An Efficient Cryptographic Protocol Verifier Based on Prolog Rules

Bruno Blanchet
INRIA Rocquencourt

June 2001

Introduction

Goal: verify cryptographic protocols.

Our goal is to solve two problems:

- State space explosion.
- Limit on the number of sessions of the protocol.

Example: secrecy properties of Skeme in 23 s, less than 2 Mb.

Our solution

Novelties:

- a simple abstract representation of these protocols, by a set of Horn clauses (Prolog rules);

- an efficient solving algorithm to find which facts can be derived from these clauses.

Using this, we can prove secrecy properties of protocols, or exhibit attacks showing why a message is not secret.

For the moment, we handle shared- and public-key cryptography, hash functions, Diffie-Hellman key agreements.

Protocol representation

- Messages $\rightsquigarrow$ terms

  $$M ::= x \mid f(M_1, \dots, M_n) \mid k[M_1, \dots, M_n]$$

  $\mathsf{pencrypt}(c_0, \mathsf{pk}(sk_A))$.

- Properties $\rightsquigarrow$ facts

  $$F ::= \mathsf{attacker}(M).$$

- Protocol, attacker $\rightsquigarrow$ Horn clauses

  $$F_1 \wedge \dots \wedge F_n \rightarrow F$$

  $\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \rightarrow \mathsf{attacker}(\mathsf{pencrypt}(m, pk))$.

Example

Denning-Sacco key distribution protocol (simplified)

$$\begin{array}{lll} \text{Message 1.} & A \rightarrow B: & \{\{k\}_{sk_A}\}_{pk_B} \\ \text{Message 2.} & B \rightarrow A: & \{s\}_k \end{array}$$

The goal of the protocol is to keep s secret.

The attack

We find the (well-known) attack against this protocol.

$$
\begin{array}{lll}
\text{Message 1.} & A \to C : & \{\{k\}_{sk_A}\}_{pk_C} \\
\text{Message 1'.} & C(A) \to B : & \{\{k\}_{sk_A}\}_{pk_B} \\
\text{Message 2.} & B \to C(A) : & \{s\}_k
\end{array}
$$

The attacker C impersonates A and obtains the secret s.

Example - Cryptographic primitives

- Encryption $\mathsf{pencrypt}(m, pk)$.
  $\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \rightarrow \mathsf{attacker}(\mathsf{pencrypt}(m, pk))$

- Public key generation $\mathsf{pk}(sk)$.
  $\mathsf{attacker}(sk) \rightarrow \mathsf{attacker}(\mathsf{pk}(sk))$

- Decryption $\mathsf{pdecrypt}(\mathsf{pencrypt}(m, \mathsf{pk}(sk)), sk) = m$.
  $\mathsf{attacker}(\mathsf{pencrypt}(m, \mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \rightarrow \mathsf{attacker}(m)$

Names

Normally, fresh names are created each time the protocol is run. Here, we only distinguish two names when they are created after receiving different messages.

Each name k becomes a function of the messages previously received:

$$k[M_1, \ldots, M_n].$$

(Skolemisation)

These functions can only be applied by the principal that creates the name, not by the attacker.

Protocol

- $A \to B : \{\{k\}_{sk_A}\}_{pk_B}$ $\quad$ k fresh

  A talks with any principal represented by its public key $\mathsf{pk}(x)$.
  A sends to it the message $\{\{k\}_{sk_A}\}_{\mathsf{pk}(x)}$.

  $\mathsf{attacker}(\mathsf{pk}(x)) \to \mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(k[\mathsf{pk}(x)], sk_A[]), \mathsf{pk}(x)))$.

- $B \to A : \{s\}_k$

  B has received a message $\{\{y\}_{sk_A}\}_{pk_B}$.
  B sends $\{s\}_y$.

  $\mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(y, sk_A[]), \mathsf{pk}(sk_B[]))) \to \mathsf{attacker}(\mathsf{sencrypt}(s[], y))$.

General coding of a protocol

If a principal A has received the messages $M_1, \ldots, M_n$ and sends the message M,

$$\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \rightarrow \mathsf{attacker}(M).$$

Approximations

- The freshness of nonces is partially modeled.

- The number of times a message appears is ignored, only the fact that is has appeared is taken into account.

- The state of the principals is not fully modeled.

These approximations are keys for an efficient verification.

Solve the state space explosion problem.

No limit on the number of runs of the protocols.

$\Rightarrow$ essential for the certification of protocols.

Secrecy

If $\mathsf{attacker}(M)$ cannot be derived from the rules, M is secret.

The term M cannot be built by an attacker.
(This is a weaker notion of secrecy than non-interference.)

Goal: Determine whether a given fact can be derived from the rules.

Solving algorithm - First phase

First phase: completion of the rule base, by guarded resolution.

$$\frac{\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \rightarrow F \qquad F'_1 \wedge \ldots \wedge F'_{n'} \rightarrow F'}{\mathsf{attacker}(\sigma x_1) \wedge \ldots \wedge \mathsf{attacker}(\sigma x_n) \wedge \sigma F'_2 \wedge \ldots \wedge \sigma F'_{n'} \rightarrow \sigma F'}$$

where σ is the most general unifier of F and F'_1,
and F'_1 is not of the form $\mathsf{attacker}(x)$, x variable.

Perform this step until a fixed point is reached,
take only the rules $\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \rightarrow F$.

Solving algorithm - First phase

Example of a step:

$$\frac{\begin{array}{l}\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \rightarrow \mathsf{attacker}(\mathsf{pencrypt}(x, y)) \\ \mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(z, sk_A[]), \mathsf{pk}(sk_B[]))) \rightarrow \mathsf{attacker}(\mathsf{sencrypt}(s[], z))\end{array}}{\mathsf{attacker}(\mathsf{sign}(z, sk_A[])) \wedge \mathsf{attacker}(\mathsf{pk}(sk_B[])) \rightarrow \mathsf{attacker}(\mathsf{sencrypt}(s[], z))}$$

Main theorem:

Theorem 1 *The rules obtained after the first phase prove the same facts as the starting rules.*

Solving algorithm: Why it works

$$\frac{\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \rightarrow F \qquad F'_1 \wedge \ldots \wedge F'_{n'} \rightarrow F'}{\mathsf{attacker}(\sigma x_1) \wedge \ldots \wedge \mathsf{attacker}(\sigma x_n) \wedge \sigma F'_2 \wedge \ldots \wedge \sigma F'_{n'} \rightarrow \sigma F'}$$

The conditions "hypotheses $\mathsf{attacker}(x_i)$ and F'_1 not of the form $\mathsf{attacker}(x)$" strongly restrict the possible unifications:

- The rules with all facts (hypotheses and conclusion) of the form $\mathsf{attacker}(x)$ are useless.
- Then, F and F'_1 are not of the form $\mathsf{attacker}(x)$.
- Therefore, two non-variable terms must unify for a new rule to be created.

Solving algorithm - second phase

Second phase: backward depth-first search.

This is the usual Prolog algorithm, except that we detect cycles.

Technique similar to the ordered resolution with selection (Weidenbach, 1999).

Termination

- The first phase does not always terminate.

  Termination can be enforced by limiting the depth of terms.
  (Terms deeper than the limit are replaced with variables.)

  This yields a loss of precision, but the result is still correct.

- When searching a closed fact,
  the search of the second phase terminates.
  (The hypotheses of rules are smaller than the conclusion.)

Optimizations and extensions

Optimizations:

- Tuples
- Secrecy assumptions: use conjectures to prune the search space.

Extensions:

- Diffie-Hellman key agreement
- Model the compromise of some session keys

Experimental results

Pentium MMX 233 MHz.

Protocol	Result	s
Needham-Schroeder public key	Attack [Lowe]	0.07
Needham-Schroeder public key corrected	Secure	0.06
Needham-Schroeder shared key	Attack [Denning]	0.76
Needham-Schroeder shared key corrected	Secure	1.19
Denning-Sacco	Attack [BAN]	0.04
Denning-Sacco corrected	Secure	0.04
Otway-Rees	Secure	0.27
Otway-Rees, variant of Paulson98	Attack [Paulson]	0.26
Yahalom	Secure	0.11
Simpler Yahalom	Secure	0.31
Main mode of Skeme	Secure	23.07

Conclusion

- A particularly efficient verifier
- Can handle complex protocols
- No limit on the number of runs of the protocol
- Can be used for certification of protocols

Further work

- Other properties (authenticity, ...).
- Other cryptographic primitives (xor, ...).
- Termination.