Cryptographic Protocol Verification Using Tractable Classes of Horn Clauses

Helmut Seidl and Kumar Neeraj Verma

Institut für Informatik, TU München, Germany
{seidl,verma}@in.tum.de

Abstract. We consider secrecy problems for cryptographic protocols modeled using Horn clauses and present general classes of Horn clauses which can be efficiently decided. Besides simplifying the methods for the class of flat and one-variable clauses introduced for modeling of protocols with single blind copying [7,25], we also generalize this class by considering k-variable clauses instead of one-variable clauses with suitable restrictions similar to those for the class $\mathcal{S}^+$. This class allows to conveniently model protocols with joint blind copying. We show that for a fixed k, our new class can be decided in DEXPTIME, as in the case of one variable.

1 Introduction

Cryptographic protocols are today widely deployed for securing communication in various applications notably electronic commerce. These protocols are rules for exchanging messages, and rely on certain cryptographic algorithms like encryption and decryption of messages using keys. Experience has shown that even very simple protocols can have subtle flaws which are hard to detect by manual analysis. The classic example is that of the Needham-Schroeder public key protocol [19], which was considered to be correct until a bug was found 15 years after its publication [17]. Such experiences have recently led to considerable work on techniques for automatic verification of cryptographic protocols.

An important point in the example cited above is that the attack does not consist in breaking any underlying cryptographic algorithm like encryption or decryption. These algorithms are assumed to be perfect, and the attack only involves simple techniques like replaying intercepted messages, encrypting and decrypting messages with known keys, etc. Such considerations have led to the use of the so-called Dolev-Yao model [10] which essentially consists in treating cryptographic algorithms as black boxes, and assuming agents to be communicating over a completely hostile network, in the sense that any message passing through it can be intercepted or deleted by an all-powerful adversary. Further, new messages known to the adversary could be sent to agents, for example with the aim of impersonating as another honest agent. These assumptions allow us to treat messages as terms and allow us to design symbolic techniques, e.g. those based on automata and logic, for analyzing these protocols.

The complexity of verifying such protocols is due to several factors, like potentially infinite number of sessions of the same protocol between different agents, possibly in

T. Reps, M. Sagiv, and J. Bauer (Eds.): Wilhelm Festschrift, LNCS 4444, pp. 97–119, 2007.

parallel with complex interactions between them, possibly infinite number of agents, infinite possibilities for messages etc. Several kinds of restrictions are considered in order to have efficient algorithms for the verification problem. For example if the number of sessions is bounded, then checking for the presence of an attack is NP-complete [24]. This may be helpful for detecting some simple attacks during the design phase, considering that most known attacks involve very small number of sessions. However we are often interested in *certifying* protocols, i.e. guaranteeing that there are no attacks involving any number of sessions. This is a difficult problem, and remains undecidable even with serious restrictions [5]. This class of problem is often modeled using Horn clauses of first order logic [2,7,25,26] and related formalisms like automata and set constraints [5,16,18]. In order to obtain tractable problems, one often uses *safe* abstractions, i.e. those that detect all attacks, but can possibly introduce false attacks. Further, algorithms are also designed for specific classes of protocols which can be efficiently treated.

In this paper we present several interesting classes of Horn clauses which can be efficiently treated. We consider secrecy questions about cryptographic protocols modeled as satisfiability problems about Horn clauses. Our goal is to have as general classes of clauses as possible, which can be decided in single exponential time. We consider single exponential time to be the feasibility limit for this class of protocol verification problems, as in [25]. For our modeling, we typically use a unary predicate known that represents the set of messages that the adversary can know after arbitrary many sessions of the protocol. The secrecy question is then whether $\mathsf{known}(t)$ does not hold for a certain message t. In particular we adopt the approach of normalization of Horn clauses, i.e. given a set of clauses we transform it to a set of simpler clauses on which various kinds of queries, e.g. whether some ground $P(t)$ holds, can be easily evaluated (in polynomial time). The classes we deal with are interesting in the sense that they are all general classes which still allow exponential time normalization, and further, no decidable generalizations of these classes seem to be evident.

Compared to related work on verification of cryptographic protocols, note that our interest is in certifying protocols, whereas approaches dealing with a bounded number of sessions [24] only help to find some attacks involving small number of sessions. Approaches involving general classes of Horn clauses [2] seem to work well in practice, but no termination guarantees are offered for the algorithm. Most similar to our approach are works like [5,26,7] which try to find decidable classes of automata or clauses which however may not model all protocols. In these approaches, the infinitely many nonces (random numbers), generated in different protocol sessions, are typically abstracted to finitely many nonces. This is a safe abstraction. A more precise modeling may represent these nonces not as constants but as functions of the history (all previous sessions). But this leads to clauses which are difficult to treat efficiently. Another possibility is to use linear logic instead of classical logic [3]. Finally, certain decidability results have been obtained for general classes of protocols with infinitely many sessions, by putting certain tagging constraints on the protocols [4,21,22].

In the rest of the section we describe the classes dealt with in the paper. We start with the class $\mathcal{H}_1$ [20] which generalizes uniform Horn clauses [12] and further allows us to express operations on relations like Cartesian products of relations, transitive closures

and permutations of components, still allowing exponential time normalization. This also shows that despite their generality, $\mathcal{H}_1$ describes only (Cartesian products of) regular tree languages. We then consider the related class of flat clauses which models various extensions of tree automata like two-wayness and alternation, permutations and duplications of components, but also equality constraints between brothers which is disallowed by $\mathcal{H}_1$. Compared to the various classes of flat clauses considered in the literature, we allow maximal generality. In particular different functional terms may contain different sets of variables. We show that this class can still be normalized in exponential time.

While these two classes have very general features, they do not easily model specific kinds of actions occurring in cryptographic protocols. In particular, we consider cryptographic protocols in which each step involves copying of at most one piece of unknown message. This class was considered in [7,9] and was modeled using flat clauses and clauses involving at most one variable. The original upper bound provided in [7,9] for this class of clauses and protocols was triply exponential. We introduced new techniques like on-the-fly decomposition of one-variable terms to show that this class can in fact be decided in single exponential time, giving us the optimal complexity for the class of protocols as well as of clauses [25]. The proof in [25] used resolution and dealt only with the satisfiability problem (in the Horn as well as non-Horn cases).

The main contribution of the present paper therefore is to simplify the algorithm for the Horn case as well as to obtain a normalized set of clauses. Furthermore, we consider protocols where, instead of just one piece of unknown message, several pieces of unknown messages can be copied in a protocol step. Multiple blind copying, however, easily leads to undecidability. We show here, though, that multiple blind copying can be dealt with given that copying is always *joint*, i.e., all copied parts always occur in every non-ground functional subexpression of the protocol. For that, we introduce a generalization of the class of flat and one-variable Horn clauses. The new class allows k-variable instead of one-variable clauses, but with some restriction, similar to those for the class $\mathcal{S}^+$ [11]. We show that our ideas for the one-variable case can be suitably generalized to give an exponential normalization procedure also for the k-variable case.

Several proofs have been ommitted in order to keep the paper readable. They can be found in a longer version available from the authors and at `http://www2.in.tum de/ verma`.

2 Horn Clauses and Cryptographic Protocols

A *Horn clause* is of the form $A \Leftarrow A_1 \wedge \ldots \wedge A_n$ where $n \geq 0$ and A, A_i are atoms of the form $P(t_1, \ldots, t_k)$ where P is a k-ary predicate for $k \geq 0$ and t_i are terms built up from variables and function symbols of fixed arities. A is called the *head*, and the remaining part the *body* of the clause. *Substitutions* map variables to terms. Application of a substitution σ to a term, atom, clause or substitution M is denoted $M\sigma$ and is defined as usual. Composition of substitutions is defined as usual: $M(\sigma_1 \ldots \sigma_k)$ should be read as $(\ldots(M\sigma_1)\ldots)\sigma_k$. The *least Herbrand model* H of a set S of clauses is a set of ground atoms (i.e. those without variables) inductively defined as: if σ is a ground substitution (mapping variables to ground terms), if clause $A \Leftarrow A_1 \wedge \ldots \wedge A_n \in S$

and if each $A_i\sigma \in H$ then $A\sigma \in H$. Further we are interested in considering clauses as representing generalizations of tree automata. Accordingly if $P(t_1, \ldots, t_n)$ is in the least Herbrand model of a set of clauses then we also say that the *state* P *accepts* the tuple $(t_1, \ldots, t_n)$. We also say that $P(t_1, \ldots, t_n)$ holds. Hence given a set of Horn clauses, we can talk of questions like *membership* (is a given tuple accepted at a given state), *non-emptiness* (does a given state accept at least one tuple) or *intersection-non-emptiness* (do two given states of the same arity accept at least one common tuple?). Another approach is to treat these questions as questions about satisfiability of a set of clauses. For example a term $P(t_1, \ldots, t_n)$ holds iff the set of clauses, together with the clause $\bot \Leftarrow P(t_1, \ldots, t_n)$ is *unsatisfiable*, i.e. $\bot$ is present in the least Herbrand model, where $\bot$ is a special zero-ary predicate representing a contradiction. Our approach to dealing with these questions is to convert the set of clauses into another equivalent set of simple clauses, which we call normal clauses, on which such queries can be easily evaluated (in polynomial time). Hence our emphasis in this paper is to give normalization algorithms for the clause sets that we consider.

For modeling of cryptographic protocols, we assume at least the binary functions $\{\_\}\_$ and $\langle \_, \_ \rangle$ denoting encryption and pairing. Here is the Needham-Schroeder protocol, mentioned in the introduction, in standard notation. Message $\{\langle x, y\rangle\}_k$ is abbreviated as $\{x, y\}_k$.

$$\begin{aligned} A &\rightarrow B : \{A, N_A\}_{K_B} \\ B &\rightarrow A : \{N_A, N_B\}_{K_A} \\ A &\rightarrow B : \{N_B\}_{K_B} \end{aligned}$$

The meaning of the protocol is as follows. The notation $A \rightarrow B : M$ denotes agent A sending message M to agent B. Inside messages, A and B represent the identities of respective agents, K_A and K_B their public keys. Public keys are known to everyone and messages encrypted with the public key are decrypted with the corresponding private keys (which are known only to the respective agents), and vice-versa. In the first step, A starts a session with B by sending his identity and a nonce (a random number) N_A that he generates, both encrypted with the public key of B. B decrypts the message and sends back the received nonce with a nonce that he generates, both encrypted again appropriately. This can be considered as a proof that B got the original message and sent this message as a reply. A sends back the received nonce, again encrypted, as a confirmation. Such a protocol is usually executed to establish authenticity of the communicating parties before going ahead with some further transaction. For example B could be a bank which is contacted by client A for some monetary transaction. Hence they may execute this protocol at the beginning to agree on secret values N_A and N_B which is expected to remain unknown to third parties. Further encrypted messages dealing with some financial transactions would then include these secret values as proof that these messages have really been generated by A and B and not by some adversary pretending to be A or B. While it may not be apparent at first glance, we have mentioned that the protocol has a flaw discovered 15 years after its publication. The attack involves two parallel sessions, one in which the adversary plays the role of B and another in which he plays the role of A but impersonating as the agent playing A's role in the first session. This kind of attack is also known as a *man-in-the-middle attack*.

We present here a modeling of this protocol using Horn clauses. Our modeling will only deal with the secrecy problem, i.e. whether some message is known to the adversary or not. Note however that other security questions are also interesting. For example authenticity talks about questions like whether some message received by a given agent was actually sent by another given agent. When we restrict ourself to the secrecy problem then we only need to consider three agents in our model for finding whether the above protocol is secure. This is a consequence of a result [8,7]) stating that under some mild assumptions on the protocol and for many security properties including secrecy, we need to consider a very small number (which is calculable from the protocol and the security property) of agents in our modeling in order to find whether there exists an attack. For the example protocol we need to consider two honest and one dishonest agent. We now need to consider all possible sessions between these bounded number of agents. For every two agents u and v in our model, we have infinitely many sessions in which u is the initiator (i.e. plays the role of A) and v is the responder (i.e. plays the role of B). In each of these sessions we have a fresh nonce. We choose two nonces n^1_{uv} and n^2_{uv} to represent these infinitely many nonces. This modeling using only finitely many nonces is a safe abstraction. Corresponding to the three steps in the protocol we have the following clauses for all agents u and v. Our intention is that $\mathsf{known}(m)$ should hold exactly for messages m which the adversary can know.

$$\begin{array}{ll} \mathsf{known}(\{\langle u, n^1_{uv}\rangle\}_{k_v}) & \\ \mathsf{known}(\{\langle x, n^2_{uv}\rangle\}_{k_u}) & \Leftarrow \mathsf{known}(\{\langle u, x\rangle\}_{k_v}) \\ \mathsf{known}(\{x\}_{k_v}) & \Leftarrow \mathsf{known}(\{\langle n^1_{uv}, x\rangle\}_{k_u}) \end{array}$$

Note that we write such clauses for all (finitely many) pairs of agents in our system, whether they are honest or dishonest agents. The above clauses correspond to our assumption that all messages received by agents are sent by the adversary, and hence should be known to the adversary, and that all messages sent by agents become known to the adversary. For example the second clause represents the fact that v, on receiving a message of the form $\{\langle u, x\rangle\}_{k_v}$, sends a message of the form $\{\langle x, n^2_{uv}\rangle\}_{k_u}$. We use a variable x in place of the nonce sent by u, since this is an unknown for the recipient v, and the adversary could send to him any message of this form, provided he knows such a message. Implicit in this modeling is the fact that we may have arbitrary many sessions between u and v. Note that we can apply the clauses as many times as we like for different values of concerned variables. The essential abstraction is in our modeling here is that infinitely many nonces from different sessions are represented by finitely many nonces. This is a safe abstraction and insecure protocols remain insecure in our modeling. It is possible to make less severe abstractions at the cost of introducing more complex clauses, and we will see some examples in Sections 5 and 7.

We need further clauses to express adversary capabilities. The clauses

$$\begin{array}{ll} \mathsf{known}(\{x\}_y) & \Leftarrow \mathsf{known}(x) \wedge \mathsf{known}(y) \\ \mathsf{known}(x) & \Leftarrow \mathsf{known}(\{x\}_k) \wedge \mathsf{known}(k') \\ \mathsf{known}(\langle x, y\rangle) & \Leftarrow \mathsf{known}(x) \wedge \mathsf{known}(y) \\ \mathsf{known}(x) & \Leftarrow \mathsf{known}(\langle x, y\rangle) \\ \mathsf{known}(y) & \Leftarrow \mathsf{known}(\langle x, y\rangle) \end{array}$$

express the ability of the adversary to perform encryption, decryption, pairing and unpairing, where k' is the private key corresponding to public key k. We have considered k and k' to be constants, although there exist other methods of modeling public and private keys, e.g. letting the term $sk(x)$ represent the private key corresponding to the public key x.

The adversary's knowledge of other data m like agents' names, public keys, etc. are expressed by clauses $\mathsf{known}(m)$. For example if w is some dishonest agent then we have the clause $\mathsf{known}(k'_w)$ to say that private key of w is known to the adversary. Further we have clauses $\mathsf{known}(n^1_{wu})$ and $\mathsf{known}(n^2_{uw})$ for every other agent u, to say that nonces generated by dishonest agents are known to the adversary. Then the secrecy question, whether some message m is known to the adversary is translated to the membership question, whether $\mathsf{known}(m)$ holds. More precisely, our modeling involves safe abstraction, so that if m is known to the adversary then $\mathsf{known}(m)$ holds. Further checking whether $\mathsf{known}(m)$ holds is equivalent to checking whether the clause set together with the clause $\bot \Leftarrow \mathsf{known}(m)$ is unsatisfiable.

In the above particular example, our modeling requires two honest agents a and b and a dishonest agent c. Each of these can be the initiator of a session or the responder of a session. We don't consider the cases where the same agent can be the initiator as well as the responder, though some models and tools allow this possibility, and this can easily be allowed in our Horn clause modeling. After writing the necessary clauses and running a suitable solver, e.g. a solver for the class $\mathcal{H}_1$, we find that $\mathsf{known}(n^1_{ab})$ does not hold. As our modeling involves safe abstraction, we are then sure that the nonce generated by an honest initiator for an honest responder is never leaked to the adversary. On the other hand, $\mathsf{known}(n^2_{ab})$ holds. This suggests the possibility that the nonce created by an honest responder for an honest initiator can be leaked to the adversary. Indeed such a leak happens in the man-in-the-middle attack mentioned above. More precisely, in this attack, the nonce n^2_{ab} is generated by the agent b because he is fooled into believing that the agent a has started a session with him.

3 $\mathcal{H}_1$ and Strongly Recognizable Relations

A relation on ground terms is called *strongly recognizable* if it can be described as finite union of Cartesian products of recognizable languages (i.e. languages accepted by tree automata). In case of unary relations, strongly recognizable relations are just the recognizable languages. The class $\mathcal{H}_1$ presented in this section allows general forms of clauses which still describe only strongly recognizable relations. Although $\mathcal{H}_1$ is not specifically meant for modeling some particular class of protocols, the advantage of this class is that we have a systematic way of safely abstracting arbitrary clauses to clauses in this class [14].

With a clause we associate a *variable dependence graph* whose vertices are the atoms in the body of the clause, and two atoms are neighbors if they have a common variable. Two variables are called *connected* if they occur within connected atoms. In particular two variables in the same atom are connected. A clause has property $H1$ if

1. the head is linear (i.e. no variables occur twice in it).
2. if variables x and y occur in the head and are connected in the body then they are siblings in the head.

Here we call two variables as siblings if they occur as arguments of a common father. Hence x, y are connected in $P(x, y)$ and in $Q(f(x, a, y))$. The class $\mathcal{H}_1$ consists of finite sets of $H1$ clauses. $\mathcal{H}_1$ allows clauses which can express various operations on relations, like arbitrary projections through constructors, permutations of components, and compositions of relation [20]:

$$\begin{array}{ll} P(x,y) & \Leftarrow Q(f(x,y,z)) \\ Q(x,y,z) & \Leftarrow Q(y,z,x) \\ P(x,z) & \Leftarrow Q(x,y) \wedge R(y,z) \end{array}$$

However they still describe only strongly recognizable relations. To show this we give a procedure that converts the set of clauses into a new set of simple clauses, that we call *normal clauses*. *Normal* H1 clauses are $H1$ clauses of the form

1. $P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1) \wedge \ldots \wedge P_n(x_n)$.
2. $P(x_1, \ldots, x_n) \Leftarrow P_1(x_1) \wedge \ldots \wedge P_n(x_n)$ where $n \neq 1$.

where $x_1, \ldots, x_n$ are pairwise distinct variables. Here we allow $n = 0$ to take care of nullary symbols and predicates. Clearly a set of normal $H1$ clauses can describe only strongly recognizable relations, and conversely every strongly recognizable relation can be described by a set of normal $H1$ clauses. We now show that any set of $H1$ clauses can be normalized, i.e. converted to an equivalent set of normal $H1$ clauses, implying that $\mathcal{H}_1$ describes exactly strongly recognizable relations.

Theorem 1 ([20,14]). *A set of clauses in $\mathcal{H}_1$ can be normalized in DEXPTIME.*

Proof: First we ensure that all variables in the head occur also in the body, by adding atoms $P(x)$ in the body where P is a fresh predicate defined to accept all terms. Then we ensure that every head is of the form $P(f(x_1, \ldots, x_n))$ or $P(x_1, \ldots, x_n)$. For example the clause $P(f(x, y), z) \Leftarrow P_1(x, x') \wedge P_2(x', y) \wedge P_3(z, z')$ is replaced by clauses $P(x, z) \Leftarrow P'(x), P_3(z, z')$ and $P'(f(x, y)) \Leftarrow P_1(x, x') \wedge P_2(x', y)$ where P' is a fresh predicate. Now it remains to simplify the bodies of clauses.

We use sets $\{P_1, \ldots, P_n\}$ of unary predicates to represent intersections of the unary predicates. P_1 is identified with the set $\{P_1\}$. The normalization procedure essentially consists of using a non-normal clause and a normal clause to produce a new simpler clause and continuing this process until the non-normal clauses are redundant. The following kinds of steps are involved.

- Clauses $T_i(f(x_1, \ldots, x_n)) \Leftarrow S_i^1(x_1) \wedge \ldots \wedge S_i^n(x_n)$ produce clause $T(f(x_1, \ldots, x_n)) \Leftarrow S_1(x_1) \wedge \ldots \wedge S_n(x_n)$ where $T = \bigcup_i T_i$ and $S_j = \bigcup_i S_i^j$.
- Clauses $h \Leftarrow \mathcal{B} \wedge S(f(t_1, \ldots, t_n))$ and $S(f(x_1, \ldots, x_n)) \Leftarrow S_1(x_1) \wedge \ldots \wedge S_n(x_n)$ produce the clause $h \Leftarrow \mathcal{B} \wedge S_1(t_1) \wedge \ldots \wedge S_n(t_n)$, where $\mathcal{B}$ is used here and elsewhere to denote a conjunction of atoms.
- Clauses $h \Leftarrow \mathcal{B} \wedge S(t_1, \ldots, t_n)$ and $S(x_1, \ldots, x_n) \Leftarrow S_1(x_1) \wedge \ldots \wedge S_n(x_n)$ produce the clause $h \Leftarrow \mathcal{B} \wedge S_1(t_1) \wedge \ldots \wedge S_n(t_n)$ where $n \geq 1$.
- Clause $h \Leftarrow \mathcal{B} \wedge S_1(x) \wedge S_2(x)$ produces the clause $h \Leftarrow \mathcal{B} \wedge (S_1 \cup S_2)(x)$.

- Clause $S(x) \Leftarrow S'(x)$ and $S'(f(x_1, \ldots, x_n)) \Leftarrow S_1(x_1) \wedge \ldots \wedge S_n(x_n)$ produce $S(f(x_1, \ldots, x_n)) \Leftarrow S_1(x_1) \wedge \ldots \wedge S_n(x_n)$.
- Clause $h \Leftarrow \mathcal{B} \wedge S(x)$ produces the clause $h \Leftarrow \mathcal{B}$ if x does not occur in $h \Leftarrow \mathcal{B}$, and if S accepts at least one term using only the normal predicates.

The essential idea in the above and other normalization algorithms is that if at all the clause set contains a non-normal clause which is not redundant, then we consider a minimal derivation which uses some non-normal clause C of the form $A \Leftarrow A_1 \wedge \ldots \wedge A_n$. This clause allows us to derive the atom $A\sigma$ by applying some ground substitution σ. Now we consider the reason why C is not normal. If some A_i is of the form $P(f(t_1, \ldots, t_n))$, then we know that the atom $A_i\sigma$ is derivable using only normal clauses, because of the minimality assumption. The last clause D used in the latter derivation is of the form $P(f(x_1, \ldots, x_n)) \Leftarrow S_1(x_1) \wedge \ldots \wedge S_n(x_n)$. But then a normalization step involving C and D can produce a "simpler" clause which could instead be used for deriving $A\sigma$. A similar argument holds for the case where the body of C contains a non-unary predicate. On the other hand, if C is not normal because two of the atoms in the body are of the form $S_1(x)$ and $S_2(x)$ then we can replace these two atoms by the atom $(S_1 \cup S_2)(x)$. The first rule above intuitively defines the meaning of the fresh predicates $\{P_1, \ldots, P_n\}$. The intuition behind the last two rules is simple, and they remove some other clauses which are not normal. □

Note that $\mathcal{H}_1$ allows for example the modeling of the Needham-Schroeder protocol described above, and more [13]. As our approach always involves safe abstractions, all attacks, in particular the man-in-the-middle attack on the Needham-Schroeder protocol, are found. The normalization procedure further means that for example the secrecy question, whether some term is not accepted at some predicate, can be evaluated in time polynomial on the resulting clause size. Further, subclasses of $\mathcal{H}_1$, for example the class $\mathcal{H}_3$ [20] can be normalized in polynomial time and suffice for the above example protocol. $\mathcal{H}_1$ has also been successfully used to verify real implementations of cryptographic protocols in the C language [15].

4 General Flat Clauses

The class $\mathcal{H}_1$ allows us to represent tree automata, as well as their extensions like alternating tree automata and two-way tree automata [6]. Alternation is described by clauses of the form

$$P(x) \Leftarrow P_1(x) \wedge \ldots \wedge P_n(x)$$

whereas two-way tree automata contain clauses like

$$P(x) \Leftarrow Q(f(x, y, z)), Q_1(y), Q_2(z)$$

The clauses in Section 2 describing abilities of the adversary to perform encryption, decryption, pairing and unpairing are clauses of two-way tree automata, upto some details. Despite its generality, $\mathcal{H}_1$ does not allow for example the clause

$$P(f(x, y, x)) \Leftarrow Q(x) \wedge R(y)$$

Such clauses describe tree automata with equality constraints between brothers [6]. In fact, it may be verified that allowing such clauses will make the class $\mathcal{H}_1$ undecidable.

We now consider a class which is specially suited for describing such features, e.g. tree automata with alternation, two-wayness and equality constraints between brothers. A *general flat clause* is one that contains only atoms of the form $P(x)$ and $P(f(x_1, \ldots, x_n))$. We put no restrictions on the occurrences and repetitions of variables. An example clause is

$$P(f(x, y, x)) \Leftarrow Q(g(y, y, z)) \wedge R(y)$$

Note that we consider only unary predicates. This class of clauses is more general (in the Horn case) than the flat clauses considered in [25]. The complexity for this class is however still DEXPTIME, as for the class in [25]. Among other things, these clauses can model the ability of the adversary to perform operations like encryption, decryption, pairing, unpairing, hashing etc. But we have tried throughout this paper to obtain maximal classes which can be efficiently decided. A general flat clause is called *normal* if it is of the form

$$P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_{i_1}) \wedge \ldots \wedge P_k(x_{i_k})$$

where $\{x_1, \ldots, x_n\} = \{x_{i_1}, \ldots, x_{i_k}\}$ and $x_{i_1}, \ldots, x_{i_k}$ are pairwise distinct. Define *trivial* terms, atoms or clauses to be those in which no function symbols appear.

Theorem 2. *A set of general flat clauses can be normalized in DEXPTIME.*

Proof: We proceed exactly as in the case of $H1$ clauses, by trying to simplify the body. However since the heads can be non-linear, the variables in the non-trivial atoms in clauses can get unified. E.g. clauses $P_1(f(x, y)) \Leftarrow P_2(g(x, y, z)) \wedge P_3(h(x, x))$ and $P_2(g(x, x, z)) \Leftarrow P_4(x) \wedge P_5(z)$ produce $P_1(f(x, x)) \Leftarrow P_3(h(x, x)) \wedge P_4(x) \wedge P_5(z)$. Similarly clauses $P(f(x, y, y)) \Leftarrow P_1(x) \wedge P_2(y)$ and $Q(f(x, x, y)) \Leftarrow Q_1(x) \wedge Q_2(y)$ produce the $\{P, Q\}(f(x, x, x)) \Leftarrow \{P_1, P_2, Q_1, Q_2\}(x)$. Hence in general arbitrary sequences of variables can occur in the non-trivial atoms in a clause. However the number of such atoms is always linearly bounded and the number of variables in clauses is also linearly bounded. Hence only exponential number of clauses are possible. □

5 One Variable Clauses

One-variable clauses are defined to be clauses in which at most one variable occurs. Note that we put no restriction on the number of occurrences of this variable. The following is an example of a one-variable clause.

$$P(f(x, g(h(x), i(x, x)))) \Leftarrow Q(g(x, x)) \wedge R(x)$$

Having dealt with general flat clauses, our next goal is to allow general flat clauses in the presence of one-variable clauses. However we will restrict the form of general flat clauses that we consider. This is done in the next section. In this section we show how to deal with just one-variable clauses. The main motivation is that this allows us to naturally encode a very interesting class of cryptographic protocols, namely cryptographic

protocols with *single blind copying*, introduced in [7]. As we saw in Section 2, each protocol step involves copying certain unknown parts of the received message into the sent message. In the example we discussed, an agent always copies only one unknown (the nonce created by the other participant) into the sent message. This is precisely the restriction imposed by the restriction of single blind copying. Although in our example, this unknown occurs exactly once in the received and sent messages, we may allow more than one occurrences thereof. As a consequence of this restriction, the clauses required for modeling the protocol steps are one-variable clauses, as we can see for our example protocol. The other clauses which are independent of the protocol steps, e.g. encryption and decryption abilities of the adversary, are modeled using general flat clauses of Section 4 or their restrictions introduced in the next section.

The modeling described in Section 2 of the Needham-Schroeder protocol is based on abstraction of an infinite set of nonces by a single constant. This is secure in the sense that no attacks are missed. However this may sometimes lead to too many false attacks. Hence we could adopt a less severe abstraction in which a nonce is not necessarily a constant, but a function of some previous messages exchanged in the protocol. Hence nonces in two distinct sessions may still be the same. This kind of abstraction leads to the following clauses for the steps of the Needham-Schroeder protocol. Note that the second nonce n^2_{uv} is now a function of the first nonce n^1_{uv} which is a constant. These clauses are still one-variable clauses, although they do not belong to the class $\mathcal{H}_1$.

$$\begin{array}{ll} \mathsf{known}(\{\langle u, n^1_{uv}\rangle\}_{k_v}) & \\ \mathsf{known}(\{\langle x, n^2_{uv}(x)\rangle\}_{k_u}) & \Leftarrow \mathsf{known}(\{\langle a, x\rangle\}_{k_v}) \\ \mathsf{known}(\{x\}_{k_v}) & \Leftarrow \mathsf{known}(\{\langle n^1_{uv}, x\rangle\}_{k_u}) \end{array}$$

We restrict ourselves to only unary predicates. For one-variable clauses, this causes no loss of generality as we can encode atoms $P(t_1, \ldots, t_n)$ as $P(c(t_1, \ldots, t_n))$ by choosing a fresh symbol c. *Normal* one-variable clauses are one-variable clauses of the forms

1. $P(t) \Leftarrow Q(x)$ where t is non-ground and non-trivial.
2. $P(t)$ where t is ground.
3. $P(x)$.

To restrict the form of unifiers of one-variable terms required during normalization, we decompose these terms, similar to decomposing a string into symbols. If t is a one-variable term (i.e. one containing at most one variable) which is non-ground and s is any other term, then $t[s]$ denotes the effect of replacing the variable in t by s. This notation is extended to sets of terms in the expected manner. A non-ground one-variable term $t[x]$ is called *reduced* if it is not of the form $u[v[x]]$ for any non-ground non-trivial one-variable terms $u[x]$ and $v[x]$. The term $f(g(x), h(g(x)))$ for example is not reduced because it can be written as $f(x, h(x))[g(x)]$. The term $f'(x, g(x), a)$ is reduced. Unifying it with the reduced term $f'(h(y), g(h(a)), y)$ produces ground unifier $\{x \mapsto h(y)[a], y \mapsto a\}$ and both $h(y)$ and a are strict subterms of the given terms. Indeed we find:

Lemma 1. *Let $s[x]$ and $t[y]$ be reduced, non-ground and non-trivial terms where $x \neq y$ and $s[x] \neq t[x]$. If s and t have a unifier σ, then $x\sigma, y\sigma \in U[V]$ where U is the set of non-ground (possibly trivial) strict subterms of s and t, and V is the set of ground strict subterms of s and t.*

In case both terms (even if not reduced) have the same variable:

Lemma 2. *Let σ be a unifier of two non-trivial, non-ground and distinct one-variable terms $s[x]$ and $t[x]$. Then $x\sigma$ is a ground strict subterm of s or of t.*

The intuition behind Lemma 2 is that the variable x could not be mapped to a non-ground non-trivial term because that term itself would contain x. Similarly the intuition behind Lemma 1 is that if x (resp. y) is not immediately mapped to a ground term, then x (resp. y) is mapped to a non-ground subterm of $t[y]$ (resp. $s[x]$) and then y (resp. x) could only be mapped to a ground term.

Hence in the following one-variable clauses are simplified to involve only reduced terms.

Lemma 3. *Any non-ground one-variable term $t[x]$ can be uniquely written as $t[x] = t_1[t_2[\ldots[t_n[x]]\ldots]]$ where $n \geq 0$ and each $t_i[x]$ is non-trivial, non-ground and reduced. This decomposition can be computed in time polynomial in the size of t.*

Proof: We represent $t[x]$ as a DAG by doing maximal sharing of subterms. If $t[x] = x$ then the result is trivial. Otherwise let N be the position in this graph, other than the root node, closest to the root such that N lies on every path from the root to the node corresponding to the subterm x. Let t' be the strict subterm of t at position N and let t_1 be the term obtained from t by replacing the sub-DAG at N by x. Then $t = t_1[t']$ and t_1 is reduced. We then recursively decompose t'.

Uniqueness of decomposition follows from Lemma 1. □

Above and elsewhere, if $n = 0$ then $t_1[t_2[\ldots[t_n[x]]\ldots]]$ denotes x. Now if there is an atom $P(t[x])$ occurring in some clause, with $t[x]$ being non-ground, and if $t[x] = t_1[\ldots[t_n[x]]\ldots]$ where each t_i is non-trivial and reduced, then we create fresh predicates $Pt_1\ldots t_i$ for $1 \leq i \leq n-1$ and replace this atom by the atom $Pt_1\ldots t_{n-1}(t_n[x])$. Also we add clauses $Pt_1\ldots t_i(t_{i+1}[x]) \Leftarrow Pt_1\ldots t_{i+1}(x)$ and $Pt_1\ldots t_{i+1}(x) \Leftarrow Pt_1\ldots t_i(t_{i+1}[x])$ for $0 \leq i \leq n-2$ to our clause set.

Hence now we assume that non-ground atoms in clauses involve only reduced terms or trivial terms as arguments of predicates. Let Ng be the set of these terms, w.l.o.g. containing also the trivial term. Let Ngs be the set of non-ground subterms of terms in Ng. Let G be the ground terms occurring in the clauses. During normalization we are only going to produce atoms $P(t)$ with $t \in \mathsf{Ng} \cup \mathsf{Ng}[\mathsf{Ngs}[\mathsf{G}]]$. As before we consider sets of predicates to represent intersections of individual predicates. If a clause has a non-ground head and a ground body then we add the atom $P(x)$ in the body, where $P(x)$ is a fresh predicate. We add the clause $P(x)$ to the set to say that P accepts all terms. Here are the possible normalization steps.

- We have clause $h \Leftarrow \mathcal{B} \wedge S(t[x])$ and normal clause $S(s[y]) \Leftarrow S'(y)$ where $s, t \in \mathsf{Ng}$ are non-trivial. The normalization step produces $h\sigma \Leftarrow \mathcal{B}\sigma \wedge S'(y\sigma)$ where σ unifies s and t. If $s[x] = t[x]$ then σ is a renaming. Otherwise $x\sigma, y\sigma \in \mathsf{Ngs}[\mathsf{G}]$ by Lemma 1. Further $\mathsf{Ngs}[\mathsf{G}] \subseteq \mathsf{Ng}[\mathsf{Ngs}[\mathsf{G}]]$. Ground atoms $S''(g)$ are removed from the body by checking that g is accepted at S'' using the normal clauses only.
- We have clauses $h \Leftarrow \mathcal{B} \wedge S(t[y])$ and $S(s)$ where $t \in \mathsf{Ng}$ is non-trivial and $s \in \mathsf{Ng}[\mathsf{Ngs}[\mathsf{G}]]$. The normalization step produces $h\sigma \Leftarrow \mathcal{B}\sigma$ where σ unifies t and s.

The ground atoms from the body are removed as before. If $s \in \mathsf{Ngs}[\mathsf{G}]$ then the clause is clearly of the right form. Otherwise $s = u[g]$ where $u \in \mathsf{Ng}$ is non-trivial, and $g \in \mathsf{Ngs}[\mathsf{G}]$. Hence σ is also a unifier of non-trivial reduced terms t and u. If $s[x] = t[x]$ then the result is easy. Otherwise by Lemma 1, $x\sigma \in \mathsf{Ngs}[\mathsf{G}]$, and the result is again easy. The last argument is crucial: the unifier is independent of g hence the ground terms in clauses don't grow arbitrarily.

- Clause $S(x) \Leftarrow S'(x)$ and normal clause $S'(t) \Leftarrow \mathcal{B}$ produce $S(t) \Leftarrow \mathcal{B}$ where $\mathcal{B}$ may be possibly empty.
- The case of normalization steps involving clause $P(x)$ is easy.
- Clause $h \Leftarrow \mathcal{B} \wedge S_1(x) \wedge S_2(x)$ produces $h \Leftarrow \mathcal{B} \wedge (S_1 \cup S_2)(x)$.
- Normal clauses $S_1(s) \Leftarrow T_1(x)$ and $S_2(t) \Leftarrow T_2(x)$ produce $(S_1 \cup S_2)(t\sigma) \Leftarrow (T_1 \cup T_2)(x\sigma)$ where σ unifies s and t. Similarly a normal non-ground and a normal ground clause can produce a new clause. The unifications involved are as considered above.

Hence we produce only polynomially many terms during normalization and hence only exponentially many clauses. While [25] uses resolution techniques to decide satisfiability for these clauses, we further show here that the clauses can even be put into normal form.

Theorem 3 ([25]). *A set of one-variable clauses can be normalized in DEXPTIME.*

6 One Variable Clauses and Flat Clauses

We now return to our goal of having both general flat clauses and one-variable clauses, in order to be able to model cryptographic protocols with single blind copying. However instead of the general flat clauses considered before, we consider *flat clauses* which are those general flat clauses in which every non-trivial atom contains all variables of the clause. The clause

$$P(f(x, y)) \Leftarrow Q(g(y, z))$$

is a general flat clause but not a flat clause. The set of variables in one atom is $\{x, y\}$ and in the other is $\{y, z\}$. The following clause is a flat clause.

$$P_1(f(x, y, x, z)) \Leftarrow P_2(g(y, z, x)) \wedge P_3(h(y, y, x, z, z)) \wedge P_4(x)$$

This class of flat clauses is what is considered in [25] and also suffices for modeling cryptographic protocols with single blind copying. Recall that the one-variable clauses model the protocol steps involving single blind copying, as explained in Section 5 and the flat clauses model the additional capabilities of the adversary to perform encryption, decryption etc. This restriction simplifies the interaction between flat and one-variable clauses. *Normal* clauses are now defined to be clauses which are either normal one-variable clauses or normal flat clauses. Note that the definition of normal flat clauses is same as in the case of general flat clauses. Our goal in this case is to obtain a set of normal flat clauses and normal one-variable clauses.

We will now have three kinds of normalization steps. Normalization steps between two flat clauses or between two one-variable clauses are as in Sections 4 and 5. The

third kind of normalization step is between a one-variable clause and a flat clause, and this always produces a one-variable clause. As a typical example a normalization step between the following two clauses

$$\begin{aligned} C_1 = \quad & P_1(f(x,y)) \Leftarrow P_2(g(y,x)) \wedge P_3(x) \\ C_2 = P_2(g(x,h(h(x)))) & \Leftarrow P_4(x) \wedge P_5(x) \end{aligned}$$

produces the clause

$$P_1(f(h(h(x)),x)) \Leftarrow P_3(h(h(x))) \wedge P_4(x) \wedge P_5(x)$$

Further the term $f(h(h(x)),x)$ is reduced. However $h(h(x))$ is not reduced and hence this term needs to be further decomposed. We replace this clause by the following clauses where $P_3h(x)$ is a fresh predicate. In general we require new predicates corresponding to original predicates and a sequence of one-variable terms. The variables occurring in the terms in these sequences are not important, and these new predicates are identified upto replacements of these variables by other variables.

$$\begin{aligned} P_1(f(h(h(x)),x)) &\Leftarrow P_3h(x)(h(x)) \wedge P_4(x) \wedge P_5(x) \\ P_3h(x)(x) &\Leftarrow P_3(h(x)) \\ P_3(h(x)) &\Leftarrow P_3h(x)(x) \end{aligned}$$

Let Ngs be the set of non-ground (subterms of) terms in the one-variable clauses, $\mathsf{Ngr} = \{\{s[\mathbf{x}_{r+1}] \mid s \text{ is non-ground and reduced,and for some } t, s[t] \in \mathsf{Ngs}\}$. Define $\mathsf{Ngrr} = \{s_1[\ldots[s_m]\ldots] \mid s_1[\ldots[s_n]\ldots] \in \mathsf{Ngs}, m \leq n, \text{ and each } s_i \text{ is non-trivial and reduced}\}$. The reason we need these new sets is that during resolution we create subterms of reduced terms which need to be then further decomposed, as in the above example. We further define the set $\mathsf{Ngr}_1 = \{f(s_1,\ldots,s_n) \mid g(t_1,\ldots,t_m) \in \mathsf{Ngr}, \{s_1,\ldots,s_n\} = \{t_1,\ldots,t_n\}\}$. These terms are reduced but are exponentially many. These are produced as instances of the non-trivial terms in the flat clauses as in the above example. However, the number of such terms in a clause is linear in the initial clause size. Further a normalization step of this clause with a normal flat clause can only produce strict subterms of these terms (in Ngrr) which can then be further decomposed. Readers may consult [25] for precise details about the form of clauses produced during normalization.

The other important observation is that we need only polynomially many fresh predicates for performing decompositions, because the trivial atoms in the flat clause C_2 above can never involve the auxiliary predicates. This is because when we introduce auxiliary predicate as above, the clause becomes a one-variable clause. No further steps can transform this atom into a trivial atom in a non-trivial flat clause.

Theorem 4 ([25]). *A set of flat and one-variable clauses can be normalized in DEXPTIME.*

While only the satisfiability problem is considered in [25] (for Horn and non-Horn clauses), we have here presented a simpler procedure which further produces a set of normal clauses in the Horn case.

Example 1. Consider the set $S = \{C_1, \ldots, C_5\}$ of clauses where

$$\begin{array}{ll} C_1 = & P(a) \\ C_2 = & Q(a) \\ C_3 = & P(f(g(\mathbf{x}_1, a), g(a, \mathbf{x}_1), a)) \Leftarrow P(\mathbf{x}_1) \\ C_4 = & P(f(g(\mathbf{x}_1, a), g(a, \mathbf{x}_1), b)) \Leftarrow P(\mathbf{x}_1) \\ C_5 = & R(\mathbf{x}_1) \Leftarrow P(f(\mathbf{x}_1, \mathbf{x}_1, \mathbf{x}_2)) \wedge Q(\mathbf{x}_2) \end{array}$$

We first get the following normal clauses.

$$\begin{array}{ll} C_1' = & \{P\}(a) \\ C_2' = & \{Q\}(a) \\ C_3' = & \{P\}(f(g(\mathbf{x}_1, a), g(a, \mathbf{x}_1), a)) \Leftarrow \{P\}(\mathbf{x}_1) \\ C_4' = & \{P\}(f(g(\mathbf{x}_1, a), g(a, \mathbf{x}_1), b)) \Leftarrow \{P\}(\mathbf{x}_1) \end{array}$$

The clause

$$C_5' = \{R\}(\mathbf{x}_1) \Leftarrow \{P\}(f(\mathbf{x}_1, \mathbf{x}_1, \mathbf{x}_2)) \wedge \{Q\}(\mathbf{x}_2)$$

is not normal. A normalization step with C_3' gives the clause

$$\{R\}(g(a, a)) \Leftarrow \{P\}(a) \wedge \{Q\}(a)$$

As a is accepted at $\{P\}$ and $\{Q\}$ using the normal clauses C_1' and C_2', hence we get a new normal clause

$$C_6 = \{R\}(g(a, a))$$

Resolving C_5' with C_4' gives

$$\{R\}(g(a, a)) \Leftarrow \{P\}(a) \wedge \{Q\}(b)$$

But b is not accepted at $\{Q\}$ using the normal clauses hence this clause is rejected. Finally C_1' and C_2' also give the normal clause

$$C_7 = \{P, Q\}(a)$$

The resulting set of normal clauses is $\{C_1', \ldots, C_4', C_6, C_7\}$.

For protocols this gives us the following complexity of the verification problem, which is also optimal [25].

Theorem 5 ([25]). *Secrecy for cryptographic protocols with single blind copying can be decided in DEXPTIME.*

7 k-Variable Clauses and Flat Clauses

Now we consider a further generalization by allowing not just one-variable clauses but also k-variable clauses, i.e. clauses having at most k variables. Our goal is to be able to model protocols in which more than one unknown data may be blindly copied. We are interested in the case where k is small, hence we assume it is bounded by

some constant. Further, to obtain decidability, we impose restrictions on the occurrences of variables. A term or literal is called *covering* if every non-ground functional term occurring in it contains all variables of the term or literal. A clause is called *covering* if every literal in it is covering and every non-ground literal involving a n-ary predicate for $n \geq 2$ contains all variables of the clause. We are interested in covering k-variable clauses together with flat clauses. Note that every flat clause can also be considered as covering k-variable clauses for a suitable k, however we allow the flat clauses to have arbitrary many variables. Further as in the one-variable case, we could restrict covering k-variable clauses to have only unary-predicates. However during normalization, we are going to introduce new predicates of arity at most k. Hence w.l.o.g. we assume that our k-variable clauses always involve predicates of arity at most k. Flat clauses of course involve only unary predicates. Our definitions are inspired by that of the class $\mathcal{S}^+$ [11]. Flat clauses belong to the class $\mathcal{S}^+$. Our definition of covering k-variable clauses is essentially the same as that of $\mathcal{S}^+$ clauses, except for the fact that we have restricted the number of variables, and we allow arbitrary ground subterms unlike in the case of $\mathcal{S}^+$.

As example for protocols modeled by such clauses, consider the Yahalom protocol [1] below. Participants A and B use a trusted server S to compute a common key K_{AB}. N_A and N_B are nonces chosen by A and B respectively. K_{AS} and K_{BS} are long term shared keys between A and S and between B and S respectively.

$$\begin{array}{l} A \longrightarrow B : A, N_A \\ B \longrightarrow S : B, \{A, N_A, N_B\}_{K_{BS}} \\ S \longrightarrow A : \{B, K_{AB}, N_A, N_B\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}} \\ A \longrightarrow B : \{A, K_{AB}\}_{K_{BS}}, \{N_B\}_{K_{AB}} \end{array}$$

To model the protocol, as before we use constants n^1_{uv} and n^2_{uv} to represent the two respective nonces chosen by u and v for sessions among themselves. For every pair (u, v) of agents, the clauses corresponding to the protocol steps as as follows. For the first step we have the clause

$$\mathsf{known}(\langle u, n^1_{uv}\rangle)$$

For the second step we have clauses, using the fact that the adversary knows a pair of message iff he knows the individual messages.

$$\begin{array}{ll} \mathsf{known}(v) & \Leftarrow \mathsf{known}(\langle u, x\rangle) \\ \mathsf{known}(\{\langle u, x, n^2_{uv}\rangle\}_{k_{bS}}) & \Leftarrow \mathsf{known}(\langle u, x\rangle) \end{array}$$

By similar reasoning we obtain the following clauses for the third step.

$$\begin{array}{ll} \mathsf{known}(\{\langle v, k_{uv}, x, y\rangle\}_{k_{uS}}) & \Leftarrow \mathsf{known}(v), \mathsf{known}(\{\langle u, x, y\rangle\}_{k_{vS}}) \\ \mathsf{known}(\{\langle u, k_{uv}\rangle\}_{k_{vS}}) & \Leftarrow \mathsf{known}(v), \mathsf{known}(\{\langle u, x, y\rangle\}_{k_{vS}}) \end{array}$$

For the fourth step we have the following clause, where the body has only the first component of the message received by u, because the second component is copied without any checks, hence has no impact on the adversary's knowledge.

$$\mathsf{known}(\{x\}_y) \Leftarrow \mathsf{known}(\{\langle v, y, n^1_{uv}, x\rangle\}_{k_{uS}})$$

Here we consider $\langle \_, \_, \_ \rangle$ and $\langle \_, \_, \_, \_ \rangle$ to be 3-ary and 4-ary functions respectively, instead of considering them to be built up by compositions of the binary function $\langle \_, \_ \rangle$. With this choice, the clauses we obtain are covering 2-variable clauses. Further, if we adopt a milder abstraction, as in Section 5 for the Needham-Schroeder protocol, then we have the following covering k-variable clauses. n^2_{uv} is now a function of n^1_{uv} and k_{uv} is a function of both of them.

$$
\begin{array}{ll}
\mathsf{known}(\langle u, n^1_{uv} \rangle) & \\
\mathsf{known}(v) & \Leftarrow \mathsf{known}(\langle u, x \rangle) \\
\mathsf{known}(\{\langle u, x, n^2_{uv}(x) \rangle\}_{k_{vS}}) & \Leftarrow \mathsf{known}(\langle u, x \rangle) \\
\mathsf{known}(\{\langle v, k_{uv}(x, y), x, y \rangle\}_{k_{uS}}) & \Leftarrow \mathsf{known}(v), \mathsf{known}(\{\langle u, x, y \rangle\}_{k_{vS}}) \\
\mathsf{known}(\{\langle u, k_{uv}(x, y) \rangle\}_{k_{vS}}) & \Leftarrow \mathsf{known}(v), \mathsf{known}(\{\langle u, x, y \rangle\}_{k_{vS}}) \\
\mathsf{known}(\{x\}_y) & \Leftarrow \mathsf{known}(\{\langle v, y, n^1_{uv}, x \rangle\}_{k_{uS}})
\end{array}
$$

In case of k-variable clauses, we define normal clauses to be those of the form

1. $P(t_1, \ldots, t_n) \Leftarrow Q(x_1, \ldots, x_m)$ where $x_1, \ldots, x_m$ are exactly the (pairwise distinct) variables in the head, and $(t_1, \ldots, t_n)$ is not a permutation of $(x_1, \ldots, x_m)$.
2. $P(t_1, \ldots, t_n)$.

It is easy to check that given a set of normal clauses of the above form, we can verify in polynomial time whether some ground atom $P(t_1, \ldots, t_n)$ holds (membership test). First we show how to normalize a set of k-variable covering clauses. As in the case of one-variable clauses, we need to rely on decompositions of terms. But as we have more than one variable, it is more convenient to talk of decompositions of substitutions. In this section, we consider substitutions σ to be always over a finite domain $dom(\sigma)$ of variables and $\mathsf{fv}(\sigma)$ denotes the set of free variables of the terms in the range $range(\sigma)$ of σ, also called the variables *occurring* in σ. If X is the domain of σ and $Y \subset X$ then $\sigma|_X$ denotes as usual the restriction of σ to the domain Y. We define a *k-variable* term, literal, or substitution to be one in which at most k variables occur. A substitution is called *covering* if every non-ground functional term occurring in the range contains all variables in the range. A term or literal is called *simple* if it contains only variables or ground terms. A substitution is *simple* if it maps variables to variables and ground terms. A covering k-variable substitution σ is called *fat* if it maps every variable to a non-ground term and $x\sigma = y\sigma$ only when $x = y$. The only substitutions which are both fat and simple are renamings. The substitution $\{x \mapsto a, y \mapsto f(x, y)\}$ is neither fat nor simple. Composition of two fat covering k-variable substitutions is a fat covering k-variable substitution. A non-renaming fat covering k-variable substitution σ is called *reduced* if it cannot be written as composition of two non-renaming fat covering k-variable substitutions. A term t is called *reduced* if the substitution $\{x \mapsto t\}$ is reduced. We will consider tuples $(t_1, \ldots, t_n)$ interchangeably as substitutions $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ for convenience. Hence if substitution σ has a domain of size n and P is a n-ary predicate then $P(\sigma)$ denotes an atom as expected.

The extra problem in the k-variable case is that reduced terms may become non-reduced after application of some simple substitutions which unify two subterms. E.g. the substitution $\{x \mapsto f(x_1, g(x_1, x_2, a), g(x_1, a, x_2))\}$ is reduced. However the instance $\{x \mapsto f(x_1, g(x_1, a, a), g(x_1, a, a))\}$ is not reduced and can be written as

$\{x \mapsto f(x_1, x_2, x_2)\}\{x_1 \mapsto x_1, x_2 \mapsto g(x_1, a, a)\}$. Indeed the only way to unify two distinct k-variable covering terms, which have the same set of free variables, is by mapping variables to variables and to ground subterms of the two terms.

Lemmas 4 and 5 below are generalizations of Lemmas 1 and 2 to the k-variable case. Given two substitutions σ_1 and σ_2 over disjoint domains $\sigma_1 \oplus \sigma_2$ denotes as expected the substitution over the union of the two domains.

Lemma 4. *Consider two non-renaming reduced fat covering k-variable substitutions σ_1 and σ_2, over the same domain, and which are not renamings of each other. Let G be the set of ground subterms of terms in the range of σ_1 and σ_2 and $\mathsf{fv}(\sigma_1) \cap \mathsf{fv}(\sigma_2) = \emptyset$. Let σ be the mgu of σ_1 and σ_2. Then one of the following cases occur.*

- *$\sigma = \sigma_3 \oplus \sigma_4$ where $dom(\sigma_3) = \mathsf{fv}(\sigma_1)$, $dom(\sigma_4) = \mathsf{fv}(\sigma_2)$. σ_3 maps variables in $dom(\sigma_1)$ to variables in $dom(\sigma_1)$ and to terms in G. σ_4 is of the form $\theta\rho\sigma_3$ where θ maps variables in $dom(\sigma_2)$ to variables in $dom(\sigma_2)$ and to terms in G, ρ and is a fat substitution. Further, either θ or σ_3 is non-renaming.*
- *The symmetric case, with roles of σ_1 and σ_2 exchanged.*

Essentially non-renaming unification involves a fat substitution preceded and succeeded by simple substitutions. One of these two simple substitutions has to be non-renaming because of the reducedness condition. For example consider

$$\begin{array}{ll} \sigma_1 = \{x \mapsto f(h(x_1, a, y_1), h(x_1, y_1, a), g(x_1, y_1)), & y \mapsto g(x_1, y_1)\} \\ \sigma_2 = \{x \mapsto f(x_2, x_2, y_2), & y \mapsto y_2\} \end{array}$$

We have $\sigma_3 \oplus \sigma_4$ as the mgu of σ_1 and σ_2 where

$$\begin{array}{l} \sigma_3 = \{x_1 \mapsto x_1, y_1 \mapsto a\} \\ \theta \ = \{x_2 \mapsto x_2, y_2 \mapsto y_2\} \\ \rho \ = \{x_2 \mapsto h(x_1, a, y_1), y_2 \mapsto g(x_1, y_1)\} \\ \sigma_4 = \theta\rho\sigma_3 \end{array}$$

Here the first two arguments of $\sigma_1(x)$ had to be unified which led to y_1 being mapped to a. In case the sets of free variables in the ranges are the same, then we have the following generalization of Lemma 2.

Lemma 5. *Consider two covering k-variable substitutions σ_1 and σ_2 over the same domain. Let G be the set of ground subterms of terms in the range of σ_1 and σ_2 and $\mathsf{fv}(\sigma_1) = \mathsf{fv}(\sigma_2)$. Let σ be the mgu of σ_1 and σ_2. If σ_1 and σ_2 are not renamings then σ maps variables to variables and to terms in G.*

For example the mgu of the substitutions $\{x \mapsto f(x, g(x, y, z), h(a))\}$ and $\{x \mapsto f(y, g(x, y, z), z)\}$ is $\{x \mapsto x, y \mapsto x, z \mapsto h(a)\}$. The point is that a variable x cannot be mapped to a non-ground functional term since that term itself must contain the variable x. Lemma 3 is generalized as follows. We decompose covering substitutions as $\sigma = \theta\rho_1 \ldots \rho_n$ where θ is simple and ρ_i are fat and reduced. Intuitively θ tells us exactly which variables should be made equal to each other, and which variables should be made ground. The uniqueness of the choice of the ρ_i follows from Lemma 4.

Lemma 6. *1. Every non-ground covering k-variable substitution can be uniquely written as $\theta\sigma$ where θ is simple and σ is a fat covering k-variable substitution.*
2. Every fat covering k-variable substitution σ can be uniquely written as $\sigma = \rho_1 \dots \rho_n$ where $n \geq 0$ and each ρ_i is a reduced fat covering k-variable substitution.

For example the substitution $\{x_1 \mapsto f(x,y), x_2 \mapsto f(x,y), x_3 \mapsto f(y,x), x_4 \mapsto x, x_5 \mapsto h(a)\}$ can be written as $\theta\sigma$ where $\theta = \{x_1 \mapsto y_1, x_2 \mapsto y_1, x_3 \mapsto y_2, x_4 \mapsto x, x_5 \mapsto h(a)\}$ is simple and $\sigma = \{y_1 \mapsto f(x,y), y_2 \mapsto f(y,x), x \mapsto x\}$ is fat covering. The fat covering substitution $\{x_1 \mapsto f(h(x), g(y)), x_2 \mapsto f(g(y), h(x)), x_3 \mapsto h(x)\}$ can be written as $\rho_1\rho_2$ where $\rho_1 = \{x_1 \mapsto f(y_1, y_2), x_2 \mapsto f(y_2, y_1), x_3 \mapsto y_1\}$ and $\rho_2 = \{y_1 \mapsto h(x), y_2 \mapsto g(y)\}$ are reduced fat covering. If $k = 1$ then the substitution $\{x_1 \mapsto f(g(x), h(x))\}$ is reduced. But if $k \geq 2$ then we can decompose it as $\{x_1 \mapsto f(y_1, y_2)\}$ and $\{y_1 \mapsto g(x), y_2 \mapsto h(x)\}$.

Hence given a set $\mathbb{S}$ of k-variable covering clauses, let G be the set of all ground terms occurring in $\mathbb{S}$. We add to $\mathbb{S}$ all possible instances of clauses by mapping variables to variables and terms from G. This means that now we never need to consider instances of these clauses which unify two distinct subterms occurring in a term or which unify some non-ground term in a clause with a term in G.

Next we decompose the terms occurring in the clauses, as in the one-variable case. An atom of the form $P(\theta\rho_1 \dots \rho_n)$, with $n \geq 1$, in a clause is replaced by the atom $P_{\theta,\rho_1,\dots,\rho_n}(\boldsymbol{x})$ and we add clauses

$$\begin{aligned}
P_\theta(\boldsymbol{x}) &\Leftarrow P(\theta)\\
P(\theta) &\Leftarrow P_\theta(\boldsymbol{x})\\
P_{\theta,\rho_1}(\boldsymbol{x}) &\Leftarrow P_\theta(\rho_1)\\
P_\theta(\rho_1) &\Leftarrow P_{\theta,\rho_1}(\boldsymbol{x})\\
&\dots\\
P_{\theta,\rho_1,\dots,\rho_n}(\boldsymbol{x}) &\Leftarrow P_{\theta,\rho_1,\dots,\rho_{n-1}}(\rho_n)\\
P_{\theta,\rho_1,\dots,\rho_{n-1}}(\rho_n) &\Leftarrow P_{\theta,\rho_1,\dots,\rho_n}(\boldsymbol{x})
\end{aligned}$$

where in each clause, $\boldsymbol{x}$ represents a sequence of mutually distinct variables of appropriate length, $P_{\theta,\rho_1,\dots,\rho_i}$ are fresh predicates, θ is a simple covering k-variable substitution and ρ_i are non-renaming reduced fat covering k-variable substitutions. In case θ is a renaming then $P\theta$ is the same as P and the first two clauses are omitted. If $n = 0$ then the atom is replaced by $P\theta(\boldsymbol{x})$. This means that now predicates are only applied to simple or reduced fat substitutions. As an example the literal $P(f(h(x), g(y)), f(g(y), h(x)), h(x))$ is written as $P(\sigma)$ where $\sigma = \{x_1 \mapsto f(h(x), g(y)), x_2 \mapsto f(g(y), h(x)), x_3 \mapsto g(x)\}$. σ can be written as $\rho_1\rho_2$ where $\rho_1 = \{x_1 \mapsto f(y_1, y_2), x_2 \mapsto f(y_2, y_1), x_3 \mapsto y_1\}$ and $\rho_2 = \{y_1 \mapsto h(x), y_2 \mapsto g(y)\}$. Hence this literal can be replaced by the literal $P\rho_1\rho_2(x, y)$ and additionally we have the following clauses. Further if in the original clause x and y never needed to be unified then in the new clauses also x and y never need to be unified, and y_1 and y_2 never need to be unified.

$$\begin{aligned}
P\rho_1(y_1, y_2) &\Leftarrow P(f(y_1, y_2), f(y_2, y_1), y_1)\\
P(f(y_1, y_2), f(y_2, y_1), y_1) &\Leftarrow P\rho_1(y_1, y_2)\\
P\rho_1\rho_2(x, y) &\Leftarrow P\rho_1(h(x), g(y))\\
P\rho_1(h(x), g(y)) &\Leftarrow P\rho_1\rho_2(x, y)
\end{aligned}$$

Let $\mathbb{S}_1$ be the new set of clauses. Even after these transformations, we never need to consider instances of our clauses which unify two distinct subterms occurring in a term or which unify some non-ground term in a clause with a term in G. This property is going to be preserved during all stages of our normalization procedure. Further we preserve the property that at most one atom in a clause has a predicate applied to a non-renaming substitution. Let Ng be the set of non-ground terms occurring in $\mathbb{S}_1$, and Ngs the set of their subterms, as well as non-ground subterms of terms occurring in $\mathbb{S}$ (not $\mathbb{S}_1$). Let F be the set of fat covering k-variable substitutions with domain of size at most k and range containing (renamings of) terms from Ngs. Let S be the set of simple k-variable substitutions with domain of size at most k and the ground terms in the range being only from G. Compositions of sets of substitutions are defined as expected. During normalization, we are only going to produce atoms in which the predicate has an argument of one of the following forms.

- $\theta \in \mathsf{S}$.
- some non-renaming reduced $\rho \in \mathsf{F}$.
- $\theta_1\rho_1\rho_2\theta_2$, where $\theta_1, \theta_2 \in \mathsf{S}$, θ_2 is ground, $\rho_1, \rho_2 \in \mathsf{F}$ and ρ_1 is non-renaming and reduced.

The normalization procedure now consists of the following kinds of steps, quite similar to the one-variable case. Because of our assumptions about the kinds of instantiations that need to be made of clauses, we are going to avoid unnecessary unifications between the atoms involved. Further the assumptions will continue to hold after each normalization step. We further maintain the invariant that every clause has at most two literals, one of which is a renaming. This is true of the auxiliary clauses produced above. The clauses produced by replacing original clauses have only renamings as arguments in literals. To them we apply the following transformation. For every n-ary predicate and permutation π over n variables, we introduce predicate $P\pi$ which is supposed to accept tuples σ such that $\sigma\pi$ is accepted at P. We further introduce n-ary predicates $\{P_1, \ldots, P_i\}$, where P_i are n-ary predicates, with the usual meaning. Given n-ary predicate S and unary predicates $S_1, \ldots, S_n$ we introduce predicate $S[S_1, \ldots, S_n]$ which accepts tuples $(x_1, \ldots, x_n)$ accepted at S such that x_i is accepted at S_i. Given a permutation π, $S[S_1, \ldots, S_n]\pi$ is defined to be a state of the same form as expected. $S[S_1, \ldots, S_n] \cup T[T_1, \ldots, T_n]$ is defined to be $(S \cup T)[S_1 \cup T_1, \ldots, S_n \cup T_n]$. $S[\emptyset, \ldots, \emptyset]$ is same as S. $\emptyset[\emptyset, \ldots, \emptyset, S_i, \emptyset, \ldots, \emptyset](x_1, \ldots, x_n)$ is same as $S_i(x_i)$. Then a conjunction $S(x_1, \ldots, x_n) \wedge T(x_1, \ldots, x_n)$ in the body is replaced by $(S \cup T)(x_1, \ldots, x_n)$.

- We have a non-normal clause $C_1 = h \Leftarrow S(\sigma_1)$ and a normal clause $C_2 = S(\sigma_2) \Leftarrow \mathcal{B}$ and the normalization step produces $C = h\sigma \Leftarrow \mathcal{B}\sigma$ where σ is mgu of σ_1 and σ_2. $\mathcal{B}$ has at most one atom. The following cases are possible. In our case analysis below, we frequently need to forbid steps where two variables need to be made equal or where a variable needs to be instantiated to a term in G. We will do this without stating the reason explicitly.
  - σ_1 is a renaming. We consider this step only if the substitution occurring as argument in the head is also a renaming since we have assumed C_1 to be

non-normal. Then C is trivially of the required form. In the remaining subcases below, we assume that σ_1 is not a renaming. Hence the head of C_1 must have a renaming as argument.

- $\sigma_1 \in \mathsf{S}$ is not a renaming. If $\sigma_2 \in \mathsf{F}$ then this step is not performed. If $\sigma_2 \in \mathsf{S}$ then this step is performed only if σ_2 is a renaming of σ_1, and then C is of the required form. If σ_2 is of the third form above then C is a ground clause with literals of the required form. Any ground atom from the body is removed by a membership test on the normal clauses.
- $\sigma_1 \in \mathsf{F}$ is not a renaming and σ_2 is a renaming. Then $\mathcal{B}_2$ is empty and C is trivially of the required form.
- $\sigma_1 \in \mathsf{F}$ is not a renaming and $\sigma_2 \in \mathsf{S}$ is not a renaming. Then then this step is not possible.
- $\sigma_1 \in \mathsf{F}$ is not a renaming and $\sigma_2 \in \mathsf{F}$ is not a renaming. If σ_2 is a renaming of σ_1 then C is of the right form. Otherwise this normalization step is not considered because of the substitutions involved according to Lemma 4.
- $\sigma_1 \in \mathsf{F}$ is not a renaming and σ_2 is of the form $\theta_1\rho_1\rho_2\theta_2$ where $\theta_1, \theta_2 \in \mathsf{S}$, θ_2 is ground, $\rho_1, \rho_2 \in \mathsf{F}$ and ρ_1 is non-renaming and reduced. θ_1 must be a renaming for this normalization step to be allowed. Hence σ is also a unifier of σ_1 and ρ_1. By Lemma 4, σ is of the form $\rho_3\theta_3$ where $\theta_3 \in \mathsf{S}$ is ground and $\rho_3 \in \mathsf{F}$. Hence the resulting clause is a ground clause of the right form.

– Two normal clauses $S_1(\sigma_1) \Leftarrow \mathcal{B}_1$ and $S_2(\sigma_2) \Leftarrow \mathcal{B}_2$, where S_1 and S_2 have the same arity, produces a clause $(S_1 \cup S_2)(\sigma_1\sigma) \Leftarrow \mathcal{B}_1\sigma \wedge \mathcal{B}_2\sigma$ where σ unifies σ_1 and σ_2. The unifications involved are as above. A possible ground literal from the body is removed as before. A possible conjunction of two literals (with renamings as arguments) in the body is replaced by a single literal as before.
– Normal clause $S[S_1, \ldots, S_n](t_1, \ldots, t_n) \Leftarrow \mathcal{B}$ produces normal clause $S[S_1, \ldots, S_{i-1}, S_i \cup T, S_{i+1}, \ldots, S_n](t_1, \ldots, t_n) \Leftarrow \mathcal{B} \wedge T(t_i)$ if t_i is a variable. The conjunction in the body is replaced by a single literal as before.
– Given normal clauses $S[S_1, \ldots, S_n](t_1, \ldots, t_n) \Leftarrow \mathcal{B}_1$ and $T(t) \Leftarrow \mathcal{B}_2$ (t cannot be a variable) we consider the mgu σ of t_i and t. We generate the clause $S[S_1, \ldots, S_{i-1}, S_i \cup T, S_{i+1}, \ldots, S_n](t_1, \ldots, t_n)\sigma \Leftarrow \mathcal{B}_1\sigma \wedge \mathcal{B}_2\sigma$. Ground literals from body are again removed by membership tests.
– Normal clause $S(\sigma) \Leftarrow \mathcal{B}$ produces clause $S\pi(\sigma\pi^{-1}) \Leftarrow \mathcal{B}$ where π^{-1} is the inverse of the permutation π.

In other words we have polynomially many possible tuples occurring as arguments of predicates and consequently exponentially many clauses.

Theorem 6. *For a fixed k, a set of covering k-variable clauses can be normalized in DEXPTIME.*

For practical implementations, the systematic instantiations and decompositions could be wasteful. Hence it is better to do them as required. Firstly it is only necessary to decompose the arguments in heads but not in the body. Secondly the instantiations

followed by decomposition should be done before a normalization step as needed, and not in advance. This avoids unnecessary instantiations. Hence given clauses

$$\begin{aligned} h &\Leftarrow P(g(h(x)), g(h(x)))) \\ P(g(x), g(y)) &\Leftarrow Q(x, y) \end{aligned}$$

we apply the substitution $\{x \mapsto x, y \mapsto x\}$ on the second clause and the new clause can then be decomposed to produce the following clauses

$$\begin{aligned} P'(x) &\Leftarrow Q(x, x) \\ P(x, x) &\Leftarrow P''(x) \\ P''(x) &\Leftarrow P(x, x) \\ P''(h(x)) &\Leftarrow P'(x) \\ P'(x) &\Leftarrow P''(h(x)) \end{aligned}$$

Hence a normalization step then produces the new clause

$$h \Leftarrow P''(g(h(x)))$$

When we further allow flat clauses together with k-variable clauses, then the situation is again analogous to the case of one-variable clauses with flat clauses. Normalization steps between a covering k-variable clause and a flat clause produces a covering k-variable clause, which may again need to be decomposed.

Theorem 7. *For a fixed k, a set of covering k-variable clauses and flat clauses can be normalized in DEXPTIME.*

As we have considered k to be a constant, this upper bound does not apply to the class $\mathcal{S}^+$. However letting k be a variable in our algorithm still allows us to show:

Theorem 8. *Satisfiability for the class $\mathcal{S}^+$ can be decided in double exponential time in the Horn case.*

As far as we know no upper bound was previously known for this class. DEXPTIME lower bound for this class is obvious, and tightening the complexity bounds further remains to be done.

8 Conclusion

We have considered several general classes of Horn clauses. For each of them, we provided a normalization procedure which runs in exponential time but practically may be much faster. In particular, our methods can be used to decide satisfiability for these classes. Moreover, these classes provide flexible tools for modeling and certifying secrecy of protocols.

Beyond simplifying the methods from [25], we also generalized the class of flat and one-variable clauses to allow (restricted) k-variable clauses. For fixed small k, normalization and thus satisfiability still is in DEXPTIME. For unbounded k, we have provided a new double exponential time upper bound, which thus also holds for the full Horn fragment of the class $\mathcal{S}^+$. It remains as a challenging problem whether this upper bound can be significantly improved.

References

1. Spore: Security protocol open repository. Available at `http://www.lsv.ens-cachan.fr/spore/`.
2. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society Press, Cape Breton, Nouvelle-Écosse, Canada, 2001.
3. B. Blanchet. Security protocols: From linear to classical logic by abstract interpretation. *Information Processing Letters*, 95(5):473–479, 2005.
4. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. *Theoretical Computer Science*, 333(1-2):67–90, 2005.
5. H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1):143–214, 2005.
6. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata`, 1997.
7. H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In R. Nieuwenhuis, editor, *14th International Conference on Rewriting Techniques and Applications (RTA'03)*, volume 2706 of *LNCS*, pages 148–164, Valencia, Spain, June 2003. Springer-Verlag.
8. H. Comon-Lundh and V. Cortier. Security properties: Two agents are sufficient. In *12th European Symposium on Programming (ESOP'03)*, volume 2618 of *LNCS*, pages 99–113, Warsaw, Poland, Apr. 2003. Springer-Verlag.
9. V. Cortier. *Vérification Automatique des Protocoles Cryptographiques*. PhD thesis, ENS Cachan, France, 2003.
10. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.
11. C. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. *Resolution Decision Procedures*, chapter 25, pages 1791–1849. Volume II of Robinson and Voronkov [23], 2001.
12. T. Frühwirth, E. Shapiro, M. Y. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *6th Annual IEEE Symposium on Logic in Computer Science (LICS'91)*, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.
13. J. Goubault-Larrecq. Une fois qu'on n'a pas trouvé de preuve, comment le faire comprendre à un assistant de preuve? In V. Ménissier-Morain, editor, *Actes des 12èmes Journées Francophones des Langages Applicatifs (JFLA'04)*. INRIA, collection didactique, 2004.
14. J. Goubault-Larrecq. Deciding $\mathcal{H}_1$ by resolution. *Information Processing Letters*, 95(3):401–408, 2005.
15. J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real C code. In R. Cousot, editor, *6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *LNCS*, pages 363–379. Springer-Verlag, 2005.
16. J. Goubault-Larrecq, M. Roger, and K. N. Verma. Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 64(2):219–251, Aug. 2005.
17. G. Lowe. An attack on the Needham-Schroeder public-key protocol. *Information Processing Letters*, 56(3):131–133, 1995.
18. D. Monniaux. Abstracting cryptographic protocols with tree automata. In A. Cortesi and G. Filé, editors, *6th International Static Analysis Symposium (SAS'99)*, volume 1694 of *LNCS*, pages 149–163, Venice, Italy, September 1999. Springer-Verlag.
19. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

20. F. Nielson, H. R. Nielson, and H. Seidl. Normalizable Horn clauses, strongly recognizable relations and Spi. In *9th Static Analysis Symposium (SAS'02)*, volume 24477 of *LNCS*, pages 20–35. Springer-Verlag, 2002.
21. R. Ramanujam and S. P. Suresh. A decidable subclass of unbounded security protocols. In *Workshop on Issues in the Theory of Security (WITS'03)*, 2003.
22. R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. In *23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *LNCS*, pages 363–374. Springer-Verlag, 2003.
23. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. North-Holland, 2001.
24. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In P. Pandya and J. Radhakrishnan, editors, *14th IEEE Computer Security Foundations Workshop (CSFW'01)*, Cape Breton, Nova-Scotia, Canada, June 2001. IEEE Computer Society Press.
25. H. Seidl and K. N. Verma. Flat and one-variable clauses: Complexity of verifying cryptographic protocols with single blind copying. In F. B. ad Andrei Voronkov, editor, *11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04)*, volume 3452 of *LNCS*, pages 79–94. Springer-Verlag, 2005.
26. C. Weidenbach. Towards an automatic analysis of security protocols. In H. Ganzinger, editor, *16th International Conference on Automated Deduction (CADE'99)*, number 1632 in LNAI, pages 378–382. Springer-Verlag, 1999.