Towards an Automatic Analysis of Security Protocols in First-Order Logic

Christoph Weidenbach

Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany,
email: weidenb@mpi-sb.mpg.de

Abstract. The Neuman-Stubblebine key exchange protocol is formalized in first-order logic and analyzed by the automated theorem prover SPASS. In addition to the analysis, we develop the necessary theoretical background providing new (un)decidability results for monadic first-order fragments involved in the analysis. The approach is applicable to a variety of security protocols and we identify possible extensions leading to future directions of research.

1 Introduction

The growing importance of the internet causes a growing need for security protocols that protect transactions and communication. It turns out that the design of such protocols is highly error-prone. Therefore, a variety of different methods have been described that analyze security protocols to discover flaws. The topic of this paper is to add a further, new method that is based on automated theorem proving in first-order logic.

In the context of first-order automated theorem proving, Schumann (1997) implemented the well-known BAN logic (Burrows, Abadi & Needham 1990) in first-order logic and then used the automated theorem prover SETHEO to search for proofs in the BAN logic. The BAN logic is a modal belief logic, suitable to express the beliefs of parties in the course of a protocol execution. The logic has been successfully used to analyze authenticity properties of several classical protocols. The BAN logic is not very-well suited for reasoning about secrecy, e.g., possible actions of an intruder. For this purpose finite state (model checking) methods, see, e.g., the overview article by Mitchell (1998), turned out to be successful. Independently from the specific formalization used in such an approach, the protocol is eventually described by a finite model, usually guaranteeing decidability of the investigated properties. The inductive method due to Paulson (1997) uses inductive definitions for actions of the various parties, message sequences etc. as the basis for an analysis. The analysis is supported by the generic, interactive higher-order logic theorem prover Isabelle. Due to the expressiveness of the logic, a detailed modeling of protocols is possible, at the price that explicit induction proofs are usually not automatic.

Our approach tries to combine the benefits of the finite state analysis and the inductive method. The idea is to use fragments of first-order logic that are expressive enough to have infinite, inductive models, but that are still subject to automated theorem proving. For example, our theory is expressive enough to model an intruder that can send all – in general infinitely many (see page 5) – syntactically composable messages. In Section 2 we demonstrate our approach by an analysis of the Neuman & Stubblebine (1993) key exchange protocol. The protocol is translated into first-order monadic Horn fragments. We show that the automated theorem prover SPASS (Weidenbach, Afshordel, Brahm, Cohrs, Engel, Keen, Theobalt & Topic 1999) can be successfully used to automatically prove security properties of the protocol and to detect potential errors of an implementation. The SPASS input files of the analysis are available at

`http://spass.mpi-sb.mpg.de/`

For space limitations, the analysis of the Neuman-Stubblebine protocol presented here leaves out the second part of the protocol that serves subsequent authentication. This part of the protocol can also be successfully analyzed with our techniques, though. In Section 3 we investigate the Horn fragments involved in the analysis and prove that several of these are decidable in general. The paper ends with a discussion of the achieved results and pointers to future research, Section 4.

2 The Neuman-Stubblebine Protocol

The example protocol we want to study is the key-exchange protocol due to Neuman & Stubblebine (1993). The goal of this protocol is to establish a secure key K_{ab} between two principals A and B that already share secure keys K_{at} and K_{bt} with a trusted server T, respectively.

(i) $A \to B : A, N_a$
(ii) $B \to T : B, E_{K_{bt}}(A, N_a, T_b), N_b$
(iii) $T \to A : E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b$
(iv) $A \to B : E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)$

The protocol starts with A sending the clear-text message (i) to B consisting of two components: A's name and a nonce N_a created by A. So "," in the protocol description means message composition. Nonces are fresh numbers (e.g., random numbers) that are used to prevent replay attacks. Having received this message, B sends the three part message (ii) to the server T. The message starts with B's name, an encrypted middle part and ends with a nonce N_b generated by B. Encryption is denoted by an expression $E_{key}(message)$. The message A, N_a, T_b is encrypted by the secure key K_{bt} that B and T share and contains A's name, the nonce N_a and a time span T_b. The span T_b suggests the expiration time for the eventually generated session key between A and B, an aspect of the protocol that we will not study. The server T decrypts the instructions from B using the key K_{bt} and generates a session key K_{ab} for A and B. Then he sends the three part message (iii) to A. Using the secure key K_{at} the server T encrypts B's name, A's initial nonce N_a, the generated session key K_{ab} and the expiration time span T_b. The second part contains A's name, the session key K_{ab} and T_b encrypted with key K_{bt} and the third part is the nonce N_b. The principal A receives the message, she uses her key K_{at} to decrypt the first part, verifies her nonce N_a, stores the session key K_{ab} and then forwards the second part of T's message to B and adds B's nonce N_b encrypted with the session key K_{ab}, message (iv). Now B decrypts the first part of message (iv) with his key K_{bt}, extracts the session key K_{ab} and by decrypting the second part of the message using the session key he ensures the correct identity of A.

In the sequel, we develop a formalization of this protocol in monadic first-order Horn logic. We are particularly interested in a monadic Horn formalization, because Horn clauses provide a nice minimal model semantics and we are able to provide decidability results for a variety of monadic Horn theories. This will be explained in more detail in the next section, Section 3. We introduce the necessary symbols (predicates, functions, constants) in the course of subsequent message formalization. We adhere to the usual first-order notation where the used operators are $\neg$ (negation), $\wedge$ (conjunction), $\supset$ (implication), $\forall$ (universal quantification), $\exists$ (existential quantification). The key idea of the approach is to formalize the set M of messages that are sent during the execution of the protocol. This model is realistic, because typically such a protocol takes place in an asynchronous framework without any globally available clock. The initial setup for principal A together with message (i) is represented by the formulae

(1) $Ak(key(at, t))$
(2) $P(a)$
(3) $M(sent(a, b, pair(a, na))) \wedge Sa(pair(b, na))$

where key, $sent$ and $pair$ are function symbols, a, b, t, na, at are constant symbols and Ak, P and M are predicate symbols. We use the convention to name function symbols, constant symbols and variables by lowercase letters and predicate symbols start with an uppercase letter. In particular, variables always start with one of the letters u–z. Formula (1) expresses that A holds the key at for the server T. Formula (2) defines a to be one of the parties of the protocol. Formula (3) states that A sends message (i) (see page 2) and stores that she sent the message. A term $sent(x, y, z)$ means that message z is sent by x to y. The predicate M holds for all sent messages, P holds for all principals and predicates named $\langle Principal\rangle k$ hold all keys for *Principal*, cf. formula (1). Finally, Sa is A's local store that will eventually be used to verify her nonce in message (iii).

Principal B is only interested in fresh nonces. Hence, the formalization of his initial setup and his action seeing A's message (i), formula (3), is

(4) $Bk(key(bt, t))$
(5) $P(b)$
(6) $Bf(na)$
(7) $\forall xa, xna\, [(M(sent(xa, b, pair(xa, xna))) \wedge Bf(xna)) \supset$
$\quad (Sb(pair(xa, xna)) \wedge$
$\quad M(sent(b, t, triple(b, nb(xna), encr(triple(xa, xna, tb(xna)), bt)))))]$

where we formalized B to react properly on any message having the structure of message (i), without knowing the principal in advance, formula (7). So our formalization of the protocol is not a priori restricted to three parties or exactly one execution. The premise $Bf(xna)$ declares that xa's nonce is fresh to B, for otherwise B stops the protocol, since the premise of the above implication becomes false. The initial nonce of A to be fresh to B, formula (6). Furthermore, B is a principal, formula (5), and holds the secure key bt with t, formula (4). The functions nb and tb in formula (7) compute B's nonce and expiration time, respectively. Assuming freshness, these functions only depend on xa's fresh nonce xna. Therefore, for every fresh nonce, B will generate different random numbers and expiration spans. So far, we only defined A's nonce na to be fresh. At the end of this section we will model an intruder that has infinitely many numbers available that are fresh to B. Finally, B stores that he got the key request from xa in his local store Sb to verify his time stamp in the final message (iv).

On seeing message (ii), generated by the succedent of formula (7), the server T sends message (iii), formula (10):

(8) $Tk(key(at,a)) \wedge Tk(key(bt,b))$
(9) $P(t)$
(10) $\forall xb, xnb, xa, xna, xbet, xbt, xat, xk$

$$[(M(sent(xb,t,triple(xb,xnb,encr(triple(xa,xna,xbet),xbt)))) \wedge Tk(key(xbt,xb)) \wedge Tk(key(xat,xa))) \supset M(sent(t,xa,triple(encr(quadr(xb,xna,kt(xna),xbet),xat), encr(triple(xa,kt(xna),xbet),xbt),xnb)))]$$

Formula (8) declares the respective keys the server holds for A and B. Formula (9) makes the server a principal. First, in formula (10), the server checks whether he owns the secure key xbt for principal xb. Having the key xbt the server can decrypt the third part of message (ii) and checks whether he also has a secure key xat for communication with principal xa. If all this is satisfied, the server generates a session key $kt(xna)$ by applying the key generation function kt to the nonce xna and sends message (iii) to xa. If xna is a fresh nonce, $kt(xna)$ is a fresh key.

Principal A sees the server message, and tries to decrypt the first part of the message using the secure key at she shares with the server, the antecedent of formula (11). If this succeeds, she checks from her store Sa that this part of the message starts with xb and the nonce she initially sent to xb. Then A forwards the second part xm to xb and encrypts, using the new session key xk contained in her encrypted part, xb's nonce xnb and sends it to xb. In addition, A now owns the session key xk for communication with xb.

(11) $\forall xnb, xbet, xk, xm, xb, xna$

$$[(M(sent(t,a,triple(encr(quadr(xb,xna,xk,xbet),at),xm,xnb))) \wedge Sa(pair(xb,xna)))) \supset (M(sent(a,xb,pair(xm,encr(xnb,xk)))) \ \wedge \ Ak(key(xk,xb)))]$$

Finally, formula (12), B decrypts the first part of the message he received from xa, checks whether it contains his expiration time $xbet$ and uses the session key xk to check whether the second part of the message contains his nonce in the context of xa and xna which he stored in Sb. If all this succeeds, B accepts xk as a secure session key for xa.

(12) $\forall xbet, xk, xnb, xa, xna$
$$[(M(sent(xa, b, pair(encr(triple(xa, xk, tb(xna)), bt), encr(nb(xna), xk)))) \wedge Sb(pair(xa, xna))) \supset Bk(key(xk, xa))]$$

This finishes the formalization of the Neuman-Stubblebine protocol and we now start to analyze it using Spass. First, we want to verify that the protocol yields the desired result, a key between A and B. To this end we saturate the formulae (1)–(12). Recall that the complete below analysis is available via ftp, see Section 1.

Fact 1 Spass *(finitely) saturates the formulae (1)–(12) in less than one second.*

All computations with Spass were performed on a Sun Sparc Ultra 10 with a 300MHz processor running Solaris. Spass computes a finite minimal model that consists in addition to the ground atoms already contained in the formalization of the ground atoms

1. $Bk(key(kt(na), a))$
2. $M(sent(a, b, pair(encr(triple(a, kt(na), tb(na)), bt), encr(nb(na), kt(na)))))$
3. $Ak(key(kt(na), b))$
4. $M(sent(t, a, triple(encr(quadr(b, na, kt(na), tb(na)), at), encr(triple(a, kt(na), tb(na)), bt), nb(na))))$
5. $M(sent(b, t, triple(b, nb(na), encr(triple(a, na, tb(na)), bt))))$
6. $M(sent(a, b, pair(a, na)))$
7. $Sb(pair(a, na))$
8. $Sa(pair(b, na))$

In the next section, Section 3, we will explain what saturation means and how this model is in fact automatically computed by Spass. The model contains exactly the messages that A, B and T communicate according to the protocol. Atoms 1 and 3 indicate the established key $kt(na)$ between A and B. This can be automatically extracted.

Fact 2 Spass *proves the conjecture $\exists x\,[Ak(key(x, b)) \wedge Bk(key(x, a))]$ with respect to the saturated theory (Fact 1) in less than one second.*

So, we automatically proved a first important property of the protocol: The protocol terminates and it establishes a key between A and B.

The rest of this section is devoted to an analysis to what extent an intruder can disturb the protocol and how such attacks can be prevented using appropriate implementations for A, B and T. We use the following assumptions for the intruder: First, the intruder can record all sent messages. Second, the intruder cannot break any secure key. In particular, he cannot break the initial keys at and bt. Third, the intruder can send messages and can forge the sender of a message. Fourth, the intruder has no access to the local stores Sa and Sb. The first assumption is formalized by the formula

(13) $\forall xa, xb, xm\,[M(sent(xa, xb, xm)) \supset Im(xm)]$

where $Im(xm)$ means that xm is a message recorded or composed by the intruder. We allow the intruder to decompose messages that are not encrypted

(14) $\forall u, v\, [Im(pair(u, v)) \supset (Im(u) \wedge Im(v))]$

(15) $\forall u, v, w\, [Im(triple(u, v, w)) \supset (Im(u) \wedge Im(v) \wedge Im(w))]$

(16) $\forall u, v, w, z\, [Im(quadr(u, v, w, z)) \supset (Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z))]$

and to newly compose and send these messages in an arbitrary way.

(17) $\forall u, v\, [(Im(u) \wedge Im(v)) \supset Im(pair(u, v))]$

(18) $\forall u, v, w\, [(Im(u) \wedge Im(v) \wedge Im(w)) \supset Im(triple(u, v, w))]$

(19) $\forall u, v, w, x\, [(Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(x)) \supset Im(quadr(u, v, w, x))]$

(20) $\forall x, y, u\, [(P(x) \wedge P(y) \wedge Im(u)) \supset M(sent(x, y, u))]$

So far, the intruder can only decompose messages, rearrange and send them in an arbitrary way. The next two formulae allow the intruder to consider anything he records/composes as a key for anybody and to encrypt messages that way.

(21) $\forall v, w\, [(Im(v) \wedge P(w)) \supset Ik(key(v, w))]$

(22) $\forall u, v, w\, [(Im(u) \wedge Ik(key(v, w)) \wedge P(w)) \supset Im(encr(u, v))]$

So $Ik(key(v, w))$ holds if the intruder considers v to be a key for principal w. Next we tried SPASS to saturate the formulae (1)–(22).

Fact 3 SPASS *does not terminate on saturating the formulae (1)–(22).*

An analysis (by hand) of the produced clauses shows that the formulae (1)–(22) generate infinitely many clauses of the form

$$\forall u\, [Im(u) \supset M(sent(t, a, triple(encr(quadr(b, kt^{i}(na), kt^{i+1}(na), tb(na)), at), \\ encr(triple(a, kt^{i+1}(na), tb(na)), bt), u)))]$$

where $kt^{i}(na)$ abbreviates the i-fold application of kt to na. In terms of the protocol this corresponds to the following potentially infinite sequence of messages:

$$\begin{array}{l}
I : B \to T : : B, E_{K_{bt}}(A, K_{ab}^{0}, T_b), N_b \\
T : T \to A : : E_{K_{at}}(B, N_a, K_{ab}^{1}, T_b), E_{K_{bt}}(A, K_{ab}^{1}, T_b), N_b \\
I : B \to T : : B, E_{K_{bt}}(A, K_{ab}^{1}, T_b), N_b \\
T : T \to A : : E_{K_{at}}(B, N_a, K_{ab}^{2}, T_b), E_{K_{bt}}(A, K_{ab}^{2}, T_b), N_b \\
\vdots
\end{array}$$

where the first column shows the real sender of the message (the intruder I fakes the sender to be B) and K_{ab}^{i} is the i^{th} key generated by T with $K_{ab}^{0} = K_{ab}$. Thus, the server can be used by the intruder to generate arbitrarily many messages of the form $E_{K_{bt}}(A, K_{ab}^{i}, T_b)$. Since the intruder knows part of the clear-text message (the principal A) and he knows that only the key K_{ab}^{i} differs in all these messages and he can get as many messages of this form as he needs, he may be able to start a known-plaintext attack to the protocol. What may be really crucial here is that the intruder can get as many messages of the above format as he may need to break the key K_{bt} (accordingly for K_{at}).

Fact 4 *The non-termination of* SPASS *on formulae (1)–(22) indicates a potential attack to the protocol.*

We can get rid of this attack by a modification to the server T that causes him to reject his own keys as nonces. The (modified) formulae are

$(10)'$ $\forall xb, xnb, xa, xna, xbet, xbt, xat, xk$

$$[(M(sent(xb, t, triple(xb, xnb, encr(triple(xa, xna, xbet), xbt)))) \wedge Tk(key(xbt, xb)) \wedge Tk(key(xat, xa)) \wedge Nonce(xna)) \supset M(sent(t, xa, triple(encr(quadr(xb, xna, kt(xna), xbet), xat), encr(triple(xa, kt(xna), xbet), xbt), xnb)))]$$

(23) $Nonce(na)$

(24) $\forall x \neg Nonce(kt(x))$

(25) $\forall x\, [Nonce(tb(x)) \wedge Nonce(nb(x))]$

We give the modified set of formulae to SPASS.

Fact 5 SPASS *terminates on saturating the formulae (1)-(9),(10)',(11)-(25) in less than one second.*

The minimal model generated by SPASS is infinite, contains the finite minimal model computed out of the formulae (1)-(12) and is described by 64 clauses. For example, it still contains clauses like $Im(x), Im(y) \rightarrow Im(pair(x, y))$, formula (17), that cause the set of potential intruder messages in the minimal model to be infinite. This clause together with formula (20) and one formula out of (9), (2), (5) implies that the set of sent messages is infinite in the minimal model, too

Now we want to check whether the intruder is able to break the protocol. To this end we give SPASS the conjecture $\exists x\, [Ik(key(x, b)) \wedge Bk(key(x, a))]$ expressing that the intruder owns a key for B which B assumes to be a secure key for A.

Fact 6 SPASS *proves the conjecture* $\exists x\, [Ik(key(x, b)) \wedge Bk(key(x, a))]$ *in less than one second.*

The proof indicates a potential attack to the protocol and it was based on the minimal model generated before (Fact 5). By an inspection of the proof it can be seen that the nonce na is the key shared by the intruder and B to communicate with A. With respect to the original protocol and following the proof found by SPASS, the intruder sends instead of message (iv) the message $E_{K_{bt}}(A, N_a, T_b), E_{K_{N_a}}(N_b)$ to B, where he knows N_a from message (i) and N_b, $E_{K_{bt}}(A, N_a, T_b)$ from message (ii). So, without breaking the keys at or bt, the intruder can break the protocol, if nonces can be confused with keys. We can also get rid of this attack by a refinement to B's behavior on A's message (iv), where B does not accept nonces as keys.

$(7)'$ $\forall xbet, xk, xnb, xa, xna$

$$[(M(sent(xa, b, pair(encr(triple(xa, xk, tb(xna), bt), encr(nb(xna), xk)))) \wedge Sb(pair(xa, xna) \wedge Key(xk)) \quad \supset \quad Bk(key(xk, xa))]$$

(26) $\forall x \neg[Key(x) \wedge Nonce(x)]$

(27) $\forall x\, Key(kt(x))$

Fact 7 SPASS *terminates on saturating the formulae (1)-(6), (7)', (8), (9), (10)', (11)-(27) in less than one second. With respect to the saturation,* SPASS *disproves the conjecture* $\exists x, y, z\, [Ik(key(x, y)) \wedge Bk(key(x, z))]$ *in less than one second.*

Note that we have generalized the conjecture of Fact 6 as we now proved in Fact 7 that the intruder I and B do not share any key at all. The same can be proved by SPASS for possible keys between A and the intruder or the server T and the intruder.

Finally, we also allow the intruder to generate infinitely many nonces that are fresh to B. This enables him to enter the protocol from the very beginning.

(28) $If(ni)$

(29) $\forall x\, [If(x) \supset If(nif(x))]$

(30) $\forall x\, [If(x) \supset (Bf(x) \wedge Im(x))]$

The predicate *If* holds for all fresh intruder nonces that are generated by application of *nif* to the initial fresh nonce *ni*, formulae (28), (29). All fresh intruder nonces are also fresh to B and can be used by the intruder to compose messages, formula (30).

Fact 8 SPASS *terminates on saturating the formulae (1)–(6), (7)′, (8), (9), (10)′, (11)–(30) in less than one second. With respect to the saturation,* SPASS *disproves the conjecture* $\exists x, y, z\, [Ik(key(x, y)) \wedge Bk(key(x, z))]$ *in less than one second.*

So this extension to the possibilities of the intruder does not cause an additional attack. In summary, our analysis showed that the protocol terminates, establishes a secure key between A and B and that an intruder cannot break the protocol as long as nonces are not confused with keys.

3 Monadic Horn Theories

In this section we study the monadic Horn theories involved in the previous section. We adhere to the usual definitions for variables, terms, substitutions, atoms, (positive and negative) literals, multisets, and clauses. We give just the most important definitions for our purpose.

The function *vars* maps terms, atoms, literals, clauses and sets of such objects to the set of variables occurring in these objects. A term t is called *shallow* if t is a variable or is of the form $f(x_1, \ldots, x_n)$ where the x_i are not necessarily different. A term t is called *linear* if every variable occurs at most once in t. It is called *semi-linear* if it is a variable or of the form $f(t_1, \ldots, t_n)$ such that every t_i is semi-linear and whenever $vars(t_i) \cap vars(t_j) \neq \emptyset$ we have $t_i = t_j$ for all i, j.

A *clause* is a multiset of literals. We denote clauses by implications of the form $\Gamma \rightarrow \Delta$ where the multiset Γ contains all atoms occurring negatively in the clause and Δ contains all atoms occurring positively in the clause. We abbreviate $\{A\} \cup \Gamma$ by A, Γ for some atom A.

An (Herbrand) *interpretation* I is a set of ground atoms. For any predicate symbol P, we define $I(P) = \{(t_1, \ldots, t_n) \mid P(t_1, \ldots, t_n) \in I\}$. A ground clause $\Gamma \rightarrow \Delta$ is satisfied by I if $\Gamma \not\subseteq I$ or $\Delta \cap I \neq \emptyset$. A non-ground clause is satisfied by I if all its ground instances are satisfied by I. If a clause C is satisfied by I we also say that I is a *model* for C and write $I \models C$. An interpretation is a model for a set of clauses N ($I \models N$), if it is a model for all $C \in N$. A model I is *minimal* for some set of clauses N, if there is no model J with $J \subset I$ and $J \models N$. The model relation $\models$ can be extended to first-order formulae in the usual way.

A *Horn clause* is a clause with at most one positive literal. A *monadic Horn theory* is a set of Horn clauses where all occurring predicates are monadic. A *declaration* is a clause $S_1(x_1), \ldots, S_n(x_n) \to S(t)$ with $\{x_1, \ldots, x_n\} \subseteq vars(t)$. It is called a *term declaration* if t is not a variable and a *subsort declaration* otherwise. A subsort declaration is called *trivial* if $n = 0$. A term declaration is called *shallow* (*linear*, *semi-linear*) if t is shallow (linear, semi-linear). Note that shallow term declarations do not include arbitrary ground terms. However, any ground term declaration can be equivalently represented, with respect to the minimal model semantics defined below, by finitely many shallow term declarations. A *sort theory* is a finite set of declarations. It is called *shallow* (*linear*, *semi-linear*) if all term declarations are shallow (linear, semi-linear).

Let N be a sort theory. Then we define the interpretation T^N recursively as follows: (i) for every declaration $\to S(t) \in N$, substitution σ such that $S(t)\sigma$ is ground, we define $S(t)\sigma \in T^N$ (ii) for every ground substitution σ, declaration $S_1(x_1), \ldots, S_n(x_n) \to S(t) \in N$, if $S_i(x_i)\sigma \in T^N$, for all $1 \le i \le n$, $t\sigma$ ground, then $S(t)\sigma \in T^N$. It is well-known that T^N is the minimal Herbrand model for a sort theory N.

If N is a sort theory then the *first-order theory over* N is the set of all first-order formulae using only predicate and function symbols occurring in N. The first-order theory over N is *decidable*, if we can decide $T^N \models \phi$ for any formula ϕ in the first-order theory over N.

An *atom ordering* is a well-founded, total ordering on ground atoms. Given an atom ordering $\succ$, we will call an atom A maximal with respect to a multiset of atoms Γ, if for any B in Γ we have $B \not\succ A$. Any atom ordering $\succ$ is extended to an ordering on literals by taking the multiset extension of $\succ$ and by identifying any positive literal A with the singleton $\{A\}$ and any negative literal $\neg A$ with the multiset $\{A, A\}$. With this definition, $\neg A$ is greater than A, but is smaller than any literal B or $\neg B$ with $B \succ A$. The multiset extension of the literal ordering induces an ordering on ground clauses. Let us also use $\succ$ to denote both the extension to literals and clauses of any given atom ordering $\succ$. The clause ordering is compatible with the atom ordering; if the maximal literal in C is greater than the maximal literal in D then $C \succ D$. These notions are lifted to the non-ground level as usual: For two non-ground atoms A, B we define $A \succ B$ if $A\sigma \succ B\sigma$ for all ground instances $A\sigma$, $B\sigma$. We say that a Horn clause $\Gamma \to A$ is *reductive for* the positive literal A, if A is the maximal literal with respect to Γ.

A *selection function* assigns to each (ground) clause a possibly empty set of occurrences of negative literals. If C is a clause and *sel* a selection function then the literal occurrences in $sel(C)$ are called *selected*. In particular, $sel(C) = \emptyset$ indicates that no literal is selected.

Definition 1. *An inference by* ordered resolution (with selection) *between two Horn clauses takes the form*

$$\frac{\Gamma \rightarrow A \qquad B, \Lambda \rightarrow \Delta}{\Gamma\sigma, \Lambda\sigma \rightarrow \Delta\sigma}$$

such that (i) σ is the most general unifier between A and B (ii) $\Gamma\sigma \rightarrow A\sigma$ is reductive for $A\sigma$, (iii) no literal is selected in Γ, and (iv) B is selected, or else no literal is selected in $B, \Lambda \rightarrow \Delta$ and $B\sigma$ is maximal in $B\sigma, \Lambda\sigma \rightarrow \Delta\sigma$. The multiset Δ is either empty or contains exactly one atom.

The inference rule *sort resolution* we employ in this paper is an instance of ordered resolution. Using an appropriate option setting SPASS implements sort resolution with respect to the theories considered in the previous section. The used ordering is any atom ordering satisfying $T(s) \succ S(t)$ if s contains t as a proper subterm. For example, a Knuth-Bendix ordering where all function and predicate symbols have weight one has this property. Given a clause $C = S_1(t_1), \ldots, S_i(t_n) \rightarrow S(t)$ the selection function *sor* for sort resolution is defined by $S_i(t_i) \in sor(C)$ if (i) t_i is a non-variable term or (ii) all t_j are variables and t_i is a variable that does not occur in t or (iii) all t_j are variables occurring in t and t is a variable.

Let $\succ$ be a total atom ordering and *sel* a selection function. Given a set of ground clauses N, we use induction with respect to $\succ$ to define a Herbrand interpretation I_C and a set E_C, for each clause C in N, as follows.

Definition 2. *Let I_C be the set $\bigcup_{C \succ D} E_D$. Furthermore, $E_C = \{A\}$ if (i) $C = \Gamma \rightarrow A$ is reductive for A, (ii) $\Gamma \subseteq I_C$ and (iii) $A \notin I_C$. Otherwise, E_C is the empty set.*

If $E_C = \{A\}$, we also say that C *produces* A and call C a *productive clause*. Finally, by I, we denote the Herbrand interpretation $\bigcup_{C \in N} E_C$. Whenever we need to emphasize the dependency of the interpretation I from the particular clause set N, we will use the notation I^N. If N is clause set containing non-ground clauses, then I^N is the interpretation generated by all ground instances of clauses from N. A non-ground clause $C \in N$ is called productive if a ground instance of C is productive for I^N.

A clause C is a *condensation* of a clause D, if C is a proper (unordered) factor of D that subsumes D. A set of clauses N is called *saturated* if it is closed under condensation, the deletion of subsumed clauses and any clause generated by an ordered resolution inference from clauses from N is subsumed by some clause in N. Ordered resolution in general allows more powerful notions of simplification/redundancy (Bachmair & Ganzinger 1994), but for the purpose of this paper subsumption and condensation suffices.

Corollary 1 (Bachmair & Ganzinger (1994)). *Let N be a set of Horn clauses saturated by ordered resolution. Then either N contains the empty clause or I^N is a model for N.*

Lemma 1. *Let N be a monadic Horn theory saturated by sort resolution that does not contain the empty clause. Then $I^N = T^{N'}$ where N' is the set of all term declarations and trivial subsort declarations occurring in N.*

Proof. First, we show by contradiction that all productive clauses in N are either term declarations or trivial subsort declarations. This implies $I^N = I^{N'}$. So assume $C\sigma = S_1(t_1)\sigma, \ldots, S_n(t_n)\sigma \to S(t)\sigma$ is the minimal (with respect to $\succ$) ground instance of a clause $C \in N$ that produces $S(t)\sigma$ but some $S_i(t_i)$ is selected in C. Since $C\sigma$ produces $S(t)\sigma$, we know $\{S_1(t_1)\sigma, \ldots, S_n(t_n)\sigma\} \subseteq I^N_{C\sigma}$ and $S(t)\sigma \notin I^N_{C\sigma}$. So there is a ground clause $D\tau = T_1(s_1)\tau, \ldots, T_m(s_m)\tau \to S_i(s)\tau$, $D \in N$, $S_i(s)\tau = S_i(t_i)\sigma$ where $D\tau \prec C\sigma$. No literal in D is selected (we chose $C\sigma$ to be minimal) and therefore we can generate a sort resolution resolvent R from C and D. This resolvent has a ground instance $R\lambda = (S_1(t_1), \ldots, S_{i-1}(t_{i-1}), T_1(s_1)\tau, \ldots, T_m(s_m)\tau, S_{i+1}(t_{i+1}), \ldots, S_n(t_n) \to S(t))\sigma$, $R\lambda \prec C\sigma$, is productive and produces $S(t)\sigma$ contradicting that $C\sigma$ produces $S(t)\sigma$. Therefore, every productive clause in N has no selected literal and is hence a term declaration or a trivial subsort declaration.

Second, we show $I^{N'} \subseteq T^{N'}$ by induction on the clause ordering. If $S(t)\sigma \in I^{N'}$ then there is a productive clause $S_1(x_1), \ldots, S_n(x_n) \to S(t) \in N'$. If $n = 0$ then by definition of $T^{N'}$, case (i), we have $S(t)\sigma \in T^{N'}$. If $n \neq 0$ then by definition of $I^{N'}$ we know $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq I^{N'}$ and therefore by induction hypothesis $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq T^{N'}$. Now, by definition of $T^{N'}$, case (ii), we have $S(t)\sigma \in T^{N'}$.

Third, we show $T^{N'} \subseteq I^{N'}$ by structural induction on the definition of $T^{N'}$. If $S(t)\sigma \in T^{N'}$ is generated by some clause $\to S(t) \in N'$ then $\to S(t)\sigma$ is productive and hence $S(t)\sigma \in I^{N'}$. If $S(t)\sigma \in T^{N'}$ is generated by some clause $C = S_1(x_1), \ldots, S_n(x_n) \to S(t) \in N'$ we know $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq T^{N'}$ and hence, by induction hypothesis, $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq I^{N'}$. Furthermore, C is a term declaration and by definition of our ordering C is reductive for $S(t)$. Hence, $S_1(x_1)\sigma, \ldots, S_n(x_n)\sigma \to S(t)\sigma$ is productive and generates $S(t)\sigma \in I^{N'}$.

Lemma 2. *Let N be a semi-linear sort theory. Then the first-order theory over N is decidable.*

Proof. First, we transform N into a shallow sort-theory N'. We recursively replace every declaration $S_1(x_1), \ldots, S_n(x_n) \to S(f(t_1, \ldots, t_n))$ where t_i is not a variable by two new declarations

$$S_{m_1}(x_{m_1}), \ldots, S_{m_l}(x_{m_l}), R(y) \to S(f(s_1, \ldots, s_n))$$
$$S_{j_1}(x_{j_1}), \ldots, S_{j_k}(x_{j_k}) \to R(t_i)$$

where y and R are new, $s_j = t_j$ if $t_j \neq t_i$, $s_j = y$ if $t_j = t_i$ for all $1 \leq j \leq n$, $vars(t_i) = \{x_{j_1}, \ldots, x_{j_k}\}$ and $vars(f(t_1, \ldots, t_n)) \setminus vars(t_i) = \{x_{m_1}, \ldots, x_{m_l}\}$. Since $f(t_1, \ldots, t_n)$ is semi-linear, t_i and $f(s_1, \ldots, s_n)$ are semi-linear as well. The transformation terminates generating a shallow sort theory N' and by an *induction argument it can be proved that $T^N(P) = T^{N'}(P)$ for any (monadic) predicate P occurring in N. Second, Weidenbach (1998) showed that any shallow sort theory N' can be transformed into a $Rec_=$ tree automaton (Bogaert & Tison 1992) such that for any monadic predicate P the language accepted by the tree automaton in the state corresponding to P is exactly $T^{N'}(P)$. Third, Comon & Delor (1994) showed that the first-order theory over $Rec_=$ automata can be decided.*

Theorem 9. *Let N be a monadic Horn theory finitely saturated by sort resolution. If all term declarations in N are semi-linear, then the first-order theory over the productive clauses in N is decidable.*

Proof. *By Lemma 1 and Lemma 2.*

The technique used to prove Theorem 9 is a combination of results obtained in the context of saturation based theorem proving with results developed in the area of finite tree automata. In particular, saturation based theorem proving cannot be a priori used to decide the first-order theory for some finitely saturated semi-linear sort theory. One problem is that Skolemization cannot be carried out in the usual way, since existential quantifiers must not introduce new symbols but have to be interpreted over the minimal model of the sort theory. The fragment for conjectures we used in the previous section is indeed decidable by saturation based methods in general.

Lemma 3. *Let N be a semi-linear sort theory. Then the first-order fragment $\exists x_1, \ldots, x_n\, [A_1 \wedge \ldots \wedge A_k]$ from the first-order theory over N where all A_i are atoms can be decided by sort resolution.*

Proof. *The formula $\exists x_1, \ldots, x_n\, [A_1 \wedge \ldots \wedge A_k]$ holds in T^N iff $\forall x_1, \ldots, x_n\, [\neg A_1 \vee \ldots \vee \neg A_k]$ does not hold in T^N. This can be checked by sort resolution saturating the set $N \cup \{A_1, \ldots, A_k \rightarrow\}$. This was shown to be decidable by Jacquemard, Meyer & Weidenbach (1998).*

The above Lemma 3 explains the success of Spass on the queries tested for Fact 2, Fact 6 and Fact 7 in Section 2. The saturated theory, Fact 1, is a ground theory and therefore semi-linear. This ground theory is the result of saturating the clauses resulting from the formulae (1)–(12), see Section 2, by sort resolution. By deleting all non-productive clauses, the remaining clauses are exactly the atoms shown on page 5 plus the ground atoms that are already contained in the formulae (1)–(12). Hence, Spass can in fact decide the query of Fact 2. The saturated theories that are the basis for the conjectures investigated in Fact 6 and Fact 7 are syntactically not semi-linear but contain non-linear variable occurrences at different depth. However, all these occurrences are restricted by monadic predicates having a finite extension in the minimal model. Lemma 3 also holds for this extension.

Saturating a sort theory extended by a clause $A_1, \ldots, A_k \rightarrow$ is a process closely related to sorted unification (Weidenbach 1998). This is even decidable for sort theories extended by certain, restricted forms of equations as shown by Jacquemard et al. (1998). What remains to be shown is in which cases saturation of a monadic Horn theory terminates.

Lemma 4. *Let N be a monadic Horn theory where all positive literals are linear and shallow. Then N can be finitely saturated by sort resolution and the productive clauses of the saturated theory form a linear shallow sort theory.*

Proof. *Since any clause where no negative literal is selected is either a trivial subsort declaration or a linear shallow term declaration, any application of sort resolution takes one of the following three forms:*

$$\frac{S_1(x_{i_1}), \ldots, S_k(x_{i_k}) \to S(f(x_1, \ldots, x_n)) \qquad S(f(t_1, \ldots, t_n)), \Lambda \to \Delta}{S_1(x_{i_1})\sigma, \ldots, S_k(x_{i_k})\sigma, \Lambda \to \Delta}$$

where $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq \{x_1, \ldots, x_n\}$ *or*

$$\frac{S_1(x_{i_1}), \ldots, S_k(x_{i_k}) \to S(f(x_1, \ldots, x_n)) \qquad S(y), \Lambda \to \Delta}{S_1(x_{i_1}), \ldots, S_k(x_{i_k}), \Lambda\tau \to \Delta\tau}$$

where $\tau = \{y \mapsto f(x_1, \ldots, x_n)\}$, *again* $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq \{x_1, \ldots, x_n\}$ *or*

$$\frac{\to S(x) \qquad S(t), \Lambda \to \Delta}{\Lambda \to \Delta}$$

For all three types of inferences the following invariant holds: All non-variable terms in the resolvent are proper subterms of the right parent clause or the resolvent contains at most one linear shallow non-variable term. Since N contains only finitely many different terms and only monadic predicates, there are only finitely many resolvents that can be generated with respect to subsumption and condensation. Hence, the saturation terminates. Furthermore, for every type of a resolution inference, if the resolvent contains a positive atom $S(t)$ then t is a linear shallow term. Hence, all productive clauses in the saturated theory are either trivial subsort declarations or linear shallow term declarations and therefore form a linear shallow sort theory.

The restrictions on the terms required for Lemma 4 are close to the border of non-termination (undecidability). If we only require declarations to be linear, then even if we additionally require that every literal is linear, the saturation process does not terminate, in general. This can be shown by a reduction of the ground word problem for word equations to this problem. We encode equality by a two place function symbol e and add one monadic predicate T for "truth". Then all axioms of equality, except reflexivity, result monadic Horn clauses with linear literals. The congruence axioms result in linear clauses because all function symbols (except e itself) are monadic. Reflexivity would result in a non-linear clause. However, it is sufficient for the reduction to code reflexivity for ground terms, i.e, the occurring constants, resulting in linear clauses. Finally, any word equation can be transformed into a linear Horn clause by expressing non linearities through linear disequations. For example, the word equation $ab \approx a$

that corresponds to the term equation $a(b(x)) \approx b(x)$ is expressed by the linear monadic Horn clause $T(e(b(x), y)) \rightarrow T(e(a(b(x)), y))$.

If we only require the positive literals in the Horn theories of Lemma 4 to be shallow, the resulting theories can also not be finitely saturated, in general. We can simply reduce the undecidable unifiability problem for arbitrary sort theories to this problem. We code an arbitrary term declaration $S_1(x_1), \ldots, S_n(x_n) \rightarrow S(t)$ by $S_1(x_1), \ldots, S_n(x_n), T(f(y, t)) \rightarrow S(y)$ where y, f and T are new and add the shallow declaration $\rightarrow T(f(x, x))$.

With respect to the saturation results in Section 2, Lemma 4 partly explains the behavior of SPASS. All considered sets of formulae in Section 2, have positive occurrences of non-linear atoms and Fact 3 shows that the suggested techniques are in general not strong enough to guarantee termination. However, the theory we used in the previous section for the intruder, is a theory where all positive atoms are linear and shallow. So, by Lemma 4, this theory can always be finitely saturated. Note that this theory is independent of the investigated protocol as long as it can be represented by a set of messages. Furthermore, this theory can still be finitely saturated if we add finitely many ground term declarations. The productive clauses of the saturation of the initial protocol, Fact 1, solely contains ground declarations. Hence, the non-termination of the saturation of the protocol theory extended by the intruder theory indicates an infinite number of messages "exchanged" by the protocol and the intruder, the messages we isolated below Fact 3. Note that none of the above termination results for sort resolution holds for standard resolution or ordered resolution without our particular selection strategy.

4 Conclusion

This paper consists of two major parts: First, it shows that using automated theorem proving techniques security protocols like the Neumann-Stubblebine protocol can be successfully analyzed. The suggested techniques apply to a variety of protocols, e.g, further key exchange protocols (see Schneier (1996)). In particular, we can finitely model an "infinite" intruder. Second, we have been able to prove that parts of the used first-order fragments can be decided in general. This includes more sophisticated properties than the properties we tested for the Neumann-Stubblebine protocol. The decidability results are also useful in a more general context. For example, Lemma 4 offers a new fine grained approximation for Horn programs (Charatonik, McAllester, Niwinski, Podelski & Walukiewicz 1998).

So far our formalization mainly models "reachability", i.e., that certain situations can(not) occur. For many protocols "liveness" properties, e.g., that a certain situation will definitely occur, require a more sophisticated formalization. We already successfully analyzed signature exchange protocols with respect to such properties using SPASS. In order to explain the termination of SPASS on these experiments, the results of Section 3 have to be extended to the non-Horn equality case. So an important direction of future research is to extend the decidability results with respect to the needed fragments. The results of the experiments with SPASS we did so far support that this should be possible. Furthermore, our results indicate that it should be possible to design an automatic tool for the analysis of security protocols that is not restricted to a finite state model.

Acknowledgments: I'm indebted to Harald Ganzinger, Andreas Nonnengart, Birgit Pfitzmann and Matthias Schunter for numerous useful comments and discussions. The comments of the reviewers helped a lot in improving the quality of the paper.

References

Bachmair, L. & Ganzinger, H. (1994), 'Rewrite-based equational theorem proving with selection and simplification', *Journal of Logic and Computation* **4**(3), 217–247.

Bogaert, B. & Tison, S. (1992), Equality and disequality constraints on direct subterms in tree automata, *in* A. Finkel & M. Jantzen, eds, 'Proceedings of 9th Annual Symposium on Theoretical Aspects of Computer Science, STACS92', Vol. 577 of *LNCS*, Springer, pp. 161–171.

Burrows, M., Abadi, M. & Needham, R. (1990), 'A logic of authentication', *ACM Transactions on Computer Systems* **8**(1), 18–36.

Charatonik, W., McAllester, D., Niwinski, D., Podelski, A. & Walukiewicz, I. (1998), The horn mu-calculus, *in* 'Proceedings 13th IEEE Symposium on Logic in Computer Science, LICS'98', IEEE Computer Society Press, pp. 58–69.

Comon, H. & Delor, C. (1994), 'Equational formulae with membership constraints', *Information and Computation* **112**, 167–216.

Jacquemard, F., Meyer, C. & Weidenbach, C. (1998), Unification in extensions of shallow equational theories, *in* T. Nipkow, ed., 'Rewriting Techniques and Applications, 9th International Conference, RTA-98', Vol. 1379 of *LNCS*, Springer, pp. 76–90.

Mitchell, J. C. (1998), Finite-state analysis of security protocols, *in* A. J. Hu & M. Y. Vardi, eds, 'Computer Aided Verification (CAV-98) : 10th International Conference', Vol. 1427 of *LNCS*, Springer, pp. 71–76.

Neuman, B. C. & Stubblebine, S. G. (1993), 'A note on the use of timestamps as nonces', *ACM SIGOPS, Operating Systems Review* **27**(2), 10–14.

Paulson, L. C. (1997), Proving properties of security protocols by induction, *in* J. Millen, ed., 'Proceedings of the 10th IEEE Computer Security Foundations Workshop', IEEE Computer Society, pp. 70–83.

Schneier, B. (1996), *Applied Cryptography*, 2 edn, Wiley.

Schumann, J. (1997), Automatic verification of cryptographic protocols with setheo, *in* 'Proceedings of the 14th International Conference on Automated Deduction, CADE-14', Vol. 1249 of *LNAI*, Springer, Townsville, Australia, pp. 87–100.

Weidenbach, C. (1998), Sorted unification and tree automata, *in* W. Bibel & P. H. Schmitt, eds, 'Automated Deduction - A Basis for Applications', Vol. 1 of *Applied Logic*, Kluwer, chapter 9, pp. 291–320.

Weidenbach, C., Afshordel, B., Brahm, U., Cohrs, C., Engel, T., Keen, E., Theobalt, C. & Topic, D. (1999), System description: Spass version 1.0.0, *in* H. Ganzinger, ed., '16th International Conference on Automated Deduction, CADE-16', LNAI, Springer. This volume.