath}_1,\dots,\overline{\imath}_h} \\ \{f(p'_1,\dots,p'_h) \mid p'_1 \in \mathbb{I}(\overline{p}_1),\dots,p'_h \in \mathbb{I}(\overline{p}_h)\} & \text{if } \overline{p} = f(\overline{p}_1,\dots,\overline{p}_h) \\ \{a_1[p'_1,\dots,p'_h] \mid p'_1 \in \mathbb{I}(\overline{p}_1),\dots,p'_h \in \mathbb{I}(\overline{p}_h)\} & \text{if } \overline{p} = a_{\overline{\imath}}[\overline{p}_1,\dots,\overline{p}_h] \\ \{\langle p\rangle \mid p \in \mathbb{I}(\overline{p}_1) \cup \dots \cup \mathbb{I}(\overline{p}_h)\} & \text{if } \overline{p} = \langle \overline{p}_1,\dots,\overline{p}_h\rangle \end{cases}$$

This translation maps all indices of variables and names to 1. The translation of a list is a list with one element, containing the translation of any element of the initial list. Several choices are possible for the translation of a list; $\mathbb{I}(\overline{p})$ returns the set of all possible patterns.

Given a fact $\overline{F} = \text{att}(\overline{p})$, its translation when the bounds are fixed to 1 is $\mathbb{I}(\text{att}(\overline{p})) = \{\text{att}(p) \mid p \in \mathbb{I}(\overline{p})\}$ Given a conjunction of facts $\overline{F}_1 \wedge \dots \wedge \overline{F}_h$, its translation when the bounds are fixed to 1 is $\mathbb{I}(\overline{F}_1 \wedge \dots \wedge \overline{F}_h) = \mathbb{I}(\overline{F}_1) \cup \dots \cup \mathbb{I}(\overline{F}_h)$.

We say that a term or fact is *linear* when it contains at most one occurrence of each variable $x\_$ (with any indices, so it cannot contain x_i and x_j for instance). Finally, we can state the main theorem of our paper:

Theorem 1. *Let $\mathcal{R}^G$ be a set of generalized Horn clauses such that, for each clause $R^G \in \mathcal{R}^G$, R^G is well typed, that is, there exists Γ such that $\Gamma \vdash R^G$, with the following conditions:*

1. *the free indices of R^G have pairwise distinct types in Γ;*
2. *the conclusion of R^G is linear and the bound indices in the conclusion of R^G have pairwise distinct bounds, and bounds different from the bounds of free indices of R^G in Γ.*

For all facts $\overline{F}$, if $\overline{F}$ is derivable from $\mathcal{R}^{G\mathcal{T}}$, then for all $F \in \mathbb{I}(\overline{F})$, F is derivable from $\mathcal{R}^{G1}$.

If we show that, for some $F \in \mathbb{I}(\overline{F})$, F is not derivable from $\mathcal{R}^{G1}$, then using this theorem, $\overline{F}$ is not derivable from $\mathcal{R}^{G\mathcal{T}}$. Suppose that we want to show that s is secret in a protocol represented by the clauses $\mathcal{R}^G$. We show using for instance ProVerif that att(s) is not derivable from $\mathcal{R}^{G1}$, that is, we prove secrecy when the bounds are all fixed to 1. By Theorem 1, we conclude that att(s) is not derivable from $\mathcal{R}^{G\mathcal{T}}$, so we obtain secrecy for any bounds.

Unfortunately, this theorem does not apply to all Horn clauses: Hypotheses 1 and 2 have to be satisfied. The clauses of our running example do not satisfy these hypotheses. We shall see in Sect. 5 how to transform the clauses so that they satisfy the required hypotheses.

4.2 Examples

To illustrate why the hypotheses of the theorem are necessary, we provide examples for which the theorem does not hold because some hypotheses are not satisfied. Consider the following protocol:

(1) $A \rightarrow B : \{(a,a)\}_k$
(2) $B \rightarrow A : (\{(b,b)\}_k, \{s\}_{f(a,b)})$
(3) $A \rightarrow C : \langle \{(a_1,a'_1)\}_k, \dots, \{(a_N,a'_N)\}_k \rangle$
(4) $C \rightarrow A : \langle \langle f(a_1,a'_1), \dots, f(a_1,a'_N) \rangle, \dots, \langle f(a_N,a'_1), \dots, f(a_N,a'_N) \rangle \rangle$

At the beginning, the participants A, B, C share a key k. A first sends to B a fresh nonce a paired with itself and encrypted under k. When B receives it, he creates a fresh nonce b, computes the hash $f(a,b)$ and sends the pair $(\{(b,b)\}_k, \{s\}_{f(a,b)})$, where s is some secret. A can then decrypt $\{(b,b)\}_k$, obtain b, compute $f(a,b)$, decrypt $\{s\}_{f(a,b)}$, and obtain s, but an adversary should be unable to compute s. In the second part of the protocol (Messages 3 and 4), A sends to C a list of N fresh pairs (a_i,a'_i) encrypted with k and C replies with the matrix of the hashes $f(a_i,a'_j)$.

Now, if an attacker sends $\langle \{(a,a)\}_k, \{(b,b)\}_k \rangle$ to C as Message 3, he obtains $f(a,b)$ by decomposition of the list $\langle \langle f(a,a), f(a,b) \rangle, \langle f(b,a), f(b,b) \rangle \rangle$ and can now decrypt $\{s\}_{f(a,b)}$ and obtain the secret s.

However, if we consider only lists of one element, there is no attack: the last message consists of $\langle \langle f(a,a') \rangle \rangle$ if Message 3 was $\{(a,a')\}_k$, so the adversary would need to have $\{(a,b)\}_k$ in order to obtain $f(a,b)$.

The generalized Horn clause for Message 4 is:

$$\mathrm{att}(\mathit{list}(i' \leq N, \mathit{senc}((x_{i'},y_{i'}),k))) \Rightarrow \mathrm{att}(\mathit{list}(i \leq N, \mathit{list}(j \leq N, f(x_j,y_i))))$$

In this clause, the Hypothesis 2 of Theorem 1 is not satisfied, because the bound indices i and j have the same bound N. If we translate this clause for lists of one element, we obtain

$$\mathrm{att}(\langle \mathit{senc}((x_1,y_1),k) \rangle) \Rightarrow \mathrm{att}(\langle \langle f(x_1,y_1) \rangle \rangle)$$

and with this clause (and other clauses representing this protocol), $\mathrm{att}(s)$ is not derivable because $\mathrm{att}(f(a,b))$ is not derivable, while with lists of length two, as we previously showed, there is an attack: $\mathrm{att}(s)$ is derivable. This example confirms that bound indices in the conclusion must have pairwise distinct bounds.

Similarly, we can define a group protocol between a participant B, a leader L, and N group members A_i:

(1) $L \rightarrow B : \{(a,a)\}_p$
(2) $B \rightarrow L : (\{(b,b)\}_p, \{s\}_{f(a,b)})$
(3) $L \rightarrow A_i : \langle \{(a_1,a'_1)\}_p, \dots, \{(a_N,a'_N)\}_p \rangle$
(4) $A_i \rightarrow L : \langle f(a_1,a'_i), \dots, f(a_N,a'_i) \rangle$

In this case, the generalized Horn clause for Message 4 is:

$$\mathrm{att}(\mathit{list}(i' \leq N, \mathit{senc}((x_{i'},y_{i'}),p))) \Rightarrow \mathrm{att}(\mathit{list}(j \leq N, f(x_j,y_i)))$$

where again the Hypothesis 2 of Theorem 1 is not satisfied: the bound index j has the same bound N as the free index i, because they index the same variable $x\_$. As above, $\mathrm{att}(s)$ is derivable from the clauses for lists of length 2 but not for lists of length one. There are similar examples regarding Hypothesis 1, for instance with the clause

$$\mathrm{att}(\mathit{list}(i' \leq N, \mathit{senc}((x_{i'}, y_{i'}), p))) \Rightarrow \mathrm{att}(f(x_j, y_i))$$

in which the free indices i and j have the same type $[1, N]$, but it is more difficult to find a concrete protocol that would generate such a clause. (Typically, the protocol participants are indexed by a single index i, so clauses often have a single free index.)

Next, we consider a different kind of example: for the following protocol, the set of Horn clauses satisfies the hypothesis of Theorem 1, so we can apply the theorem. However, the protocol preserves secrecy for lists of length one but not for lists of unbounded length. This illustrates that the approximations made in the translation to Horn clauses are key for our theorem to hold: $\mathrm{att}(s)$ is derivable from the clauses, even for lists of length one. Let A and B be the two participants of the protocol that share a key k. Let h be a hash function.

$$\begin{array}{l}
(1)\ A \to B : \{e\}_k, (b_1, b_2), \{s\}_{h(\{b_1\}_e, \{b_2\}_e)} \\
(2)\ B \to A : \langle x_1, \dots, x_M \rangle \\
(3)\ A \to B : \langle \{x_1\}_e, \dots, \{x_M\}_e \rangle
\end{array}$$

A chooses a fresh key e and two random nonces b_1, b_2, and sends to B the message $\{e\}_k, (b_1, b_2), \{s\}_{h(\{b_1\}_e, \{b_2\}_e)}$ where s is a secret. B obtains e by decryption, computes the key $h(\{b_1\}_e, \{b_2\}_e)$, and obtains s by decrypting with this key. Later, B sends a list $\langle x_1, \dots, x_M \rangle$ and A returns that list with all components encrypted under e. Clearly, if we consider this protocol for lists of length $M \geq 2$, there is an attack: the attacker can send to A the list $\langle b_1, b_2, \dots \rangle$ and he obtains at Message 3 the list $\langle \{b_1\}_e, \{b_2\}_e, \{\dots\}_e \rangle$. He can then compute the hash $h(\{b_1\}_e, \{b_2\}_e)$ and decrypt $\{s\}_{h(\{b_1\}_e, \{b_2\}_e)}$ to obtain the secret s. However, if we translate this protocol to lists of length one, we do not find the attack: the attacker can only ask for $\langle \{b_1\}_e \rangle$ or $\langle \{b_2\}_e \rangle$, but cannot obtain both. For this point to hold, it is important that the participants do not repeat the Messages 2-3 more than once for each session.

ProVerif finds an attack against this protocol (which is a false attack for lists of length one): the abstraction done with the representation by Horn clauses in fact allows the participants to repeat their messages more than once. The translation of the protocol into clauses for lists of length one contains:

A sends the first message:

$$\mathrm{att}((\mathit{senc}(e, k), (b_1, b_2), \mathit{senc}(s, h(\mathit{senc}(b_1, e), \mathit{senc}(b_2, e))))) \tag{7}$$

A receives message 2 and sends message 3:

$$\mathrm{att}(\langle x \rangle) \Rightarrow \mathrm{att}(\langle \mathit{senc}(x, e) \rangle) \tag{8}$$

plus clauses for tuples, encryption, and the hash function h, where $\langle \cdot \rangle$ is a unary function such that $\mathrm{att}(\langle x \rangle) \Rightarrow \mathrm{att}(x)$ and $\mathrm{att}(x) \Rightarrow \mathrm{att}(\langle x \rangle)$. Now, if we query for the secrecy of s, ProVerif will find the attack: att(s) is derivable from these clauses. Indeed, we get b_1 and b_2 from (7), then obtain $senc(b_1, e)$ and $senc(b_2, e)$ by two applications of (8) (note that we apply this clause twice for the same e, while the corresponding action can in fact be applied only once in the protocol itself), then compute $h(senc(b_1, e), senc(b_2, e))$, and finally obtain s by decrypting $senc(s, h(senc(b_1, e), senc(b_2, e)))$.

4.3 Proof of Theorem 1

This section sketches the proof of Theorem 1. Lemmas and details of the proof can be found in the long version of the paper. The proof proceeds by building a derivation of F from $\mathcal{R}^{G1}$, from a derivation of $\overline{F}$ from $\mathcal{R}^{G\mathcal{T}}$, by induction on this derivation. Informally, the derivation of F from $\mathcal{R}^{G1}$ is obtained by applying $\mathbb{I}$ to the derivation of $\overline{F}$ from $\mathcal{R}^{G\mathcal{T}}$. If $\overline{F}$ is derived by $R^{GT} = H^{GT} \Rightarrow C^{GT}$, $\overline{F}$ is an instance of C^{GT} by a substitution σ: $\overline{F} = \sigma C^{GT}$; we show that any $F \in \mathbb{I}(\overline{F})$ is an instance of C^{G1} by a substitution σ' obtained from σ: $F = \sigma' C^{G1}$. Hence, in order to derive F using $R^{G1} = H^{G1} \Rightarrow C^{G1}$, we need to derive $\sigma' H^{G1}$ from $\mathcal{R}^{G1}$, knowing a derivation of σH^{GT} from $\mathcal{R}^{G\mathcal{T}}$. Informally, to show that this is possible, we prove that $\sigma' H^{G1} \subseteq \mathbb{I}(\sigma H^{GT})$ and conclude by induction.

5 An Approximation Algorithm

In Sect. 3.4, we gave the representation of the Asokan-Ginzboorg protocol with generalized Horn clauses. However, some of them do not satisfy the hypotheses of Theorem 1. For example, the clause (6) does not have a linear conclusion and the same bound appears twice in the conclusion.

5.1 Approximation Algorithm

Here we give an algorithm for transforming generalized Horn clauses into clauses that satisfy the hypothesis of Theorem 1. We suppose that the initial set of clauses $\mathcal{R}^G$ satisfies:

Hypothesis 1. *For each clause $R^G \in \mathcal{R}^G$, R^G is well-typed, that is, there exists Γ such that $\Gamma \vdash R^G$, and each variable has indices of pairwise distinct types, that is, if $\Gamma \vdash x\_ : [1, N_1] \times \dots, \times [1, N_h]$, then $N_1, \dots, N_h$ are pairwise distinct.*

This hypothesis on the initial clauses is often satisfied in practice. In particular, it is satisfied by our running example, and it should generally be satisfied by group protocols. Indeed, the variables typically have only one index (the number of the group member).

Given a clause R^G well typed in Γ, the approximation algorithm performs the following three steps, until it reaches a fixpoint:

1. Suppose $R^G = H^G \Rightarrow \text{att}(p^G)$, where H^G contains a free index i such that $\Gamma \vdash i : [1, N]$ and p^G contains a bound index j with bound N, or R^G contains two free indices i, j such that $\Gamma \vdash i : [1, N]$ and $\Gamma \vdash j : [1, N]$.

The algorithm chooses a fresh variable $y\_ = \rho x\_$ for each variable $x\_$ that occurs in R^G with index i, and replaces all occurrences of variables $x\_$ that have index i with $\rho x\_$ (the indices remain the same).
The obtained clause can then be typed in an environment Γ' equal to Γ except that $\Gamma' \vdash i : [1, M]$ for some fresh bound M and that $\Gamma' \vdash y\_ : [1, M_1] \times \cdots \times [1, M_h]$ if $y\_ = \rho x\_$, $\Gamma \vdash x\_ : [1, N_1] \times \cdots \times [1, N_h]$, and for each $k = 1, \ldots, h$, $M_k = N_k$ if $N_k \neq N$ and $M_k = M$ if $N_k = N$. The indices i and j then have different types in the obtained clause.

2. Suppose $R^G = H_1^G \wedge H_2^G \Rightarrow \mathrm{att}(p^G)$, where p^G contains a pattern $\mathit{list}(i \leq N, p_1^G)$ as well as a pattern $\mathit{list}(j \leq N, p_2^G)$ or a free index j such that $\Gamma \vdash j : [1, N]$, H_1^G contains all hypotheses of R^G in which the bound N appears or a free index of type $[1, N]$ appears, and H_2^G contains the other hypotheses of R^G.
The algorithm chooses a fresh bound M and replaces R^G with
$$H_1^G \wedge H_1'^G \wedge H_2^G \Rightarrow \mathrm{att}(p'^G)$$
where:
   - ρ is a substitution that replaces each variable $x\_$ of H_1^G and p_1^G such that $\Gamma \vdash x\_ : [1, N_1] \times \cdots \times [1, N_h]$ with $N_k = N$ for some $k \in \{1, \ldots, h\}$ with a fresh variable $y\_$ (the indices remain the same); the obtained clause will be typed in an environment Γ' obtained from Γ by adding $\Gamma' \vdash y\_ : [1, M_1] \times \cdots \times [1, M_h]$ where, for each $k = 1, \ldots, h$, $M_k = N_k$ if $N_k \neq N$ and $M_k = M$ if $N_k = N$;
   - $H_1'^G$ is obtained from ρH_1^G by replacing the bound N with M;
   - p'^G is obtained from p^G by replacing $\mathit{list}(i \leq N, p_1^G)$ with $\mathit{list}(i \leq M, p_1'^G)$, where $p_1'^G$ is p_1^G in which all occurrences of variables $x\_$ that have index i have been replaced with $\rho x\_$.
3. Suppose $R^G = H_1^G \wedge H_2^G \Rightarrow \mathrm{att}(p^G)$ where p^G contains at least two occurrences of a variable $x\_$, H_1^G contains all hypotheses of R^G in which $x\_$ appears, and H_2^G contains the other hypotheses of R^G.
The algorithm chooses a fresh variable $y\_$ and replaces R^G with
$$H_1^G \wedge H_1'^G \wedge H_2^G \Rightarrow \mathrm{att}(p'^G)$$
where $H_1'^G$ is obtained from H_1^G by replacing each occurence of $x\_$ with $y\_$ (the indices remain the same), and p'^G is obtained from p^G by replacing one occurrence of $x\_$ with $y\_$.

Step 1 is applied first, until it cannot be applied. Then step 2 is applied, until there are no list constructors that match the condition. Step 2 may already rename some variables that occur more than once in the conclusion of the clause. Then, when a fixpoint is reached with step 2, we start applying step 3, until no variable occurs more than once in the conclusion. Step 1 ensures that free indices have pairwise distinct types and that free indices of the hypothesis have types distinct from those of bound indices in the conclusion. Step 2 ensures that the bound indices in the conclusion have pairwise distinct bounds and bounds distinct from the bounds of free indices in the conclusion. Step 3 ensures that the conclusion is linear.

This algorithm is similar to the algorithm that transforms any Horn clauses into Horn clauses of the class $\mathcal{H}_1$ [10]. Both algorithms ensure the linearity of the conclusion in the same way (step 3). Step 2 uses an idea similar to step 3 to guarantee that the types of the indices are distinct.

We illustrate this algorithm on an example below. The next theorem shows its correctness. Its proof can be found in the long version of the paper.

Theorem 2. *Let $\mathcal{R}^G$ be a set of clauses that satisfies Hypothesis 1. The approximation algorithm terminates on $\mathcal{R}^G$ and the final set of clauses $\mathcal{R}'^G$ satisfies the hypothesis of Theorem 1. Moreover, for any fact F, if F is derivable from $\mathcal{R}^{G\mathcal{T}}$, then F is also derivable from $\mathcal{R}'^{G\mathcal{T}}$.*

5.2 Examples

We apply the approximation algorithm to our running example. For instance, let us transform the clause (6):

$$\begin{aligned}&\mathrm{att}((L, \mathit{senc}(y, \mathit{pw}))) \wedge \mathrm{att}(\mathit{senc}((\mathit{list}(j \leq N, z_j), z'), r_i[y]))\\ &\Rightarrow \mathrm{att}((a_i, \mathit{senc}((s_i[y], h((\mathit{list}(j \leq N, z_j), z'))), f((\mathit{list}(j \leq N, z_j), z')))))\end{aligned}$$

First, as there are two list constructors with the same bound N in the conclusion, we apply step 2 of the algorithm: we rename the bound and variables of one of the two occurrences of $\mathit{list}(j \leq N, z_j)$ in the conclusion, so we obtain:

$$\begin{aligned}&\mathrm{att}((L, \mathit{senc}(y, \mathit{pw}))) \wedge\\ &\mathrm{att}(\mathit{senc}((\mathit{list}(j \leq N, z_j), z'), r_i[y])) \wedge \mathrm{att}(\mathit{senc}((\mathit{list}(j \leq M, x_j), z'), r_i[y]))\\ &\Rightarrow \mathrm{att}((a_i, \mathit{senc}((s_i[y], h((\mathit{list}(j \leq N, z_j), z'))), f((\mathit{list}(j \leq M, x_j), z')))))\end{aligned}$$

Next, as variable z' appears twice in the conclusion, we apply step 3 and obtain:

$$\begin{aligned}&\mathrm{att}((L, \mathit{senc}(y, \mathit{pw}))) \wedge\\ &\mathrm{att}(\mathit{senc}((\mathit{list}(j \leq N, z_j), z'), r_i[y])) \wedge \mathrm{att}(\mathit{senc}((\mathit{list}(j \leq M, x_j), z'), r_i[y]))\\ &\mathrm{att}(\mathit{senc}((\mathit{list}(j \leq N, z_j), x'), r_i[y])) \wedge \mathrm{att}(\mathit{senc}((\mathit{list}(j \leq M, x_j), x'), r_i[y]))\\ &\Rightarrow \mathrm{att}((a_i, \mathit{senc}((s_i[y], h((\mathit{list}(j \leq N, z_j), z'))), f((\mathit{list}(j \leq M, x_j), x')))))\end{aligned}$$

Finally, this clause satisfies the hypothesis of Theorem 1. All clauses $\mathcal{R}^G$ given in Sect. 3.4, which represent our running example, can be transformed in a similar way, yielding clauses $\mathcal{R}'^G$. We have then shown that $\mathrm{att}(s)$ is not derivable from $\mathcal{R}'^{G1}$, using ProVerif with the input file given at `http://www.di.ens.fr/~paiola/publications/PaiolaBlanchetPOST12.html`. By Theorem 1, we conclude that $\mathrm{att}(s)$ is not derivable from $\mathcal{R}'^{G\mathcal{T}}$, so by Theorem 2, $\mathrm{att}(s)$ is not derivable from $\mathcal{R}^{G\mathcal{T}}$. Therefore, the Azokan-Ginzboorg protocol preserves the secrecy of s, that is, it preserves the secrecy of the key K that a_i has. We have shown in a similar way that it preserves the secrecy of the key K that L has.

We have also considered a basic XML encryption [9] protocol. It is a very simple protocol between two principals A and B that share an encryption key

k and a MAC key k'. In order to encrypt a list $\langle a_1, \ldots, a_M \rangle$ using the encrypt-then-MAC scheme, one encrypts each component of the list, and computes a MAC of the list of ciphertexts:

$$A \rightarrow B: \ (\langle \{a_1\}_k, \ldots, \{a_M\}_k \rangle, mac(k', \langle sha1(\{a_1\}_k), \ldots, sha1(\{a_M\}_k) \rangle)).$$

We have used ProVerif to show that $\mathrm{att}(a_1)$ is not derivable from the set of Horn clauses for the protocol with lists of length one. Therefore, this protocol preserves the secrecy of each a_j, for $j = 1, \ldots, M$.

6 Conclusions and Future Work

We have proposed a new type of clauses, generalized Horn clauses, useful to represent protocols that manipulate lists of unbounded length, as well as group protocols with an unbounded number of participants. We have shown that, for a subclass of generalized Horn clauses, if secrecy is proved by the Horn clause technique for lists of length one, then we have secrecy for lists of any length. We have also provided an approximation algorithm that transforms a set of generalized Horn clauses for satisfying the hypothesis of our main theorem. Using these results, one can prove secrecy for lists of any length for some group protocols, as we did for the Azokan-Ginzboorg protocol, and for simple XML protocols.

Future work includes supporting more general data structures and protocols, including more realistic XML protocols (web services). This will probably require a new extension of Horn clauses and of the resolution algorithm, since these protocols may not fit in a class for which secrecy for lists of any length can be proved from underivability for lists of length one. In particular, as we mentioned in Sect. 3.4, our technique does not support equality tests on certain components of lists, because in the representation of unbounded lists, all elements need to have the same form. We plan to support such tests in the future. Moreover, some group protocols (e.g. A.GDH-2) use the Diffie-Hellman key agreement, which we cannot handle yet. We believe that it could be handled by combining our result with [12].

ProVerif supports a variant of the applied pi calculus for modeling protocols. However, our result models group protocols with generalized Horn clauses. We plan to extend the input language of ProVerif to model group protocols, and to translate it automatically to generalized Horn clauses.

Finally, we plan to consider other security properties, such as authentication, perhaps using lists of length two instead of one.

Acknowledgments. This work was partly supported by the ANR project ProSe (decision number ANR-2010-VERS-004-01).

References

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

2. Asokan, N., Ginzboorg, P.: Key agreement in ad hoc networks. Computer Communications 23(17), 1627–1637 (2000)
3. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Handbook of Automated Reasoning, vol. 1, ch. 2, pp. 19–100. North Holland (2001)
4. Blanchet, B.: Using Horn clauses for analyzing security protocols. In: Cortier, V., Kremer, S. (eds.) Formal Models and Techniques for Analyzing Security Protocols. Cryptology and Information Security Series, vol. 5, pp. 86–111. IOS Press, Amsterdam (2011)
5. Bryans, J., Schneider, S.: CSP, PVS and recursive authentication protocol. In: DIMACS Workshop on Formal Verification of Security Protocols (1997)
6. Chridi, N., Turuani, M., Rusinowitch, M.: Constraints-based Verification of Parameterized Cryptographic Protocols. Research Report RR-6712, INRIA (2008), `http://hal.inria.fr/inria-00336539/en/`
7. Chridi, N., Turuani, M., Rusinowitch, M.: Decidable analysis for a class of cryptographic group protocols with unbounded lists. In: CSF 2009, pp. 277–289. IEEE, Los Alamitos (2009)
8. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory IT-29(12), 198–208 (1983)
9. Eastlake, D., Reagle, J.: XML encryption syntax and processing. W3C Candidate Recommendation (2002), `http://www.w3.org/TR/2002/CR-xmlenc-core-20020802/`
10. Goubault-Larrecq, J.: Une fois qu'on n'a pas trouvé de preuve, comment le faire comprendre à un assistant de preuve? In: JFLA 2004, pp. 1–20. INRIA (2004)
11. Kremer, S., Mercier, A., Treinen, R.: Proving Group Protocols Secure Against Eavesdroppers. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 116–131. Springer, Heidelberg (2008)
12. Küsters, R., Truderung, T.: Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In: CSF 2009, pp. 157–171. IEEE, Los Alamitos (2009)
13. Küsters, R., Truderung, T.: On the Automatic Analysis of Recursive Security Protocols with XOR. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 646–657. Springer, Heidelberg (2007)
14. Meadows, C.: Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In: WITS 2000 (2000)
15. Meadows, C., Syverson, P., Cervesato, I.: Formal specification and analysis of the Group Domain of Interpretation protocol using NPATRL and the NRL protocol analyzer. Journal of Computer Security 12(6), 893–931 (2004)
16. Meadows, C., Narendran, P.: A unification algorithm for the group Diffie-Hellman protocol. In: WITS 2002 (2002)
17. Paulson, L.C.: Mechanized proofs for a recursive authentication protocol. In: CSFW 1997, pp. 84–95. IEEE, Los Alamitos (1997)
18. Pereira, O., Quisquater, J.J.: Some attacks upon authenticated group key agreement protocols. Journal of Computer Security 11(4), 555–580 (2003)
19. Pereira, O., Quisquater, J.J.: Generic insecurity of cliques-type authenticated group key agreement protocols. In: CSFW 2004, pp. 16–19. IEEE, Los Alamitos (2004)
20. Steel, G., Bundy, A.: Attacking group protocols by refuting incorrect inductive conjectures. Journal of Automated Reasoning 36(1-2), 149–176 (2006)
21. Steiner, M., Tsudik, G., Waidner, M.: CLIQUES: A new approach to group key agreement. In: ICDCS 1998, pp. 380–387. IEEE, Los Alamitos (1998)
22. Truderung, T.: Selecting Theories and Recursive Protocols. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 217–232. Springer, Heidelberg (2005)