Automatic Verification of Protocols with Lists of Unbounded Length

Bruno Blanchet
INRIA Paris-Rocquencourt, France
Bruno.Blanchet@inria.fr

Miriam Paiola
INRIA Paris-Rocquencourt, France
Miriam.Paiola@inria.fr

ABSTRACT

We present a novel automatic technique for proving secrecy and authentication properties for security protocols that manipulate lists of unbounded length, for an unbounded number of sessions. This result is achieved by extending the Horn clause approach of the automatic protocol verifier ProVerif. We extend the Horn clauses to be able to represent lists of unbounded length. We adapt the resolution algorithm to handle the new class of Horn clauses, and prove the soundness of this new algorithm. We have implemented our algorithm and successfully tested it on several protocol examples, including XML protocols coming from web services.

Categories and Subject Descriptors

C.2.2 [**Computer-communication networks**]: Network Protocols—*Protocol verification*

Keywords

Security protocols; verification; lists; web services; Horn clauses; resolution.

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. However, attacks are 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 [13], 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, some protocols, for instance XML protocols of web services, use more complex data structures, such as lists. The verification of protocols that manipulate such data structures has been less studied and presents additional difficulties: these complex data structures add another cause of undecidability.

In this work, we extend the verifier ProVerif [7] to protocols with lists of unbounded length. ProVerif is an automatic verifier that takes as input a protocol, translates it into a representation in first-order logic 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, ProVerif uses 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 lengths. If there is an attack against a protocol, then there is a bound such that this attack appears with lists shorter than the bound, but this bound depends on the protocol and is not easy to compute. So our goal is to extend ProVerif to the verification of protocols with lists of any length without bounding the length (and still not bounding the number of sessions). To obtain this result, 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 adapt the resolution algorithm to deal with these new clauses, and prove the soundness of the new algorithm. The obtained algorithm can prove secrecy and authentication properties. (More precisely, the authentication property that we consider is non-injective agreement [17]: if some participant B terminates the protocol with some parameters, then A started the protocol with the same parameters.) This algorithm performs approximations, so it may fail to prove security properties that actually hold, and it does not always terminate. This is unavoidable because the problem of verifying protocols with an unbounded number of sessions is undecidable. However, the algorithm works well in practice: we have implemented it and successfully tested it on several protocol examples (see Section 6). Applications of our results include in particular the treatment of XML protocols such as web services, XML documents being modeled using possibly nested lists. In this paper, we focus mainly on these protocols. We illustrate our work on the SOAP extension [9] to XML signatures [4]. XML signatures allow one to sign several parts of an XML

CCS'13, November 4–8, 2013, Berlin, Germany.

http://dx.doi.org/10.1145/2508859.2516679.

document, recorded in a SignedInfo element, which can be modeled as a list of references to the signed parts of the document. A known attack [18] may occur when the document contains several Body elements, and the signed Body is not the one that is used for subsequent treatments. Our tool detects this attack, and proves that a corrected version is secure. Another possible application is the verification of some group protocols that manipulate lists that contain one element for each group member, with an unbounded number of group members.

1.1 Related Work

The first approach considered for proving protocols with recursive data structures was interactive theorem proving: a recursive authentication protocol was studied for an unbounded number of participants, using Isabelle/HOL [20], and using rank functions and PVS [10]. However, this approach requires considerable human effort.

Truderung [22] showed a decidability result (in NEXPTIME) for secrecy in recursive protocols, which include transformations of lists, for a bounded number of sessions. This result was extended to a class of recursive protocols with XOR [16] in 3-NEXPTIME. Chridi et al [11, 12] 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.

Several approaches were considered for verifying XML protocols [6, 21, 15, 3], by translating them to the input format of a standard protocol verifier: the tool TulaFale [6] uses ProVerif as back-end; Kleiner and Roscoe [21, 15] translate WS-Security protocols to FDR; Backes et al [3] use AVISPA. All these approaches have little or no support for lists of unbounded length. For instance, TulaFale has support for list membership with unbounded lists, but does not go further.

Recently, Paiola and Blanchet [19] showed that, for a certain class of Horn clauses, if secrecy is proved by ProVerif for lists of length one, then secrecy also holds for lists of unbounded length. Their target applications are group protocols and XML protocols. However, this work is limited to secrecy, and the class of protocols that they handle is limited to protocols that treat all elements of lists uniformly. When their reduction result does not apply, a different approach is needed. We propose such an approach in this paper. We provide a practical algorithm that can prove both secrecy and authentication properties of protocols that manipulate different list elements in different ways.

1.2 Outline

The next section recalls the technique used by ProVerif. In Section 3, we introduce our running example and motivate the introduction of a new type of clauses, the generalized Horn clauses, that we will use to model group protocols. In Section 4, we formally define generalized Horn clauses, and their semantics by giving their translation into Horn clauses. In Section 5, we adapt the resolution algorithm to generalized Horn clauses and prove its soundness. Section 6 summarizes our experimental results. Because of space constraints, the proofs of all our results and additional details on our algorithm can be found in the long version of the paper, available at [8], together with our implementation.

2. A REMINDER ON PROVERIF

ProVerif translates the initial protocol into a set of Horn clauses. The syntax of these clauses is defined in Figure 1.

$p ::=$	patterns
$\quad x, y, z, v, w$	variable
$\quad a[p_1, \ldots, p_n]$	name
$\quad f(p_1, \ldots, p_n)$	constructor application
$F ::= pred(p)$	facts
$R ::= F_1 \wedge \cdots \wedge F_n \Rightarrow F$	Horn clause

Figure 1: Syntax of Horn clauses

The patterns represent messages that are 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 represented by the pattern $a[p_1, \ldots, p_n]$. These functions take as arguments the messages previously received by the principal that creates the name as well as session identifiers, which are variables that take a different value at each run of the protocol, to distinguish names created in different runs. As shown in, e.g., [7], this representation of names is a sound approximation. When a name has no arguments, we write a instead of $a[\,]$. We use a for a generic name and identifiers in sans serif font (e.g., sk) for fixed names.

The fact $\mathrm{att}(p)$ means that the attacker may have the pattern (message) p. ProVerif models authentication as correspondence assertions, such as "if event $e(x)$ has been executed, then event $e'(x)$ has been executed". It uses the fact $\mathrm{m\text{-}event}(p)$ to represent that the event p must have been executed, and the fact $\mathrm{event}(p)$ to represent that the event p may have been executed.

A clause $F_1 \wedge \cdots \wedge F_n \Rightarrow F$ 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. 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 $f(p_1, \ldots, p_n)$. Destructors manipulate patterns. A destructor g is defined by a set $def(g)$ of rewrite rules of the form $g(p_1, \ldots, p_n) \rightarrow p$ where $p_1, \ldots, p_n, p$ are patterns with only variables and constructors and the variables of p appear in $p_1, \ldots, p_n$. Using constructors and destructors, one can represent data structures and cryptographic operations. For instance, $senc(m, k)$ is the constructor that represents the symmetric key encryption of the message m under the key k. The corresponding destructor $sdec(m', k)$ returns the decryption of m' if m' is a message encrypted under k. The rewrite rule that defines $sdec$ is

$$sdec(senc(m, k), k) \rightarrow m.$$

A protocol is represented by three sets of Horn clauses:

- initial knowledge of the attacker: we have a fact $\mathrm{att}(p)$ for each p initially known by the attacker.
- abilities of the attacker: $\mathrm{att}(\mathsf{a}[x])$

for each constructor f of arity n:
$\text{att}(x_1) \wedge \cdots \wedge \text{att}(x_n) \Rightarrow \text{att}(f(x_1, \ldots x_n))$
for each destructor g,
for each rule $g(p_1, \ldots, p_n) \to p$ in $def(g)$:
$\text{att}(p_1) \wedge \cdots \wedge \text{att}(p_n) \Rightarrow \text{att}(p)$
The first clause represents the ability of the attacker to create fresh names: all fresh names that the attacker may create are represented by the names $\mathsf{a}[x]$ for any x. The other clauses mean that if the attacker has some messages, then he can apply constructors and destructors to them.

- the protocol itself: for each message p of the protocol sent by agent A, we have the clause $\text{att}(p_1) \wedge \cdots \wedge \text{att}(p_n) \wedge \text{m-event}(p'_1) \wedge \ldots \wedge \text{m-event}(p'_{n'}) \Rightarrow \text{att}(p)$, where A receives messages $p_1, \ldots, p_n$ and executes events $p'_1, \ldots, p'_{n'}$ before sending message p. Indeed, if the attacker has $p_1, \ldots, p_n$, then he can send them to A and intercept A's reply p; the events $p'_1, \ldots, p'_{n'}$ are executed by A during this operation.

  Similarly, for each event p executed by A, we have the clause $\text{att}(p_1) \wedge \cdots \wedge \text{att}(p_n) \wedge \text{m-event}(p'_1) \wedge \ldots \wedge \text{m-event}(p'_{n'}) \Rightarrow \text{event}(p)$, where A receives messages $p_1, \ldots, p_n$ and executes events $p'_1, \ldots, p'_{n'}$ before executing event p.

Let $\mathcal{R}_1$ be the set of these clauses. This representation of protocols by Horn clauses is approximate, mainly because Horn clauses that represent the protocol can be applied any number of times instead of only once per session. However, it is sound: if the attacker knows p after the events $p_1, \ldots, p_n$ have been executed, then $\text{att}(p)$ is derivable from $\mathcal{R}_1$ and the facts $\text{m-event}(p_1), \ldots, \text{m-event}(p_n)$. In particular, if $\text{att}(p)$ is not derivable from $\mathcal{R}_1$ and any facts $\text{m-event}(p')$, then the protocol preserves the secrecy of p. If the event p is executed after the events $p_1, \ldots, p_n$, then $\text{event}(p)$ is derivable from $\mathcal{R}_1$ and the facts $\text{m-event}(p_1), \ldots, \text{m-event}(p_n)$. (This is proved by Theorem 1 in [7] when the clauses are generated from a pi calculus model of the protocol.)

ProVerif determines whether a fact is derivable from the clauses using resolution with free selection [2]: it combines pairs of clauses by resolution; the literals upon which we resolve are chosen by a selection function. Next, we detail this algorithm.

We say that R_1 subsumes R_2 when R_2 can be obtained by adding hypotheses to an instance of R_1. In this case, all facts derivable using R_2 can also be derived by R_1, so R_2 can be eliminated. Formally, subsumption is defined by:

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).*

When the conclusion of a clause R unifies with a hypothesis of a clause R', resolution creates a new clause that corresponds to applying R and R' one after the other.

Definition 2 (Resolution). *Let R and R' be two clauses, $R = H \Rightarrow C$ and $R' = H' \Rightarrow C'$. Assume that there exists $F_0 \in H'$ such that C and F_0 are unifiable and σ is their most general unifier. In this case, we define $R \circ_{F_0} R' = \sigma(H \cup (H' \setminus \{F_0\})) \Rightarrow \sigma C'$. The clause $R \circ_{F_0} R'$ is the result of resolving R' with R upon F_0.*

$\text{SATUR}(\mathcal{R}_0) =$

1. $\mathcal{R} \leftarrow \text{ELIM}(\mathcal{R}_0)$.
2. Repeat until a fixpoint is reached
   for each $R \in \mathcal{R}$ such that $sel(R) = \emptyset$,
   for each $R' \in \mathcal{R}$, for each $F_0 \in sel(R')$ such that $R \circ_{F_0} R'$ is defined,
   $\mathcal{R} \leftarrow \text{ELIM}(\{R \circ_{F_0} R'\} \cup \mathcal{R})$.
3. Return $\{R \in \mathcal{R} \mid sel(R) = \emptyset\}$.

Figure 2: ProVerif's Algorithm

The facts upon which we resolve are selected through a selection function:

Definition 3 (Selection function). *A selection function is a function from clauses to sets of facts, such that $sel(H \Rightarrow C) \subseteq H$. If $F \in sel(R)$, we say that F is selected in R. If $sel(R) = \emptyset$, we say that no hypothesis is selected in R, or that the conclusion of R is selected.*

The algorithm is correct with any selection function that never selects facts of the form $\text{m-event}(p)$, but the choice of the selection function can change the behavior of the algorithm. Facts $\text{att}(x)$ where x is a variable can be unified with all facts $\text{att}(M)$, so we should avoid selecting $\text{att}(x)$ to reduce the number of possible resolution steps. Hence a good selection function satisfies the following formula:

$$sel(H \Rightarrow C) = \begin{cases} \{F_0\} \text{ where } F_0 \in H, \\ \qquad \text{for all variables } x, F_0 \neq \text{att}(x), \text{ and} \\ \qquad \text{for all patterns } p, F_0 \neq \text{m-event}(p) \\ \emptyset \qquad \text{if there exists no such } F_0 \end{cases}$$

The resolution algorithm is shown in Figure 2. It transforms the initial set of clauses into a new one that derives the same facts. The algorithm $\text{SATUR}(\mathcal{R}_0)$ has 3 steps:

- The first step inserts in $\mathcal{R}$ the clauses in $\mathcal{R}_0$ after elimination of subsumed clauses by ELIM: when $R' \sqsupseteq R$ and both R and R' are in $\mathcal{R}$, R is removed by $\text{ELIM}(\mathcal{R})$.
- The second step is a fixpoint iteration that adds the clauses created by resolution: if the conclusion of a clause R such that $sel(R) = \emptyset$ unifies with $F_0 \in sel(R')$, then the resolution $R \circ_{F_0} R'$ is added to $\mathcal{R}$. Then, subsumed clauses are eliminated by ELIM.
- Finally, the algorithm returns the clauses in $\mathcal{R}$ with no selected hypothesis.

Let $\mathcal{F}_{\text{me}}$ be any set of facts of the form $\text{m-event}(p)$.

Theorem 1 (Lemma 2 in [7]). *Let F be a closed fact and $\mathcal{R}_0$ a set of clauses. F is derivable from $\mathcal{R}_0 \cup \mathcal{F}_{\text{me}}$ if and only if it is derivable from $\text{SATUR}(\mathcal{R}_0) \cup \mathcal{F}_{\text{me}}$.*

To prove that a closed fact $\text{att}(p)$ is not derivable from $\mathcal{R}_0 \cup \mathcal{F}_{\text{me}}$, we use the following result, where att' is a new predicate:

Corollary 1. *If $\text{SATUR}(\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\})$ contains no clause of the form $H \Rightarrow \text{att}'(p')$, then $\text{att}(p)$ is not derivable from $\mathcal{R}_1 \cup \mathcal{F}_{\text{me}}$ for any $\mathcal{F}_{\text{me}}$. So, by soundness of the clauses, the protocol preserves the secrecy of p.*

Indeed, if $\text{SATUR}(\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\})$ contains no clause of the form $H \Rightarrow \text{att}'(p')$, then $\text{att}'(p)$ is not derivable from $\text{SATUR}(\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\}) \cup \mathcal{F}_{\text{me}}$, so by Theorem 1, $\text{att}'(p)$ is not derivable from $\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\} \cup \mathcal{F}_{\text{me}}$, so $\text{att}(p)$ is not derivable from $\mathcal{R}_1 \cup \mathcal{F}_{\text{me}}$. Similarly, we also have:

COROLLARY 2 (COROLLARY 2 IN [7]). *Suppose that all clauses of* $\text{SATUR}(\mathcal{R}_1)$ *that conclude* $\text{event}(e(p))$ *for some* p *are of the form* $\text{m-event}(e'(p')) \wedge H \Rightarrow \text{event}(e(p'))$ *for some* H *and* p'. *Then, for all* $\mathcal{F}_{\text{me}}$, *for all* p, *if* $\text{event}(e(p))$ *is derivable from* $\mathcal{R}_1 \cup \mathcal{F}_{\text{me}}$, *then* $\text{m-event}(e'(p)) \in \mathcal{F}_{\text{me}}$. *So by soundness of the clauses, the protocol satisfies the correspondence "if* $e(x)$ *has been executed, then* $e'(x)$ *has been executed".*

3. MOTIVATION

This section motivates this work, explaining the introduction of a new type of clauses through a running example.

3.1 Running Example

As a running example, we use a version of the SOAP extension to XML signatures [9]. SOAP envelopes are XML documents with a mandatory Body containing a request, response or a fault message together with an optional Header element containing application-specific information about the message (for example security information). In particular, the SOAP header can carry digital signature information within a SOAP envelope, as follows:

```
<Envelope>
  <Header>
    <Signature>
      <SignedInfo>
        <Reference URI="#theBody">
          <DigestValue> hash of the body </DigestValue>
        </Reference>
        <Reference URI="#x1">
          <DigestValue> hash of the content of x1
          </DigestValue>
        </Reference>
        ...
      </SignedInfo>
      <SignatureValue>
        signature of SignedInfo with key sk
      </SignatureValue>
    </Signature>
  </Header>
  <Body Id="#theBody"> request </Body>
</Envelope>
```

The Signature header contains two components. The first component is a SignedInfo element, which is basically a list of references to the elements of the message that are signed, designated by their identifier and accompanied by a DigestValue, a hash of their content. The hash may be computed with the hash function SHA-1. The second component of the Signature header is the signature of the SignedInfo element using a secret key sk.

We consider a simple protocol in which a client sends such a message to a server. The server processes the document and checks the signature before authorizing the request given in the Body: if the SignedInfo contains a Reference to an element with tag Body, then he will authorize the request. This protocol should guarantee that the server authorizes only requests signed by legitimate clients.

This protocol is subject to a known wrapping attack [18]: an attacker can intercept an envelope and create a new envelope wrapping the Body in the Header and adding a new body with a different id and a different request. The server will verify the signature and authorize the fake request made by the attacker. This attack is possible because the server does not check that the signed Body is the one he authorizes.

3.2 Need for Generalizing Horn Clauses

In order to model the previous example, we suppose that the XML parser parses the SOAP envelope as a pair, containing as first component a list of triplets (tag, id, corresponding content) and as second component the content of the mandatory body. The list in the first component is helpful so that one can retrieve the content of an element from its id by looking up the list. The content of the Signature header is modeled as a pair (*SignedInfo*, *SignatureValue*); *SignedInfo* is a list of pairs containing an id and the hash of the content corresponding to the id; *SignatureValue* is the signature of *SignedInfo* with a secret key sk. We use such a simple model for illustrative purposes.

A given XML document can then be represented using lists of fixed length (and other standard functions). We denote by $\langle p_1, \ldots, p_h \rangle$ a list of fixed length h; such lists are modeled as a family of constructors, one for each length.

However, the receiver of the SOAP envelope accepts messages containing any number of headers, and the SignedInfo element may also contain references to any number of elements in the message. Therefore, we need lists of variable length in order to model the expected message. So we introduce the construct $list(i \leq M, p_i)$ which stands for $\langle p_1, \ldots, p_M \rangle$, for an unknown M (inspired by [12, 19]).

Thanks to this new construct, we can model the SOAP envelope from the point of view of the server: $(list(i \leq M, (tag_i, id_i, cont_i)), r)$, where tag_i, id_i, and $cont_i$ are variables representing tags, identifiers, and contents respectively and r is the variable for the request. The server has to check the signature, that is, he has to verify that the list contains a tag tag_j equal to Signature and that $cont_j$ contains a correct signature. These checks can be represented by the following equations: $tag_j \doteq \text{Signature}$, $cont_j \doteq (sinfo, sign(sinfo, \text{sk}))$, $sinfo \doteq list(k \leq N, (id_{\phi(k)}, sha1(cont_{\phi(k)})))$, where the function ϕ is a mapping from the index k of each element in the *sinfo* list to its index $\phi(k)$ in the list that represents the whole message. Furthermore, one of the signed elements must have tag Body, that is, $\phi(m) \doteq d$, $tag_d \doteq \text{Body}$. We cannot directly replace the variables tag_i, id_i, and $cont_i$ with their values given by these equations, because in the list $list(i \leq M, (tag_i, id_i, cont_i))$, all elements tag_i, id_i and $cont_i$ need to have the same form, while the equations give different forms to different elements. So we keep the equations in the clause for future use. The equations allow us to handle protocols that treat elements of lists non-uniformly.

We can represent that the adversary has a SOAP envelope by $\text{att}((list(i \leq M, (tag_i, id_i, cont_i)), r))$. The adversary can build such a SOAP envelope from its components, so we need to express that the adversary has $(tag_i, id_i, cont_i)$ for all $i \leq M$. We use a conjunction $\bigwedge_{i \in [1,M]} \text{att}((tag_i, id_i, cont_i))$ for this purpose. More generally, $\bigwedge_{(i_1,\ldots i_h) \in I} F$ represents the conjunction of facts F for all indices $(i_1, \ldots, i_h) \in I$.

After modeling the clauses, we need to perform resolution on these generalized clauses. Let $[1, M] = \{1, \ldots, M\}$ and suppose that we resolve $R_2 = \bigwedge_{i \in [1,M]} \text{att}(sha1(cont_{\phi(i)})) \Rightarrow$

event($e(r)$) with the clause $R_1 = \text{att}(x) \Rightarrow \text{att}(sha1(x))$, that is, we use R_1 to derive one hypothesis of R_2, say the one of index $i = k_1 \in [1, M]$. So we unify $\text{att}(sha1(x))$ with $\text{att}(sha1(cont_{\phi(k_1)}))$, yielding the equality $x \doteq cont_{\phi(k_1)}$. The obtained clause would then be:

$$\text{att}(x_1) \wedge \textstyle\bigwedge_{i \in [1,M] \setminus \{k_1\}} \text{att}(sha1(cont_{\phi(i)})) \wedge \{x_1 \doteq cont_{\phi(k_1)}\} \Rightarrow \text{event}(e(r))$$

The variable x is renamed into x_1 to distinguish it from the variable x in R_1. The obtained clause can then again be resolved with R_1 for some $i = k_2 \in [1, M] \setminus \{k_1\}$, yielding

$$\text{att}(x_1) \wedge \text{att}(x_2) \wedge \textstyle\bigwedge_{i \in [1,M] \setminus \{k_1, k_2\}} \text{att}(sha1(cont_{\phi(i)})) \wedge \{x_1 \doteq cont_{\phi(k_1)}, x_2 \doteq cont_{\phi(k_2)}\} \Rightarrow \text{event}(e(r))$$

which can again be resolved with R_1, and so on. Since M is not bounded, such resolution steps yield an infinite loop. To avoid this loop, we define a resolution step that simultaneously resolves R with several instances of R' for i in any non-empty subset $I \subseteq [1, M]$. We name such a step *hyperresolution* by analogy with the hyperresolution rule that allows one to resolve one clause with several clauses [14]. To perform hyperresolution, we first transform R_1 into a clause R'_1 corresponding to the combination of several instances of R_1, one for each $i \in I$. This step is named *immersion* and yields $R'_1 = \bigwedge_{i \in I} \text{att}(x_i) \Rightarrow \text{att}(sha1(x_i))$. We can then perform hyperresolution with R_2 and obtain:

$$\textstyle\bigwedge_{i \in I} \text{att}(x_i) \wedge \bigwedge_{i \in [1,M] \setminus I} \text{att}(sha1(cont_{\phi(i)})) \wedge \{\bigwedge_{i \in I} x_i \doteq cont_{\phi(i)}\} \Rightarrow \text{event}(e(r)).$$

When $I = [1, M]$, the second hypothesis is removed and the substitution of $cont_{\phi(i)}$ for x_i is performed for all i, so we obtain

$$\textstyle\bigwedge_{i \in [1,M]} \text{att}(cont_{\phi(i)}) \Rightarrow \text{event}(e(r)).$$

When $I \neq [1, M]$, to have a more symmetric notation and to simplify the language of sets used in conjunctions, we introduce a symbol I' for $[1, M] \setminus I$, and keep the constraint that $I \uplus I' = [1, M]$: I and I' are disjoint and their union is $[1, M]$. The clause is then denoted by

$$I \uplus I' = [1, M],\ \textstyle\bigwedge_{i \in I} \text{att}(cont_{\phi(i)}) \wedge \bigwedge_{i \in I'} \text{att}(sha1(cont_{\phi(i)})) \Rightarrow \text{event}(e(r)).$$

where I and I' represent non-empty subsets of $[1, M]$.

4. GENERALIZED HORN CLAUSES

This section formally defines the syntax and semantics of *generalized Horn Clauses*, which were motivated informally in the previous section.

4.1 Syntax

The syntax of these new clauses is defined in Figure 3. The patterns p^G that represent messages are enriched with several new constructs. The variables may have indices $x_{\iota_1,\ldots,\iota_h}$. The pattern for function application $f(p_1^G, \ldots, p_l^G)$ includes not only constructor application but also names $a[p_1^G, \ldots, p_l^G]$ where a is a name without index. We consider tuples $(p_1, \ldots, p_l^G)$ and lists of fixed length $\langle p_1^G, \ldots, p_l^G \rangle$ as particular constructors. We suppose that the implementation of the protocol uses distinct encodings for tuples and for lists, so that they cannot be confused with each other. The

$\iota ::=$	index term
i	index variable
$\phi(\iota_1, \ldots, \iota_h)$	function application
$p^G, p'^G ::=$	patterns
$x_{\iota_1,\ldots,\iota_h}$	variable ($h \geq 0$)
$f(p_1^G, \ldots, p_l^G)$	function application
$a_\iota^M[p_1^G, \ldots, p_l^G]$	indexed names
$list(i \leq M, p^G)$	list constructor
$J ::=$	set computation
I	set symbol
$\{()\}$	singleton
$J \times [1, M]$	product
$\mathcal{C} ::= \bigwedge_{(i_1,\ldots i_h) \in J}$	conjunction
$F^G ::= \mathcal{C}\ pred(p^G)$	fact
$E ::= \mathcal{C}\ p^G \doteq p'^G$	equation over patterns
$E' ::= \mathcal{C}\ \iota \doteq \iota'$	equation over indices
$\mathcal{E} ::= \{E_1, \ldots, E_n, E'_1, \ldots, E'_{n'}\}$	set of equations
$\mathcal{I} ::= I_1 \uplus \cdots \uplus I_h = J$	constraint over sets
$Cts ::= \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$	set of constraints
$R^G ::= Cts,\ F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow pred(p^G)$	generalized Horn clause

Figure 3: Syntax of generalized Horn clauses

construct $a_\iota^M[p_1^G, \ldots, p_l^G]$ represents a fresh name a indexed by ι in $[1, M]$. For instance, in the context of group protocols, it may represent a name created by the group member number ι, inside a group of size M. We use the construct $list(i \leq M, p^G)$ to represent lists of length M.

We extend facts to model the possibility of having a conjunction of facts depending on indices, so that the facts become $\bigwedge_{(i_1,\ldots,i_h) \in J} pred(p^G)$. The set J can be a product of a set symbol I or a singleton $\{()\}$ and of different sets $[1, M]$, depending on different bounds M. The symbol $[1, M]$ represents the set $\{1, \ldots, M\}$. The symbol $[1, M]$ allows us to keep the information that an index ranges over the full set of indices of bound M, while other set symbols I represent an unknown, non-empty set of indices. We write $[1, M]$ instead of $\{()\} \times [1, M]$. For example, intuitively, $\bigwedge_{i \in [1,N]} pred(p^G)$ represents $pred(p^G\{i \mapsto 1\}) \wedge \cdots \wedge pred(p^G\{i \mapsto N\})$, where $p^G\{i \mapsto i'\}$ denotes p^G in which i has been replaced with i'. The conjunction $\mathcal{C} = \bigwedge_{(i_1,\ldots,i_h) \in J}$ with $J = \{()\}$ and $h = 0$ is omitted: the fact $\mathcal{C}\ pred(p^G)$ is then simply $pred(p^G)$.

In the generalized Horn clause $Cts,\ F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow pred(p^G)$, there are two new members: a *set of constraints* Cts and a *set of equations* $\mathcal{E}$. The first one is a set of constraints on sets used in conjunctions: the constraint $I_1 \uplus \ldots \uplus I_h = J$ means that $I_1, \ldots, I_h$ are pairwise disjoint and their union is J. The second one is a set of equations that represent the substitutions that cannot be done because the equations hold for some but not all values of the indices. These equations can be equations over patterns E and equations over indices E'. The clause $Cts,\ F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow pred(p^G)$ means that, if the constraints in Cts are satisfied,

and the facts $F_1^G, \ldots, F_n^G$ and the equations in $\mathcal{E}$ hold, then the fact $pred(p^G)$ also holds. The conclusion of a clause does not contain a conjunction $\mathcal{C}$: we can simply leave the indices of $pred(p^G)$ free to mean that $pred(p^G)$ can be concluded for any value of these indices.

The clauses are required to satisfy the following invariants:

1. In a conjunction $\bigwedge_{(i_1,\ldots,i_h)\in J}$, the indices $i_1, \ldots, i_h$ are pairwise distinct.
2. Each set symbol I occurs at most once in each constraint in Cts.
3. Each set symbol I occurs in at most one left-hand side of a constraint in Cts.
4. If a set symbol I occurs in a conjunction or in the right-hand side of a constraint in Cts, then it occurs in exactly one left-hand side of a constraint in Cts.

We use H^G for hypothesis and C^G for conclusions. When Cts or $\mathcal{E}$ are empty, we omit them in the clause.

4.2 Representation of the Protocol

The representation of the abilities of the attacker includes the clauses given in Section 2. For our running example, $\mathrm{att}(pk(\mathsf{sk}))$, $\mathrm{att}(\mathsf{Signature})$, $\mathrm{att}(\mathsf{Body})$ represent that the attacker initially knows the public key $pk(\mathsf{sk})$ and the constants Signature and Body, and the clauses

$$\begin{array}{ll}
\mathrm{att}(\mathsf{a}[x]) & \\
\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}(sign(x,y)) & \mathrm{att}((x,y)) \Rightarrow \mathrm{att}(x) \\
\mathrm{att}(x) \Rightarrow \mathrm{att}(pk(x)) & \mathrm{att}((x,y)) \Rightarrow \mathrm{att}(y) \\
\mathrm{att}(x) \Rightarrow \mathrm{att}(sha1(x)) & \mathrm{att}(x) \Rightarrow \mathrm{att}(\langle x \rangle) \\
\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}((x,y)) & \mathrm{att}(\langle x \rangle) \Rightarrow \mathrm{att}(x)
\end{array}$$

represent that the attacker can create fresh names, sign messages, create its own public keys, apply hash functions, compose and decompose pairs and lists of length one. We have similar clauses for triples and lists of length two. (These arities are the only ones used in our example.)

In addition, we have clauses for *list*, which generalize clauses for tuples and lists of fixed length:

$$\textstyle\bigwedge_{i\in[1,M]} \mathrm{att}(x_i) \Rightarrow \mathrm{att}(list(j \le M, x_j)) \quad (1)$$

$$\mathrm{att}(list(j \le M, x_j)) \Rightarrow \mathrm{att}(x_i) \quad (2)$$

Let us now give clauses that represent the protocol itself. To model the authentication, we use two events b and e. The event $b(r)$ means that the client sends the request r; the event $e(r)$ means that the server authorizes the request r. Our goal is to prove that, if $e(r)$ is executed, then $b(r)$ has been executed. We suppose that the only element signed by the client is the Body. Hence the document can be represented as follows: $(\langle(\mathsf{Signature}, \mathsf{ids}, (\langle(\mathsf{idb}, sha1(\mathsf{Req}))\rangle, sign((\langle(\mathsf{idb}, sha1(\mathsf{Req}))\rangle, \mathsf{sk})))), (\mathsf{Body}, \mathsf{idb}, \mathsf{Req})\rangle, \mathsf{Req})$, where ids is the identifier of the Signature and idb the one of the Body. The client executes the event $b(\mathsf{Req})$, then sends the SOAP envelope on the network and the attacker intercepts it, so we have the clause:

$$\text{m-event}(b(\mathsf{Req})) \Rightarrow \mathrm{att}((\langle(\mathsf{Signature}, \mathsf{ids}, (\langle(\mathsf{idb}, sha1(\mathsf{Req}))\rangle, sign((\langle(\mathsf{idb}, sha1(\mathsf{Req}))\rangle, \mathsf{sk})))), (\mathsf{Body}, \mathsf{idb}, \mathsf{Req})\rangle, \mathsf{Req})). \quad (3)$$

The server expects a document $(list(i \le M, (tag_i, id_i, cont_i)), r)$; he checks the signature, and if the check succeeds, he executes the event $e(r)$:

$$\begin{array}{l}
\mathrm{att}((list(i \le M, (tag_i, id_i, cont_i)), r)) \wedge \\
\{tag_j \doteq \mathsf{Signature}, cont_j \doteq (sinfo, sign(sinfo, \mathsf{sk})), \\
\quad sinfo \doteq list(k \le N, (id_{\phi(k)}, sha1(cont_{\phi(k)}))), \\
\quad \phi(m) \doteq d, tag_d \doteq \mathsf{Body}\} \Rightarrow \\
\qquad \mathrm{event}(e(r))
\end{array} \quad (4)$$

This clause uses the equations explained in Section 3.2.

As already mentioned in Section 3.1, this protocol has a wrapping attack. The corrected version of this protocol additionally requires that the signed body is the one that the server authorizes. This additional check can be modeled by adding the equation $cont_d \doteq r$ in the clause (4).

4.3 Type System

In this section, we define a simple type system for generalized Horn clauses, to guarantee that the indices of all variables vary in the appropriate interval.

DEFINITION 4. *An index i is bound if:*

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

We identity facts, equations, and clauses up to renaming of bound indices. For simplicity, we suppose that the bound indices of clauses have been renamed so that they have pairwise distinct names, and names distinct from the names of free indices. The set of free indices of a fact F^G, of a clause R^G, or of an hypothesis H^G is denoted by $fi(F^G)$, $fi(R^G)$, $fi(H^G)$ respectively.

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;
- $\phi : [1, M_1] \times \cdots \times [1, M_h] \to [1, M]$ means that the function ϕ expects as input h indices of types $[1, M_j]$, for $j = 1, \ldots, h$ and computes an index of type $[1, M]$;
- $x_\_ : [1, M_1] \times \cdots \times [1, M_h]$ means that the variable x expects indices of types $[1, M_1]$, $\ldots$, $[1, M_h]$;
- $I : [1, M_1] \times \cdots \times [1, M_h]$ means that the elements of set I are tuples of h indices, each of type $[1, M_j]$ for $j = 1, \ldots, h$.

The type system defines the judgments:

- $\Gamma \vdash \iota : [1, M]$, which means that ι has type $[1, M]$ in the type environment Γ;
- $\Gamma \vdash J : [1, M_1] \times \cdots \times [1, M_h]$, which means that the elements of set J are tuples of h indices, each of type $[1, M_j]$ for $j = 1, \ldots, h$ in the type environment Γ;

- $\Gamma \vdash p^G$, $\Gamma \vdash F^G$, $\Gamma \vdash E$, $\Gamma \vdash E'$, $\Gamma \vdash \mathcal{I}$, $\Gamma \vdash R^G$, which mean that p^G, F^G, E, E', $\mathcal{I}$, R^G, respectively, are well-typed in the type environment Γ.

The typing rules are straightforward and are given in the long version of the paper [8]. They basically guarantee that all indices have their expected type.

We suppose that all clauses are well-typed, and we consider that each clause R^G comes with its type environment Γ such that $\Gamma \vdash R^G$. It is easy to verify that the clauses of our running example are all well-typed.

4.4 Translation from Generalized Horn Clauses to Horn Clauses

A generalized Horn clause represents several Horn clauses: for each value of the bounds M, set symbols I, functions ϕ, and 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.

DEFINITION 5. *Given a well-typed generalized Horn clause $\Gamma \vdash R^G$, an environment T for $\Gamma \vdash R^G$ is a function that associates:*

- *to each bound M that appears in R^G or Γ an integer M^T;*
- *to each index i such that $i : [1, M] \in \Gamma$, an index $i^T \in \{1, \ldots, M^T\}$;*
- *to each index function ϕ such that $\phi : [1, M_1] \times \cdots \times [1, M_h] \to [1, M] \in \Gamma$, a function $\phi^T : \{1, \ldots, M_1^T\} \times \cdots \times \{1, \ldots, M_h^T\} \to \{1, \ldots, M^T\}$.*
- *to each set symbol I such that $\mathcal{I} : [1, M_1] \times \cdots \times [1, M_h] \in \Gamma$, a set I^T such that $\emptyset \subset I^T \subseteq \{1, \ldots, M_1^T\} \times \cdots \times \{1, \ldots, M_h^T\}$;*

Given an environment T and values $v_1, \ldots, v_h$, we write $T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h]$ for the environment that associates to indices $i_1, \ldots, i_h$ the values $v_1, \ldots, v_h$ respectively and that maps all other values like T.

Given an environment T for $\Gamma \vdash R^G$, the generalized Horn clause R^G is translated into the standard Horn clause R^{GT} defined as follows. We denote respectively $p^{GT}, E^T, \ldots$ the translation of $p^G, E, \ldots$ using the environment T.

The translation of an index term ι such that $\Gamma \vdash \iota : [1, M]$ is an integer $\iota^T \in \{1, \ldots, M^T\}$ defined as follows:

$$\iota^T = \begin{cases} i^T & \text{if } \iota = i \\ \phi^T(\iota_1^T, \ldots, \iota_h^T) & \text{if } \iota = \phi(\iota_1, \ldots, \iota_h) \end{cases}$$

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

$$\begin{aligned} (x_{\iota_1,\ldots,\iota_h})^T &= x_{\iota_1^T,\ldots,\iota_h^T} \\ f(p_1^G, \ldots, p_l^G)^T &= f(p_1^{GT}, \ldots, p_l^{GT}) \\ a_\iota^M[p_1^G, \ldots, p_l^G]^T &= a_{\iota^T}^{M^T}[p_1^{GT}, \ldots, p_l^{GT}] \\ \mathit{list}(i \le M, p^G)^T &= \langle p^{GT[i \mapsto 1]}, \ldots, p^{GT[i \mapsto M^T]} \rangle \end{aligned}$$

The translation of $\mathit{list}(i \le M, p^G)$ is a list of length M^T.

The translation of a set computation J is a set of tuples defined by:

$$J^T = \begin{cases} I^T & \text{if } J = I \\ \{\,()\,\} & \text{if } J = \{()\} \\ J'^T \times \{1, \ldots, M^T\} & \text{if } J = J' \times [1, M] \end{cases}$$

In this definition, the cross product $\times$ decomposes tuples before building the whole tuple, so that $J'^T \times \{1, \ldots, M^T\} = \{(v_1, \ldots, v_h, v) \mid (v_1, \ldots, v_h) \in J'^T, v \in \{1, \ldots, M^T\}\}$.

Given a conjunction $\mathcal{C} = \bigwedge_{(i_1,\ldots,i_h) \in J}$ and an environment T, we define the set of environments $T^{\mathcal{C}} = \{T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h] \mid (v_1, \ldots, v_h) \in J^T\}$: these environments map the indices $i_1, \ldots, i_h$ of the conjunction to all their possible values in J^T and map all other values like T.

The translation of a fact $F = \mathcal{C}\, \mathit{pred}(p^G)$ is

$$(\mathcal{C}\, \mathit{pred}(p^G))^T = \mathit{pred}(p_1) \wedge \ldots \wedge \mathit{pred}(p_k)$$

where $\{p_1, \ldots, p_k\} = \{p^{GT'} \mid T' \in T^{\mathcal{C}}\}$, and $(F_1 \wedge \cdots \wedge F_n)^T = F_1^T \wedge \cdots \wedge F_n^T$.

The translation of a set of equations $\mathcal{E}$ is the set $\mathcal{E}^T$ obtained by translating the equations E, E' as follows:

- $(\mathcal{C}\, \iota \doteq \iota')^T = \begin{cases} \mathit{true} & \text{if for all } T' \in T^{\mathcal{C}},\ \iota^{T'} = \iota'^{T'} \\ \mathit{false} & \text{otherwise.} \end{cases}$
- $(\mathcal{C}\, p^G \doteq p'^G)^T = \{p^{GT'} = p'^{GT'} \mid T' \in T^{\mathcal{C}}\}$.
- If $\forall E' \in \mathcal{E}$, $E'^T = \mathit{true}$, then $\mathcal{E}^T = \bigcup_{E \in \mathcal{E}} E^T$; otherwise, $\mathcal{E}^T$ is undefined.

The equations over indices can be evaluated to *true* or *false* knowing the environment T. The equations over generalized patterns are translated into equations over patterns. A set of equations $\mathcal{E}$ is translated into a set of equations over patterns if all equations over indices are true. Otherwise, the translation of $\mathcal{E}$ is undefined.

Given a set of constraints *Cts* and an environment T, we say that T satisfies *Cts* when, for all equations $I_1 \uplus \cdots \uplus I_m = J$ in *Cts*, we have that $I_1^T \uplus \cdots \uplus I_m^T = J^T$, that is, $I_1^T, \ldots, I_m^T$ are pairwise disjoint and their union is J^T.

Given a set of equations $\{p_1 = p'_1, \ldots, p_n = p'_n\}$ over standard patterns, we define as usual its most general unifier $\text{MGU}(\{p_1 = p'_1, \ldots, p_n = p'_n\})$ as the most general substitution σ such that $\sigma p_i = \sigma p'_i$ for all $i \in \{1, \ldots, n\}$, $\mathit{dom}(\sigma) \cup \mathit{fv}(\mathit{im}(\sigma)) \subseteq \mathit{fv}(p_1, p'_1, \ldots, p_n, p'_n)$, and $\mathit{dom}(\sigma) \cap \mathit{fv}(\mathit{im}(\sigma)) = \emptyset$, where $\mathit{fv}(p)$ designates the (free) variables of p, $\mathit{dom}(\sigma)$ is the domain of σ: $\mathit{dom}(\sigma) = \{x \mid \sigma x \neq x\}$, and $\mathit{im}(\sigma)$ is the image of σ: $\mathit{im}(\sigma) = \{\sigma x \mid \sigma x \neq x\}$. We denote by $\{x_1 \mapsto p_1, \ldots, x_n \mapsto p_n\}$ the substitution that maps x_i to p_i for all $i = 1, \ldots, n$.

Finally, we define the translation of the generalized Horn clause $R^G = \mathit{Cts}, H^G \wedge \mathcal{E} \Rightarrow \mathit{pred}(p^G)$ as follows. If T does not satisfy *Cts*, $\mathcal{E}^T$ is undefined, or the unification of $\mathcal{E}^T$ fails, then R^{GT} is undefined. Otherwise, $R^{GT} = \text{MGU}(\mathcal{E}^T) H^{GT} \Rightarrow \text{MGU}(\mathcal{E}^T) \mathit{pred}(p^{GT})$.

When $\mathcal{R}^G$ is a set of well-typed generalized Horn clauses (i.e., a set of pairs of a type environment Γ and a clause R^G such that $\Gamma \vdash R^G$), we define $\mathcal{R}^{GT} = \{R^{GT} \mid \Gamma \vdash R^G \in \mathcal{R}^G,\ T$ is an environment for $\Gamma \vdash R^G$ and R^{GT} is defined$\}$. 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 $\mathcal{R}^G$.

5. RESOLUTION FOR GENERALIZED HORN CLAUSES

In this section, we adapt ProVerif's resolution algorithm to generalized Horn clauses: we first define adapted algo-

rithms for subsumption, resolution, and unification, and then use these components to build the new resolution algorithm.

5.1 Subsumption

To adapt the subsumption test to generalized Horn clauses, we first define adapted notions of substitutions, then define subsumption and prove its soundness.

DEFINITION 6 (SUBSTITUTIONS). *We denote by ρ a substitution that maps:*

- *bounds to bounds $\rho(M) = N$;*
- *index variables to index terms: $\rho(i) = \iota$;*
- *set symbols to set symbols: $\rho(I) = I'$;*
- *function symbols ϕ to function symbols: $\rho(\phi) = \phi'$.*

We denote by σ^G a substitution that maps variables to patterns: $\sigma^G(x_{i_1,\ldots,i_h}) = p^G$ where $fi(p^G) \subseteq \{i_1, \ldots, i_h\}$.

As usual, substitutions are extended from variables to index terms, patterns, facts, equations, constraints, and clauses as homomorphisms: for instance, $\rho(\phi(\iota_1, \ldots, \iota_k)) = (\rho(\phi))(\rho(\iota_1), \ldots, \rho(\iota_k))$. However, since the grammar of generalized Horn clauses is richer than usual, we need to notice two points:

- Substitutions avoid the capture of bound indices: if $i \notin dom(\rho)$ and $i \notin fi(im(\rho))$, $\rho(list(i \leq M, p^G)) = list(i \leq \rho(M), \rho(p^G))$, which can be guaranteed by renaming i if necessary. We have similar formulas for indices bound by conjunctions as well as for substitutions σ^G.
- The substitutions σ^G are extended from variables $x_{i_1,\ldots,i_h}$ with indices $i_1, \ldots, i_h$ to variables $x_{\iota_1,\ldots,\iota_h}$ with terms $\iota_1, \ldots, \iota_h$ as indices: if $\sigma^G(x_{i_1,\ldots,i_h}) = p^G$, then $\sigma^G(x_{\iota_1,\ldots,\iota_h}) = \rho p^G$ where $\rho = \{i_1 \mapsto \iota_1, \ldots, i_h \mapsto \iota_h\}$.

We need a notion of subsumption between type environments, which is intuitively $\rho\Gamma_1 \subseteq \Gamma_2$. However, we cannot use exactly $\rho\Gamma_1 \subseteq \Gamma_2$ because ρ maps indices to index terms and not to indices, so we define $\Gamma_1 \leq_\rho \Gamma_2$ by:

- for each $i : [1, M] \in \Gamma_1$, we have $\Gamma_2 \vdash \rho i : [1, \rho M]$;
- for each $\phi : [1, M_1] \times \cdots \times [1, M_h] \to [1, M] \in \Gamma_1$, we have $\rho\phi : [1, \rho M_1] \times \cdots \times [1, \rho M_h] \to [1, \rho M] \in \Gamma_2$;
- for each $I : [1, M_1] \times \cdots \times [1, M_h] \in \Gamma_1$, we have $\rho I : [1, \rho M_1] \times \cdots \times [1, \rho M_h] \in \Gamma_2$;
- for each $x_- : [1, M_1] \times \cdots \times [1, M_h] \in \Gamma_1$, we have $x_- : [1, \rho M_1] \times \cdots \times [1, \rho M_h] \in \Gamma_2$.

DEFINITION 7 (SUBSUMPTION). *Given two well-typed generalized Horn clauses, $\Gamma_1 \vdash R_1^G$ and $\Gamma_2 \vdash R_2^G$, with $R_1^G = Cts_1, H_1^G \wedge \mathcal{E}_1 \Rightarrow C_1^G$ and $R_2^G = Cts_2, H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$, we say that $\Gamma_1 \vdash R_1^G$ subsumes $\Gamma_2 \vdash R_2^G$, and we write $\Gamma_1 \vdash R_1^G \sqsupseteq \Gamma_2 \vdash R_2^G$, if there exist substitutions ρ and σ^G such that $\sigma^G \rho C_1^G = C_2^G$, $\sigma^G \rho H_1^G \subseteq H_2^G$ (multiset inclusion), $\sigma^G \rho \mathcal{E}_1 \subseteq \mathcal{E}_2$ (modulo commutativity of $\doteq$), $\rho Cts_1 \subseteq Cts_2$ (modulo associativity and commutativity of $\uplus$), and $\Gamma_1 \leq_\rho \Gamma_2$.*

This definition is similar to subsumption for standard clauses but uses the richer substitutions introduced above.

Example 1. The clause

$$R_1^G = \{I \uplus I_1 = [1, M]\}, \bigwedge_{i \in I} \mathrm{att}(sign(x_i, \mathsf{sk}^L_{\phi(l)})) \wedge \{\bigwedge_{m \in I_1} y_m \doteq \mathsf{c}^M_m\} \Rightarrow \mathrm{att}(y_l)$$

typed by the type environment $\Gamma_1 = \{I : [1, M], I_1 : [1, M], x_- : [1, M], \phi : [1, M] \to [1, L], y_\cdot[1, M], l : [1, M]\}$, subsumes the clause

$$R_2^G = \{I' \uplus I'_1 = [1, N]\}, \bigwedge_{j \in I'} \mathrm{att}(sign((\mathsf{r}^N_{\psi(j)}, \mathsf{s}^N_{\psi(j)}), \mathsf{sk}^O_{\varphi(\psi(k))})) \wedge \{\bigwedge_{n \in I'_1} z_n \doteq \mathsf{c}^N_n\} \Rightarrow \mathrm{att}(z_{\psi(k)}).$$

typed by the environment $\Gamma_2 = \{I' : [1, N], I'_1 = [1, N], \psi : [1, N] \to [1, N], \varphi : [1, N] \to [1, O], k : [1, N], z_- : [1, N], k : [1, N]\}$ With $\rho = \{M \mapsto N, L \mapsto O, I \mapsto I', I_1 \mapsto I'_1, \phi \mapsto \varphi, l \mapsto \psi(k)\}$ and renaming the bound indices i into j and m into n, we obtain:

$$\rho R_1^G = \{I' \uplus I'_1 = [1, N]\}, \bigwedge_{j \in I'} \mathrm{att}(sign(x_j, \mathsf{sk}^O_{\varphi(\psi(k))})) \wedge \{\bigwedge_{n \in I'_1} y_n \doteq \mathsf{c}^N_n\} \Rightarrow \mathrm{att}(y_{\psi(k)})$$

and conclude that R_1^G subsumes R_2^G using the substitution

$$\sigma^G = \{x_i \mapsto (\mathsf{r}^N_{\psi(i)}, \mathsf{s}^N_{\psi(i)}), y_i \mapsto z_i\}.$$

The following theorem shows the soundness of subsumption for generalized Horn clauses.

THEOREM 2 (SUBSUMPTION). *Let $\Gamma_1 \vdash R_1^G$ and $\Gamma_2 \vdash R_2^G$ be two well-typed clauses. If $\Gamma_1 \vdash R_1^G \sqsupseteq \Gamma_2 \vdash R_2^G$ then, for all environments T_2 for $\Gamma_2 \vdash R_2^G$ such that $R_2^{GT_2}$ is defined, there exists an environment T_1 for $\Gamma_1 \vdash R_1^G$ such that $R_1^{GT_1}$ is defined and $R_1^{GT_1} \sqsupseteq R_2^{GT_2}$.*

As for standard clauses, our resolution algorithm for generalized Horn clauses will eliminate subsumed clauses. This theorem shows that, if R_2^G is eliminated in the algorithm for generalized Horn clauses because it is subsumed by R_1^G, then all corresponding clauses $R_2^{GT_2}$ would be eliminated in the algorithm for standard clauses: $R_2^{GT_2}$ is subsumed by some $R_1^{GT_1}$.

The subsumption test is approximate, because clauses that have the same meaning may sometimes be written in different forms. For instance, one can compose two functions ϕ and ψ, always writing $\phi(\psi(i, j))$ instead of a single function $\psi(i, j)$. These two formulas yield the same translation into standard Horn clauses, but are considered different by the subsumption test. This approximation does not contradict the soundness of subsumption, and the subsumption test is precise enough for the proof to succeed on our examples.

Because of the presence of indices, the algorithm for computing the substitutions ρ and σ^G required in Definition 7 is more complex than for standard clauses. We detail it in the long version of the paper [8].

5.2 Resolution and Hyperresolution

As introduced in Section 3.2, resolution becomes hyperresolution for generalized clauses. We first give the formal definition of hyperresolution, then explain it.

DEFINITION 8 (HYPERRESOLUTION). *Let* $R_1^G = Cts_1$, $H_1^G \wedge \mathcal{E}_1 \Rightarrow C_1^G$ *and* $R_2^G = Cts_2$, $F_0^G \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$ *be two clauses. We rename* R_1^G *and* R_2^G *so that they do not use common names for bounds* M*, variables* x*, indices* i*, functions on indices* ϕ*, and set symbols* I*.*

First case: $F_0^G = \bigwedge_{(i_1,\ldots,i_h)\in J} pred_2(p_2^G)$ *(*$h \neq 0$*). Let* $\Gamma_1 \vdash R_1^G$ *and* J *such that* $\Gamma_2 \vdash J : [1, M_1] \times \cdots \times [1, M_h]$*. The immersion of* R_1^G *into* $(i_1, \ldots, i_h) \in J$ *is the clause* $imm(R_1^G, (i_1, \ldots, i_h) \in J)$ *obtained by replacing:*

1. *all free indices* i *of* R_1^G *with* $\phi'(i_1, \ldots, i_h)$*, where a new function symbol* ϕ' *is associated to each free index* i *of* R_1^G*;*
2. *all index terms* $\phi(\iota_1, \ldots, \iota_k)$ *with* $\phi'(i_1, \ldots, i_h, \iota_1, \ldots, \iota_k)$*, where a new function symbol* ϕ' *is associated to each function symbol* ϕ *in* R_1^G*;*
3. *all variables* $x_{\iota_1,\ldots,\iota_k}$ *in* R_1^G *with* $x_{i_1,\ldots,i_h,\iota_1,\ldots,\iota_k}$*;*
4. *all conjunctions* $\bigwedge_{(j_1,\ldots,j_k)\in[1,M_1']\times\cdots\times[1,M_k']}$ *in* H_1^G *and* $\mathcal{E}_1$ *with* $\bigwedge_{(i_1,\ldots,i_h,j_1,\ldots,j_k)\in J\times[1,M_1']\times\cdots\times[1,M_k']}$*;*
5. *all conjunctions* $\bigwedge_{(j_1,\ldots,j_k)\in I\times[1,M_1']\times\cdots\times[1,M_l']}$ *in* H_1^G *and* $\mathcal{E}_1$ *with* $\bigwedge_{(i_1,\ldots,i_h,j_1,\ldots j_k)\in I'\times[1,M_1']\times\cdots\times[1,M_l']}$*, where a new set symbol* I' *is associated to each set symbol* I *in* R_1^G*;*
6. *all constraints* $I_1 \uplus \cdots \uplus I_n = [1, M_1'] \times \cdots \times [1, M_k']$ *in* Cts_1 *with* $I_1' \uplus \cdots \uplus I_n' = J \times [1, M_1'] \times \cdots \times [1, M_k']$ *and all constraints* $I_1 \uplus \cdots \uplus I_n = I \times [1, M_1'] \times \cdots \times [1, M_k']$ *in* Cts_1 *with* $I_1' \uplus \cdots \uplus I_n' = I' \times [1, M_1'] \times \cdots \times [1, M_k']$*.*

Let $imm(R_1^G, (i_1, \ldots, i_h) \in J) = Cts_J, H_J^G \wedge \mathcal{E}_J \Rightarrow pred_1(p_1^G)$*. If* $pred_1 = pred_2$*, we define:*

$$R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G = Cts_J \cup Cts_2, H_J^G \wedge H_2^G \wedge (\{\textstyle\bigwedge_{(i_1\ldots,i_h)\in J} p_1^G \doteq p_2^G\} \cup \mathcal{E}_J \cup \mathcal{E}_2) \Rightarrow C_2^G$$

Let I' *and* I'' *be fresh set symbols. Let* $imm(R_1^G, (i_1, \ldots, i_h) \in I') = Cts_{I'}, H_{I'}^G \wedge \mathcal{E}_{I'} \Rightarrow pred_1(p_1^G)$*. If* $pred_1 = pred_2$*, we define:*

$$R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G = Cts_{I'} \cup Cts_2 \cup \{I' \uplus I'' = J\}, (H_{I'}^G \wedge \textstyle\bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_2^G) \wedge H_2^G) \wedge (\{\textstyle\bigwedge_{(i_1,\ldots,i_h)\in I'} p_1^G \doteq p_2^G\} \cup \mathcal{E}_{I'} \cup \mathcal{E}_2) \Rightarrow C_2^G.$$

Second case: $F_0^G = pred_2(p_2^G)$ *(ordinary resolution). If* $pred_1 = pred_2$*, we define* $R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G = Cts_1 \cup Cts_2$, $H_1^G \wedge H_2^G \wedge (\{p_1^G \doteq p_2^G\} \cup \mathcal{E}_1 \cup \mathcal{E}_2) \Rightarrow C_2^G$*.* $R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$ *is undefined.*

Intuitively, the immersion of $R^G = H^G \Rightarrow C^G$ into $(i_1, \ldots, i_h) \in J$ corresponds to $\bigwedge_{(i_1,\ldots,i_h)\in J} H^G(i_1, \ldots, i_h) \Rightarrow C^G(i_1, \ldots, i_h)$, which represents the combination of a distinct instance of R^G for each $(i_1, \ldots, i_h) \in J$, as outlined in Section 3.2. To represent distinct instances of R^G, the free indices i, variables x, and functions ϕ use $(i_1, \ldots, i_h)$ as additional indices or arguments. Sets I should also become functions of the indices $(i_1, \ldots, i_h)$; however, to simplify the syntax of generalized Horn clauses, we avoid sets that depend on indices: when $\bigwedge_{(j_1,\ldots,j_k)\in I\times[1,M_1]\times\cdots\times[1,M_l]} pred(p^G) \in H^G$, instead of writing

$$\textstyle\bigwedge_{(i_1,\ldots,i_h)\in J} \bigwedge_{(j_1,\ldots,j_k)\in I(i_1,\ldots,i_h)\times[1,M_1]\times\cdots\times[1,M_l]} pred(p^G)$$

we write $\bigwedge_{(i_1,\ldots,i_h,j_1,\ldots,j_k)\in I'\times[1,M_1]\times\cdots\times[1,M_l]} pred(p^G)$ where the symbol I' stands for $\{(i_1, \ldots, i_h, j_1, \ldots, j_{k-l}) \mid (i_1, \ldots, i_h) \in J, (j_1, \ldots, j_{k-l}) \in I(i_1, \ldots, i_h)\}$. This leads to point 5 of the definition of immersion, and the corresponding transformation of constraints in point 6. This transformation introduces a minor approximation, since the obtained clause does not keep the information on how the set I' is built.

In order to resolve $R_2^G = Cts_2, F_0^G \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$ with R_1^G upon $F_0^G = \bigwedge_{(i_1,\ldots,i_h)\in J} pred_2(p_2^G)$, we perform the resolution on any non-empty subset I' of J. We distinguish two cases: either I' is the full set J, and we produce the full hyperresolution $R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G$, or I' is a strict subset of J, and we produce the partial hyperresolution $R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$. In the latter case, using the set I'' such that $I' \uplus I'' = J$, R_2^G can also be written $R_2^G = Cts_2 \cup \{I' \uplus I'' = J\}, \bigwedge_{(i_1,\ldots,i_h)\in I'} pred_2(p_2^G) \wedge \bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_2^G) \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$. We can then perform resolution upon $F_{I'}^G = \bigwedge_{(i_1,\ldots,i_h)\in I'} pred_2(p_2^G)$ for the full set I'. We use the immersion to compute a clause $R_{I'}^G$ that corresponds to instances of R_1^G for all $(i_1, \ldots, i_h) \in I'$, and then perform a fairly standard resolution step: we require that the conclusion of $R_{I'}^G$ equals the hypothesis $F_{I'}^G$ of R_2^G and generate the combined clause. (The unification of the conclusion of $R_{I'}^G$ with the hypothesis $F_{I'}^G$ of R_2^G is delayed until the simplification of clauses, described in Section 5.3.)

When $h = 0$ and $J = \{()\}$, $F_0^G = pred_2(p_2^G)$ and J is a singleton, so $R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$ does not exist, and we perform an ordinary resolution step. (Immersion is not necessary.)

THEOREM 3 (RESOLUTION). *Let* $\Gamma_1 \vdash R_1^G$ *and* $\Gamma_2 \vdash R_2^G$ *be two well-typed clauses, such that* $R_2^G = Cts_2, F_0^G \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$*. We have* $\Gamma_{\text{Full}} \vdash R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G$ *and* $\Gamma_{\text{Part}} \vdash R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$ *for some type environments* Γ_{Full} *and* Γ_{Part} *(when these clauses are defined).*

Let T_i *be an environment for* $\Gamma_i \vdash R_i^G$ *such that* $R_i^{GT_i}$ *is defined, for* $i = 1, 2$*. Let* $F_0 \in \text{MGU}(\mathcal{E}_2^{T_2}) F_0^{GT_2}$ *be a fact in the hypothesis of* $R_2^{GT_2}$ *that comes from the translation* F_0^G*. Suppose that* $R_1^{GT_1}$ *and* $R_2^{GT_2}$ *can be resolved upon* F_0 *into the clause* R*. Then* R *is equal to* $(R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G)^T$ *or* $(R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G)^T$ *up to renaming of variables, for some environment* T *for* $\Gamma_{\text{Full}} \vdash R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G$ *(resp.* $\Gamma_{\text{Part}} \vdash R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$*).*

Example 2. Let us consider the following two clauses:

$$R_1^G = \text{att}(x) \Rightarrow \text{att}(sha1(x)) \tag{5}$$

$$R_2^G = \textstyle\bigwedge_{i\in[1,L]} \text{att}(sha1(cont_{\phi(i)})) \wedge \text{att}(sign(list(i \leq L, sha1(cont_{\phi(i)})), \mathsf{sk})) \Rightarrow \text{event}(e(r)) \tag{6}$$

To compute the partial hyperresolution of (6) with (5) for a set I, we compute the immersion of (5) into $i \in I$: we add to x the index i and we add a conjunction $\bigwedge_{i\in I}$ in the hypothesis (replacing the omitted empty conjunction $\bigwedge_{()\in\{()\}}$):

$$\textstyle\bigwedge_{i\in I} \text{att}(x_i) \Rightarrow \text{att}(sha1(x_i)) \tag{7}$$

Partial hyperresolution yields

$$R_1^G \circ_{F_0}^{\text{Part}} R_2^G = \{I \uplus I' = [1, L]\}, \textstyle\bigwedge_{i\in I} \text{att}(x_i) \wedge \bigwedge_{i\in I'} \text{att}(sha1(cont_{\phi(i)})) \wedge \text{att}(sign(list(i \leq L, sha1(cont_{\phi(i)})), \mathsf{sk})) \wedge \{\textstyle\bigwedge_{i\in I} sha1(x_i) \doteq sha1(cont_{\phi(i)})\} \Rightarrow \text{event}(e(r)) \tag{8}$$

This clause will be further simplified by the transformations given in Section 5.3.

5.3 Simplification of Clauses

We use several additional transformations in order to simplify clauses. Some of these transformations are easily adapted from similar transformations already used in ProVerif. We summarize here the new transformations and refer the reader to the long version of the paper [8] for details on all these transformations.

5.3.1 Unification

Instead of performing unification as part of resolution, we simplify equations a posteriori. The main simplifications are as follows. (The others are detailed in the long version of the paper [8].)

- We decompose equations. We replace the equation $\mathcal{C}\, f(p_1^G, \ldots, p_k^G) \doteq f(p_1'^G, \ldots, p_k'^G)$ with $\mathcal{C}\, p_1^G \doteq p_1'^G$, $\ldots$, $\mathcal{C}\, p_k^G \doteq p_k'^G$. We replace the equation $\mathcal{C}\, a_\iota^M[p_1^G, \ldots, p_k^G] \doteq a_{\iota'}^{M'}[p_1'^G, \ldots, p_k'^G]$ with $\mathcal{C}\, p_1^G \doteq p_1'^G$, $\ldots$, $\mathcal{C}\, p_k^G \doteq p_k'^G$, $\mathcal{C}\, \iota \doteq \iota'$ and replace every occurrence of M' in the clause with M. We replace the equation $\bigwedge_{(i_1,\ldots,i_h)\in J} \mathit{list}(i \leq M, p^G) \doteq \mathit{list}(i \leq M', p'^G)$ with $\bigwedge_{(i_1,\ldots,i_h,i)\in J\times[1,M]} p^G \doteq p'^G$ and replace every occurrence of M' in the clause with M.

  The equation $\bigwedge_{(i_1,\ldots,i_{h'})\in J} \mathit{list}(i \leq M, p^G) \doteq \langle p_1^G, \ldots, p_h^G\rangle$ can be handled in two ways. The preferred solution is to instantiate M into the integer h. In this instantiation, variables x that have an index of type $[1, M]$ become h variables $x\_1, \ldots, x\_h$, and functions ϕ that have an argument of type $[1, M]$ become h functions $\phi\_1, \ldots, \phi\_h$. After instantiation, we apply unification again on the obtained clause(s). This solution is applied when the clause contains no function ϕ with a result of type $[1, M]$, no set symbol I with a type that contains $[1, M]$, and no name with an index of type $[1, M]$. (Otherwise, the instantiated clause may not fit in our language of clauses.) When this instantiation cannot be applied, we replace the equation with

  $$\bigwedge_{(i_1,\ldots,i_{h'},i)\in J\times[1,M]} x_{i_1,\ldots,i_{h'},i} \doteq p^G,$$
  $$\bigwedge_{(i_1,\ldots,i_{h'},i)\in I_1} x_{i_1,\ldots,i_{h'},i} \doteq p_1^G, \ldots,$$
  $$\bigwedge_{(i_1,\ldots,i_{h'},i)\in I_h} x_{i_1,\ldots,i_{h'},i} \doteq p_h^G$$

  and add the constraint $I_1 \uplus \ldots \uplus I_h = J \times [1, M]$, where x is a fresh variable and $I_1, \ldots, I_h$ are fresh set symbols. Intuitively, the variable $x_{i_1,\ldots,i_{h'},i}$ contains the i-th element of the list. This solution introduces an approximation: we remember that the elements of the list are p_1^G, $\ldots$, p_h^G but we forget their order and their number of repetitions.

  The clause is removed when the two sides of the equation have different root function symbols. The equation is removed when its two sides are identical.

- When there is an equation $\bigwedge_{(i_1,\ldots,i_h)\in J} x_{\iota_1,\ldots,\iota_k} \doteq p^G$, we replace $x\_$ with its value (provided the indices of x match, and no occurrence of x in p'^G can be replaced; the latter condition serves to avoid loops).

- When there is an equation $i \doteq \iota$ and i does not occur in ι, we replace i with ι and delete the equation.

- When one of the two sides of an equation is a variable that does not occur anywhere else in the clause, we remove that equation.

Example 3. Applying these transformations, the clause (8) becomes

$$R_1^G \circ_{F_0}^{\mathrm{Part}} R_2^G = \{I \uplus I' = [1, L]\},\ \textstyle\bigwedge_{i\in I} \mathrm{att}(\mathit{cont}_{\phi(i)}) \wedge$$
$$\textstyle\bigwedge_{i\in I'} \mathrm{att}(\mathit{sha1}(\mathit{cont}_{\phi(i)})) \wedge \qquad (9)$$
$$\mathrm{att}(\mathit{sign}(\mathit{list}(i \leq L, \mathit{sha1}(\mathit{cont}_{\phi(i)})), \mathsf{sk})) \Rightarrow \mathrm{event}(e(r))$$

Indeed, the equation $\bigwedge_{i\in I} \mathit{sha1}(x_i) \doteq \mathit{sha1}(\mathit{cont}_{\phi(i)})$ first becomes $\bigwedge_{i\in I} x_i \doteq \mathit{cont}_{\phi(i)}$, then $\mathit{cont}_{\phi(i)}$ is substituted for x_i, and the equation is removed.

The unication algorithm is not complete, in that for some complex equations, it may be unable to use the equations to simplify the clause or to remove it, even though that would be sound. This limitation may be lead to false attacks. In practice, the algorithm is still precise enough to be able the prove all desired properties of our case studies (Section 6).

5.3.2 Merging of Sets

When a clause uses two disjoint sets I and I' in the same facts and equations (up to renaming), we merge these two sets into a single set I''. This transformation is key to obtain termination of the algorithm: when a clause R^G is obtained by partial hyperresolution of two clauses R_1^G and R_2^G, it can be resolved again with R_1^G into R'^G, and so on, which would yield an infinite loop. However, by merging sets in R'^G, we can build a clause that is subsumed by R^G, and so removed, which stops the loop. We illustrate this point on an example. The clause (9) can be resolved again by partial hyperresolution with the clause (6). We obtain:

$$\{I \uplus I' = [1, L], I_1 \uplus I_1' = I'\}, \textstyle\bigwedge_{i\in I} \mathrm{att}(\mathit{cont}_{\phi(i)}) \wedge$$
$$\textstyle\bigwedge_{i\in I_1} \mathrm{att}(\mathit{cont}_{\phi(i)}) \wedge$$
$$\textstyle\bigwedge_{i\in I_1'} \mathrm{att}(\mathit{sha1}(\mathit{cont}_{\phi(i)})) \wedge$$
$$\mathrm{att}(\mathit{sign}(\mathit{list}(i \leq L, \mathit{sha1}(\mathit{cont}_{\phi(i)})), \mathsf{sk})) \Rightarrow \mathrm{event}(e(r))$$

which could be resolved again with (6). However, after replacing the two constraints with $I \uplus I_1 \uplus I_1' = [1, L]$, we merge I and I_1 together into the set I_2:

$$\{I_2 \uplus I_1' = [1, L]\}, \textstyle\bigwedge_{i\in I_2} \mathrm{att}(\mathit{cont}_{\phi(i)}) \wedge$$
$$\textstyle\bigwedge_{i\in I_1'} \mathrm{att}(\mathit{sha1}(\mathit{cont}_{\phi(i)})) \wedge$$
$$\mathrm{att}(\mathit{sign}(\mathit{list}(i \leq L, \mathit{sha1}(\mathit{cont}_{\phi(i)})), \mathsf{sk})) \Rightarrow \mathrm{event}(e(r))$$

This clause will be removed by the algorithm of Section 5.4, because it is subsumed by (9), so the loop is avoided.

The merging of sets is not always able to avoid loops that come from partial hyperresolution, in particular when a fresh bound M is created at each partial hyperresolution step. This limitation is the main new cause of non-termination, but its impact is limited in practice since the selection function is tuned to avoid partial hyperresolution when possible.

5.3.3 Soundness of Simplification

Let $\textsc{simp}(\Gamma \vdash R^G)$ be the set of well-typed generalized Horn clauses obtained by simplifying R^G as described above. This function is naturally extended to sets of clauses. The following theorem shows the soundness of SIMP.

$\text{SATUR}^G(\mathcal{R}_0^G) =$

1. $\mathcal{R}^G \leftarrow \text{ELIM}^G(\text{SIMP}(\mathcal{R}_0^G))$.
2. Repeat until a fixpoint is reached:
   for each $\Gamma \vdash R^G \in \mathcal{R}^G$ such that $sel^G(R^G) = \emptyset$,
   for each $\Gamma' \vdash R'^G \in \mathcal{R}^G$, for each $F_0^G \in sel^G(R'^G)$,
   $\mathcal{R}^G \leftarrow \text{ELIM}^G(\text{SIMP}(\{\Gamma_{\text{Full}} \vdash R^G \circ_{F_0^G}^{\text{Full}} R'^G,$
   $\Gamma_{\text{Part}} \vdash R^G \circ_{F_0^G}^{\text{Part}} R'^G\}) \cup \mathcal{R}^G)$.
3. Return $\{\Gamma \vdash R^G \in \mathcal{R}^G \mid sel^G(R^G) = \emptyset\}$.

Figure 4: New Resolution Algorithm

THEOREM 4. *Let $\Gamma \vdash R^G$ be a well-typed generalized Horn clause. For all environments T for $\Gamma \vdash R^G$, if R^{GT} is defined, then there exist a clause $\Gamma' \vdash R'^G \in \text{SIMP}(\Gamma \vdash R^G)$ and an environment T' for $\Gamma' \vdash R'^G$ such that $R'^{GT'} \sqsupseteq R^{GT}$.*

5.4 Extension of the Resolution Algorithm

The resolution algorithm for generalized Horn clauses simulates the one for standard Horn clauses. We need to define a generalized selection function sel^G.

DEFINITION 9 (GENERALIZED SELECTION FUNCTION). *A generalized selection function is a function from generalized Horn clauses to sets of facts, such that $sel^G(Cts, H^G \wedge \mathcal{E} \Rightarrow C^G) \subseteq H^G$.*

Similarly to the case of standard Horn clauses, a good generalized selection function does not select m-event(p^G) nor $\text{att}(x_{\iota_1,\ldots,\iota_h})$. Furthermore, resolution is much simpler for facts $\text{att}(p^G)$ without conjunction than for facts with conjunction, so we select a fact without conjunction when possible. Hence, we use the following selection function:

$sel^G(Cts, H^G \wedge \mathcal{E} \Rightarrow C^G) =$

$$\begin{cases} \{\text{att}(p^G)\} & \text{if } \text{att}(p^G) \in H^G \text{ for some non-variable } p^G \\ \emptyset & \text{if the first case does not apply and} \\ & C^G = \text{att}(p^G) \text{ for some non-variable } p^G \\ \{\mathcal{C}\,\text{att}(p^G)\} & \text{if the previous cases do not apply and} \\ & \mathcal{C}\,\text{att}(p^G) \in H^G \text{ for some non-variable } p^G \\ \emptyset & \text{otherwise} \end{cases}$$

The resolution algorithm is shown in Figure 4. It mimics the algorithm of Figure 2: $\text{SATUR}^G(\mathcal{R}_0^G)$ first inserts in $\mathcal{R}^G$ the clauses in $\mathcal{R}_0^G$ after elimination of subsumed clauses by ELIM^G: when $\Gamma' \vdash R'^G$ subsumes $\Gamma \vdash R^G$ and both $\Gamma \vdash R^G$ and $\Gamma' \vdash R'^G$ are in $\mathcal{R}^G$, $\Gamma \vdash R^G$ is removed by $\text{ELIM}^G(\mathcal{R}^G)$. Next, the resolution algorithm performs hyperresolution until a fixpoint is reached. Finally, it returns the clauses in $\mathcal{R}^G$ with no selected hypothesis.

Let $\mathcal{F}_{\text{me}}$ be any set of facts of the form m-event(p).

THEOREM 5. *Let F be a well-typed closed fact and $\mathcal{R}_0^G$ a set of well-typed generalized Horn clauses. If F is derivable from $\mathcal{R}_0^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$, then F is derivable from*

$$(\text{SATUR}^G(\mathcal{R}_0^G))^{\mathcal{T}} \cup \mathcal{F}_{\text{me}}.$$

This result shows the soundness of our algorithm. Its proof is adapted from the proof of SATUR (Theorem 1) and uses the soundness theorems for resolution, simplification, and subsumption.

To prove that a closed fact $\text{att}(p^{GT})$ is not derivable from $\mathcal{R}_1^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$, we use the following result, where att$'$ is a new predicate:

COROLLARY 3. *If $\text{SATUR}^G(\mathcal{R}_1^G \cup \{\text{att}(p^G) \Rightarrow \text{att}'(p^G)\})$ contains no clause of the form $\Gamma \vdash Cts, H^G \wedge \mathcal{E} \Rightarrow \text{att}'(p'^G)$, then, for all environments T, $\text{att}(p^{GT})$ is not derivable from $\mathcal{R}_1^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$.*

This corollary is proved like Corollary 1, and makes it possible to prove that the protocol preserves the secrecy of p, for lists of any length.

COROLLARY 4. *Suppose that all clauses of $\text{SATUR}^G(\mathcal{R}_1^G)$ that conclude $\text{event}(e(p^G))$ for some p^G are of the form $\Gamma \vdash Cts, \bigwedge_{(i_1,\ldots,i_h)\in[1,M_1]\times\cdots\times[1,M_h]} \text{m-event}(e'(p'^G)) \wedge H^G \wedge \mathcal{E} \Rightarrow \text{event}(e(\rho p'^G))$ for some Γ, Cts, $i_1, \ldots, i_h$, $M_1, \ldots, M_h$, H^G, $\mathcal{E}$, p'^G, and some substitution ρ that maps indices $i_1, \ldots, i_h$ to index terms. Then, for all $\mathcal{F}_{\text{me}}$, for all p, if $\text{event}(e(p))$ is derivable from $\mathcal{R}_1^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$, then $\text{m-event}(e'(p)) \in \mathcal{F}_{\text{me}}$.*

This corollary makes it possible to prove that the protocol satisfies the correspondence "if $e(x)$ has been executed, then $e'(x)$ has been executed".

6. EXPERIMENTAL RESULTS

Our algorithm cannot prove authentication for our running example. Indeed, the result of SATUR^G contains a clause of the form

$$\text{att}(r) \wedge \text{m-event}(b(\mathsf{Req})) \wedge \mathcal{E} \Rightarrow \text{event}(e(r))$$

where r does not occur in $\mathcal{E}$. This clause does not satisfy Corollary 4: the event $e(r)$ may be executed for any r that the adversary has, even though only $b(\mathsf{Req})$ has been executed. This clause corresponds to the wrapping attack described in Section 3.1.

In contrast, our algorithm proves authentication for the corrected version of our running example. The only clause in the result of SATUR^G that concludes $\text{event}(e(p^G))$ is of the form

$$\text{m-event}(e(\mathsf{Req})) \wedge \mathcal{E} \Rightarrow \text{event}(e(\mathsf{Req}))$$

Hence, using Corollary 4, we can conclude that, if event $e(r)$ has been executed, then event $b(r)$ has been executed as well. (In our case, the only possible value of r is the request Req of the client.)

In addition to this protocol, we tested our tool on the XML protocols studied in [5]: password digest, password-based signature, X.509 signature, and firewall-based authentication. These are web services security protocols, whose goal is to authenticate a client to a web service. The first two protocols depend on password-based authentication: a password is shared between the user and the server. In the first protocol, a digest of the password, a nonce, and a timestamp is sent by the client to the server. In the second protocol, the client signs its request using a signature key generated from the password. The X.509 signature protocol uses public-key signatures based on X.509 certificates. Finally, the firewall-based authentication protocol uses a SOAP-level firewall in addition to the server and the client. The client uses a password-based signature; the firewall verifies this signature,

and when this authentication succeeds, adds a new `firewall` header, signed using its X.509 certificate, indicating that it has authenticated the client. For all these protocols, we automatically proved the authentication property proved manually in [5].

Finally, we also modeled the Asokan-Ginzboorg group protocol [1], and proved the secrecy of the session key. This property was already proved in [19], but on a model less precise than ours. For all these examples, our tool terminates in less than 0.5 s on a MacBook Air 1.4 Ghz. The input files for all these case studies can be found at [8].

7. CONCLUSION AND FUTURE WORK

We have proposed a new type of clauses, generalized Horn clauses, useful to represent protocols that manipulate lists of unbounded length. We have adapted the definitions previously introduced for Horn clauses to these new clauses. We have thus obtained a new algorithm for verifying secrecy and authentication properties for protocols with lists, which we have proved correct and implemented. We have successfully tested our tool on several XML protocols.

ProVerif supports a variant of the applied pi calculus for modeling protocols. However, in this paper, we need to model protocols with generalized Horn clauses. We plan to extend the input language of ProVerif to model protocols with lists of unbounded length, and to translate it automatically to generalized Horn clauses. We also plan to provide an input format with messages in XML, in the style of the tool TulaFale [6], with an extension to unbounded lists.

8. ACKNOWLEDGMENTS

This work was partly supported by the ANR project ProSe (decision number ANR-2010-VERS-004-01). It was partly done while the authors were at École Normale Supérieure, Paris.

9. REFERENCES

[1] N. Asokan and P. Ginzboorg. Key agreement in ad hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[2] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–100. North Holland, 2001.

[3] M. Backes, S. Mödersheim, B. Pfitzmann, and L. Viganò. Symbolic and cryptographic analysis of the secure WS-ReliableMessaging scenario. In *FoSSaCS'06*, volume 3921 of *LNCS*, pages 428–445. Springer, 2006.

[4] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML signature syntax and processing (second edition). Available at `http://www.w3.org/TR/xmldsig-core/`.

[5] K. Bhargavan, C. Fournet, and A. D. Gordon. A semantics for web services authentication. In *POPL'04*, pages 198–209. ACM, 2004.

[6] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. Tulafale: A security tool for web services. In *FMCO'03*, volume 3188 of *LNCS*, pages 197–222. Springer, 2004.

[7] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, July 2009.

[8] B. Blanchet and M. Paiola. Automatic verification of protocols with lists of unbounded length. Long version, files available at `http://prosecco.gforge.inria.fr/personal/bblanche/publications/BlanchetPaiolaCCS13.html`, 2013.

[9] A. Brown, B. Fox, S. Hada, B. LaMacchia, and H. Maruyama. SOAP security extensions: Digital signature. Available at `http://www.w3.org/TR/SOAP-dsig/`.

[10] J. Bryans and S. Schneider. CSP, PVS and recursive authentication protocol. In *DIMACS Workshop on Formal Verification of Security Protocols*, Sept. 1997.

[11] N. Chridi, M. Turuani, and M. Rusinowitch. Constraints-based Verification of Parameterized Cryptographic Protocols. Research Report RR-6712, INRIA, 2008.

[12] N. Chridi, M. Turuani, and M. Rusinowitch. Decidable analysis for a class of cryptographic group protocols with unbounded lists. In *CSF'09*, pages 277–289. IEEE Computer Society, 2009.

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

[14] L. Georgieva, U. Hustadt, and R. A. Schmidt. A new clausal class decidable by hyperresolution. In *CADE-18*, volume 2392 of *LNCS*, pages 260–274. Springer, 2002.

[15] E. Kleiner and A. W. Roscoe. On the relationship between web services security and traditional protocols. In *MFPS 21*, volume 155 of *Electronic Notes in Theoretical Computer Science*, pages 583–603, 2006.

[16] R. Küsters and T. Truderung. On the automatic analysis of recursive security protocols with XOR. In W. Thomas and P. Weil, editors, *STACS'07*, volume 4393 of *LNCS*, pages 646–657. Springer, 2007.

[17] G. Lowe. A hierarchy of authentication specifications. In *CSFW'97*, pages 31–43. IEEE Computer Society, June 1997.

[18] M. Mcintosh and P. Austel. XML signature element wrapping attacks and countermeasures. In *SWS'05*, pages 20–27. ACM, 2005.

[19] M. Paiola and B. Blanchet. Verification of security protocols with lists: from length one to unbounded length. In *POST'12*, volume 7215 of *LNCS*, pages 69–88. Springer, 2012.

[20] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *CSFW'97*, pages 84–95. IEEE Computer Society Press, 1997.

[21] A. W. Roscoe and E. Kleiner. Web Services Security: a preliminary study using Casper and FDR. In *Automated Reasoning for Security Protocol Analysis (ARSPA'04)*, 2004.

[22] T. Truderung. Selecting theories and recursive protocols. In *CONCUR 2005*, volume 3653 of *LNCS*, pages 217–232. Springer, 2005.