Fairness

Problem:

If N is inconsistent, then $N \mid \emptyset \mid \emptyset \ \overset{*}{\rhd}\ N' \cup \{\bot\} \mid \_ \mid \_$.

Does this imply that *every* derivation starting from an inconsistent set N eventually produces $\bot$?

No: a clause could be kept in $\boldsymbol{P}$ without ever being used for an inference.

Fairness

We need in addtion a fairness condition:

If an inference is possible forever (that is, none of its premises is ever deleted), then it must be computed eventually.

One possible way to guarantee fairness:
Implement $\boldsymbol{P}$ as a queue
(there are other techniques to guarantee fairness).

With this additional requirement, we get a stronger result:
If N is inconsistent, then every *fair* derivation will eventually produce $\perp$.

Hyperresolution

There are *many* variants of resolution.
(We refer to [Bachmair, Ganzinger: Resolution Theorem Proving] for further reading.)

One well-known example is hyperresolution (Robinson 1965):

Assume that several negative literals are selected in a clause D. If we perform an inference with D, then one of the selected literals is eliminated.

Suppose that the remaining selected literals of D are again selected in the conclusion. Then we must eliminate the remaining selected literals one by one by further resolution steps.

Hyperresolution

Hyperresolution replaces these successive steps by a single inference.
As for $Res_S^{\succ}$, the calculus is parameterized by an atom ordering $\succ$ and a selection function S.

Hyperresolution

$$\frac{C_1 \vee A_1 \quad \ldots \quad C_n \vee A_n \qquad \neg B_1 \vee \ldots \vee \neg B_n \vee D}{(C_1 \vee \ldots \vee C_n \vee D)\sigma}$$

with $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, if

(i) $A_i\sigma$ strictly maximal in $C_i\sigma$, $1 \leq i \leq n$;

(ii) nothing is selected in C_i;

(iii) the indicated occurrences of the $\neg B_i$ are exactly the ones selected by S, or else nothing is selected in the right premise and $n = 1$ and $\neg B_1\sigma$ is maximal in $D\sigma$.

Similarly to resolution, hyperresolution has to be complemented by a factoring inference.

Hyperresolution

As we have seen, hyperresolution can be simulated by iterated binary resolution.

However this yields intermediate clauses which HR might not derive, and many of them might not be extendable into a full HR inference.

2.13 Example: Neuman-Stubblebine Protocol

- Formalization of a concrete application:
  Neuman-Stubblebine key exchange protocol.
- State-of-the-art in automated theorem proving.
- Proof by refutation:
  inconsistency $\Rightarrow$ intruder can break the protocol.
- Proof by consistency:
  consistency $\Rightarrow$ no unsafe states exist.
- Termination requires elimination of redundancy.

The Problem

Automatic Analysis of Security Protocols using SPASS: An Automated Theorem Prover for First-Order Logic with Equality

by Christoph Weidenbach

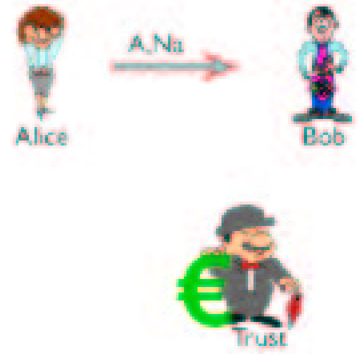
The growing importance of the internet causes a growing need for security protocols that protect transactions and communication. It turns out that the design of such protocols is highly error-prone. Therefore, there is a need for tools that automatically detect flaws like, e.g., attacks by an intruder. Here we show that our automated theorem prover SPASS can successfully be used to analyze the Neuman-Stubblebine key exchange protocol [1]. To this end the protocol is formalized in logic and then the security properties are automatically analyzed by SPASS. A detailed description of the analysis can be found in [2].

The Problem

The animation successively shows two runs of the Neuman-Stubblebine [1] key exchange protocol. The first run works the way the protocol is designed to do, i.e., it establishes a secure key between Alice and Bob.

The second run shows a potential problem of the protocol. An intruder may intercept the final message sent from Alice to Bob, replace it with a different message and may eventually own a key that Bob believes to be a secure key with Alice. The initial situation for the protocol is that the two participants Alice and Bob want to establish a secure key for communication among them. They do so with the help of a trusted server Trust where both already have a secure key for communication with Trust. The below picture shows a sequence of four message exchanges that eventually establishes the key.

1



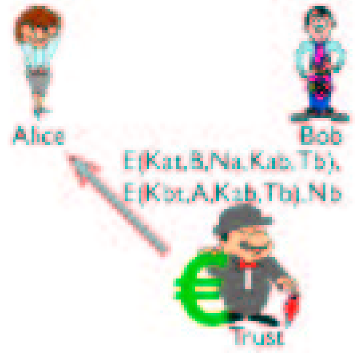
Alice sends Bob the message 1: A, Na.
It consists of her identification A together with a nonce, a random number Na. The purpose of Na is to make this run unique in order to prevent replay attacks of some intruder recording the messages.

2

Alice
Bob
B,E(Kbt,A,Na,Tb),Nb
Trust

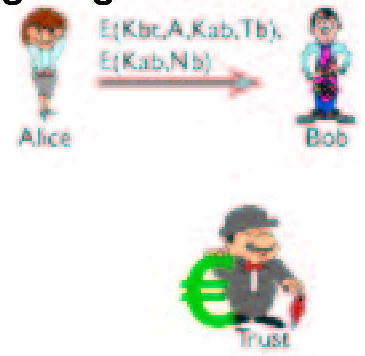
Bob sends the message 2:
B, E(Kbt,A,Na,Tb), Nb to Trust.
After having received Alice's message, Bob knows that Alice wants to establish a key with him. So he sends message 2 to Trust. It consists of his identification, an encrypted middle part and again a nonce. The encrypted middle part E(Kbt,A,Na,Tb) stands for the message A,Na,Tb encrypted with the key Kbt, the secure key that Bob and the server Trust share. The time span Tb fixes the expiration time for the eventually generated key.

3



Trust sends the message 3:
E(Kat,B,Na,Kab,Tb), E(Kbt,A,Kab,Tb), Nb to Alice. Trust reads the previous message, generates the new secure key Kab for communication between Alice and Bob and sends message 3 to Alice. The first part E(Kat,B,Na,Kab,Tb) can be decrypted by Alice, whereas the second part is meant to be forwarded by Alice to Bob, see message 4.

Alice sends the message 4: E(Kbt,A,Kab,Tb), E(Kab,Nb) to Bob. Alice reads message 3, decrypts the first part with the secure key Kat she shares with Trust and extracts the content of the message. In particular, the new key Kab for communication with Bob. Then she forwards the second part of Trust's message together with E(Kab,Nb) to Bob.



4

5

Bob receives message 4, decrypts the first part and extracts the key Kab, uses this key to decrypt the second part and by inspecting Nb he authentificates Alice. Eventually, Alice and Bob share the key Kab. Everything is fine.

Neuman-Stubblebine: A Regular Run

1. $$\begin{tikzcd} \boxed{A} \arrow[r, "{A, N_a}"] & \boxed{B} \end{tikzcd}$$

2. $$\begin{tikzcd} \boxed{B} \arrow[r, "{B, N_b, E_{K_{bt}}(A, N_a, T_b)}"] & \boxed{T} \end{tikzcd}$$

3. $$\begin{tikzcd} \boxed{T} \arrow[r, "{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b}"] & \boxed{A} \end{tikzcd}$$

4. $$\begin{tikzcd} \boxed{A} \arrow[r, "{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)}"] & \boxed{B} \end{tikzcd}$$

What Can Happen?

How can an intruder now break this protocol? The key K_{ab} is only transmitted inside encrypted parts of messages and we assume that an intruder cannot break any keys nor does he know any of the initial keys K_{at} or K_{bt}. Here is the solution:

1

E(Kb A Kab,Tb),
E(Kab Nb)
Alice
Bob
Intruder
Trust

Intruder intercepts the message

2

Alice
Bob
E(Kbt,A,Na,Tb),
E(Na,Nb)
Intruder
Trust

Intruder sends the message E(Kbt,A,Na,Tb),E(Na,Nb) to Bob. The problem is that the part E(Kbt,A,Kab,Tb) of message 4 and the part E(Kbt,A,Na,Tb) of message 2 are nearly identical. The only difference is that at the position of Kab in message 4, there is the nonce Na in message 2. The intruder can of course record all messages and intercept/send messages. So he catches message 4 and instead he sends message 5 to Bob.

3

Bob decrypts message 5 and now believes that Na is a secure key he shares with Alice.

4

Bob starts communication with Alice,

5

but talks to Intruder.

Breaking Neuman-Stubblebine

1. $$\boxed{A} \xrightarrow{\quad A,\ N_a \quad} \boxed{B}$$

2. $$\boxed{B} \xrightarrow{\quad B,\ N_b,\ E_{K_{bt}}(A,\ N_a,\ T_b) \quad} \boxed{T}$$

3. $$\boxed{T} \xrightarrow{\quad E_{K_{at}}(B,\ N_a,\ K_{ab},\ T_b),\ E_{K_{bt}}(A,\ K_{ab},\ T_b),\ N_b \quad} \boxed{A}$$

4. $$\boxed{A} \xrightarrow{\quad E_{K_{bt}}(A,\ K_{ab},\ T_b),\ E_{K_{ab}}(N_b) \quad} \boxed{B}$$

3. $$\boxed{I} \xrightarrow{\quad E_{K_{bt}}(A,\ N_a,\ T_b),\ E_{N_a}(N_b) \quad} \boxed{B}$$

The Formalization

The key idea of the formalization is to describe the set of sent messages. This is done by introducing a monadic predicate M in first-order logic. Furthermore, every participant holds its set of known keys, represented by the predicates Ak for Alice's keys, Bk for Bob's keys, Tk for Trust's keys and Ik for the keys the intruder knows. The rest of the used symbols is introduced and explained with the first appearance in a formula. Then the four messages can be translated into the following formulae:

The Formalization

Step 1) A, Na

$$Ak(key(at, t)) \tag{1}$$

$$M(sent(a, b, pair(a, na))) \tag{2}$$

The two formulae express that initially Alice holds the key *at* for communication with *t* (for Trust) and that she sends the first message. In order to formalize messages we employ a three place function sent where the first argument is the sender, the second the receiver and the third the content of the message. So the constant *a* represents Alice, *b* Bob, *t* Trust and *i* Intruder. The functions *pair* (*triple*, *quadr*) simply form sequences of messages of the indicated length.

The Formalization

Step 2) $B, E(Kbt, A, Na, Tb), Nb$

$$Bk(key(bt, t)) \tag{3}$$

$$\begin{aligned} \forall xa, xna\ [M(sent(xa, b, pair(xa, xna))) \\ \rightarrow M(sent(b, t, triple(b, nb(xna)), \\ encr(triple(xa, xna, tb(xna)), bt)))))] \end{aligned} \tag{4}$$

Bob holds the key *bt* for secure communication with Trust and whenever he receives a message of the form of message 1 (formula (2)), he sends a key request to Trust according to message 2. Note that encryption is formalized by the two place function *encr* where the first argument is the date and the second argument the key. Every lowercase symbol starting with an *x* denotes a variable. The functions *nb* and *tb* generate, respectively, a new nonce and time span out of *xa*'s (Alice's) request represented by her nonce *xna*.

The Formalization

Step 3) $E(Kat, B, Na, Kab, Tb),\ \ E(Kbt, A, Kab, Tb),\ \ Nb$

$$Tk(key(at, a))) \wedge Tk(key(bt, b)) \tag{5}$$

$$\begin{aligned}
&\forall xb, xnb, xa, xna, xbet, xbt, xat, xk \\
&\quad [\ (M(sent(xb, t, triple(xb, xnb, encr(triple(xa, xna, xbet), xbt)))) \\
&\qquad \wedge Tk(key(xbt, xb)) \\
&\qquad \wedge Tk(key(xat, xa))) \\
&\qquad \rightarrow M(sent(t, xa, triple(encr(quadr(xb, xna, kt(xna), xbet), xat), \\
&\qquad\qquad encr(triple(xa, kt(xna), xbet), xbt), xnb)))\ ]
\end{aligned} \tag{6}$$

Trust holds the keys for Alice and Bob and answers appropriately to a message in the format of message 2. Note that decryption is formalized by unification with an appropriate term structure where it is checked that the necessary keys are known to Trust. The server generates the key by applying his key generation function *kt* to the nonce *xna*.

The Formalization

Step 4) $E(Kbt, A, Kab, Tb),\ \ E(Kab, Nb)$

$$\begin{aligned}
&\forall xnb, xbet, xk, xm, xb, xna \\
&\qquad [\ M(sent(t, a, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb)) \\
&\qquad\quad \rightarrow (M(sent(a, xb, pair(xm, encr(xnb, xk)))) \wedge Ak(key(xk, xb)))\ ]
\end{aligned} \tag{7}$$

$$\begin{aligned}
&\forall xbet, xk, xnb, xa, xna \\
&\qquad [\ M(sent(xa, b, pair(encr(triple(xa, xk, tb(xna)), bt), \\
&\qquad\qquad\qquad encr(nb(xna), xk))) \rightarrow Bk(key(xk, xa))]
\end{aligned} \tag{8}$$

Finally, Alice answers according to the protocol to message 3 and stores the generated key for communication, formula (7). Formula (8) describes Bob's behaviour when he receives Alice's message. Bob decodes this message and stores the new key as well.

A's Formalization Part I

$$\begin{tikzcd} \boxed{A} \arrow[r, "{A, N_a}"] & \boxed{B} \end{tikzcd}$$

$$P(a)$$

$$Ak(key(at, t))$$

$$M(sent(a, b, pair(a, na)))$$

$$Sa(pair(b, na))$$

Sa is Alice's local store that will eventually be used to verify the nonce when it is sent back to her in Step (3).

A's Formalization Part II

$$\begin{tikzcd}
T \arrow[r, "{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b}"] & A \\
A \arrow[r, "{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)}"] & B
\end{tikzcd}$$

$$
\begin{aligned}
&\forall xb, xna, xnb, xk, xbet, xm \\
&\quad [\, M(sent(t, a, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb))) \\
&\quad \wedge Sa(pair(xb, xna)) \\
&\quad \longrightarrow \\
&\quad M(sent(a, xb, pair(xm, encr(xnb, xk)))) \\
&\quad \wedge Ak(key(xk, xb)) \,]
\end{aligned}
$$

The Intruder

The Intruder is modeled as an exhaustive hacker. He records all messages, decomposes the messages as far as possible and generates all possible new compositions. Furthermore, any object he has at hand is considered as a key and tried to used for encryption as well as for decryption. All these messages are posted. The set of messages the intruder has available is represented by the predicate *Im*.

The participants are Alice, Bob, Trust and Intruder:

$$P(a) \wedge P(b) \wedge P(t) \wedge P(i) \tag{9}$$

The intruder records all messages:

$$\forall xa, xb, xm \; [M(sent(xa, xb, xm)) \rightarrow Im(xm)] \tag{10}$$

The Intruder

He decomposes and decrypts all messages he owns the key for:

$$\forall u, v\ [Im(pair(u, v)) \rightarrow Im(u) \wedge Im(v)] \tag{11}$$

$$\forall u, v, w\ [Im(triple(u, v, w)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w)] \tag{12}$$

$$\forall u, v, w, z\ [Im(quadr(u, v, w, z)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)] \tag{13}$$

$$\forall u, v, w\ [Im(encr(u, v)) \wedge Ik(key(v, w)) \rightarrow Im(u)] \tag{14}$$

He composes all possible messages:

$$\forall u, v\ [Im(u) \wedge Im(v) \rightarrow Im(pair(u, v))] \tag{15}$$

$$\forall u, v, w\ [Im(u) \wedge Im(v) \wedge Im(w) \rightarrow Im(triple(u, v, w))] \tag{16}$$

$$\forall u, v, w, x\ [Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(x) \rightarrow Im(quadr(u, v, w, x))] \tag{17}$$

The Intruder

He considers every item to be a key and uses it for encryption:

$$\forall v, w\ [Im(v) \wedge P(w) \rightarrow Ik(key(v, w))] \tag{18}$$

$$\forall u, v, w\ [Im(u) \wedge Ik(key(v, w)) \wedge P(w) \rightarrow Im(encr(u, v))] \tag{19}$$

He sends everything:

$$\forall x, y, u\ [P(x) \wedge P(y) \wedge Im(u) \rightarrow M(sent(x, y, u))] \tag{20}$$

Finally we must formalize the insecurity requirement. Intruder must not have any key for communication with Bob that Bob believes to be a secure key for Alice:

$$\exists x\ [Ik(key(x, b)) \wedge Bk(key(x, a))]$$

SPASS Solves the Problem

Now the protocol formulae together with the intruder formulae (9)-(20) and the insecurity formula above can be given to SPASS. Then SPASS automatically proves that this formula holds and that the problematic key is the nonce *Na*. The protocol can be repaired by putting type checks on the keys, such that keys can no longer be confused with nonces. This can be added to the SPASS first-order logic formalization. *Then SPASS disproves the insecurity formula above*. This capability is currently unique to SPASS. Although some other provers might be able to prove that the insecurity formula holds in the formalization without type checks, we are currently not aware of any prover that can disprove the insecurity formula in the formalization with type checking. Further details can be found in [2], below. The experiment is available in full detail from the SPASS home page in the download area.

SPASS Solves the Problem

References:

[1] Neuman, B. C. and Stubblebine, S. G., 1993, A note on the use of timestamps as nonces, ACM SIGOPS, Operating Systems Review, 27(2), 10-14.

[2] Weidenbach, C., 1999, Towards an automatic analysis of security protocols in first-order logic, in 16th International Conference on Automated Deduction, CADE-16, Vol. 1632 of LNAI, Springer, pp. 378-382.

2.14 Summary: Resolution Theorem Proving

- Resolution is a machine calculus.
- Subtle interleaving of enumerating ground instances and proving inconsistency through the use of unification.
- Parameters: atom ordering $\succ$ and selection function S.
On the non-ground level, ordering constraints can (only) be solved approximatively.
- Completeness proof by constructing candidate models from productive clauses $C \vee A$, $A \succ C$; inferences with those reduce counterexamples.

Summary: Resolution Theorem Proving

- *Local* restrictions of inferences via $\succ$ and S
  $\Rightarrow$ fewer proof variants.
- *Global* restrictions of the search space via elimination of redundancy
  $\Rightarrow$ computing with "smaller" clause sets;
  $\Rightarrow$ termination on many decidable fragments.
- However: not good enough for dealing with orderings, equality and more specific algebraic theories (lattices, abelian groups, rings, fields)
  $\Rightarrow$ further specialization of inference systems required.