An Automatic Security Protocol Verifier based on Resolution Theorem Proving

Bruno Blanchet

CNRS, Département d'Informatique,
École Normale Supérieure

July 2005

Introduction

Many techniques exist for the verification of cryptographic protocols (theorem proving, model checking, typing, . . .)

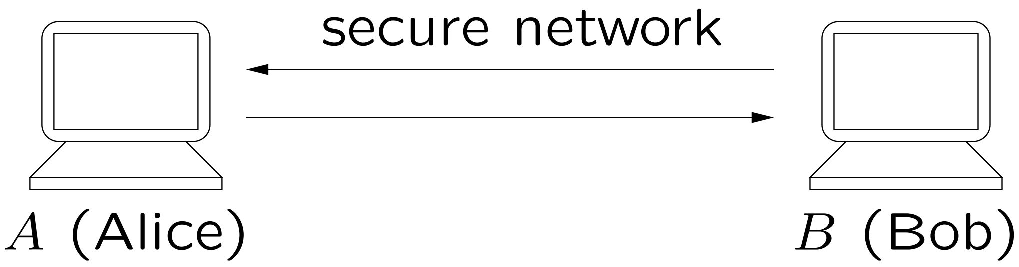
We present a technique based on an abstract representation of the protocol by a set of Horn clauses.

This technique yields fully automatic proofs of protocols for an unbounded number of runs (many clients connecting to a server, for instance) and an unbounded message space.

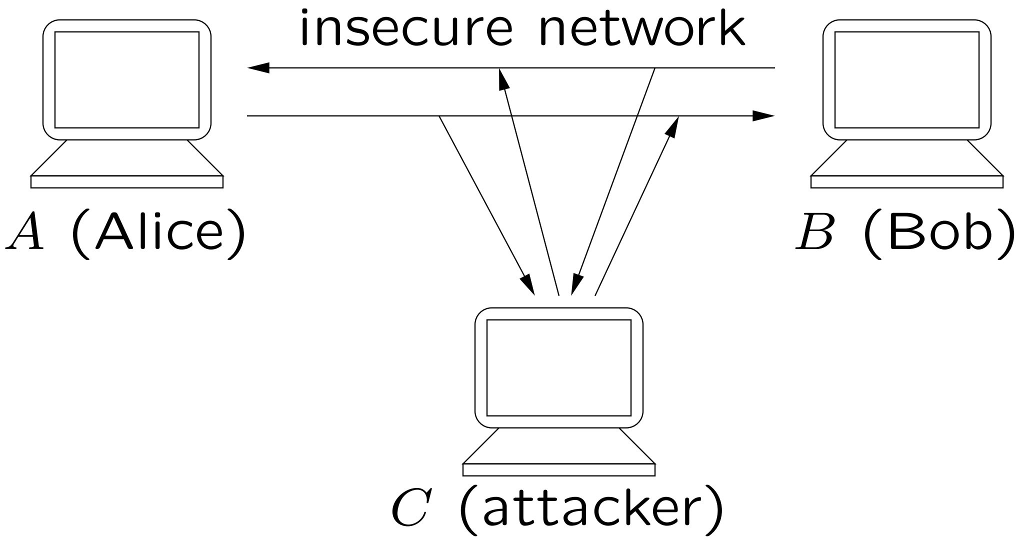
Outline

- Introduction to cryptographic protocols
- ProVerif and the Horn clause verification method
- Demo and experimental results
- Conclusion and future work

Communications on a secure network



Communications on an insecure network



A talks to B on an insecure network
$\Rightarrow$ encrypt messages to preserve the secrets.

Encryption

The main cryptographic primitive is encryption:

$$\{m\}_k$$

is the encryption of message m under key k.

When you have the decryption key, you can get m from $\{m\}_k$.

- Shared key encryption: the decryption key is equal to the encryption key.
- Public key encryption: the decryption key (secret key sk) is different from the encryption key (public key pk).
- Signature: one signs with the secret key sk ($\{m\}_{sk}$), and checks the signature with the public key pk.

Dolev-Yao model

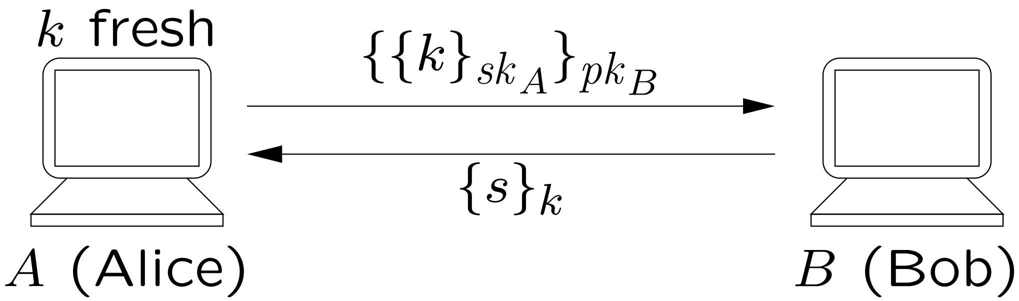
We use the so-called Dolev-Yao model:

- the attacker can get all messages sent on the network
- he can compute messages
- he can send messages on the network

We assume perfect cryptography: one can decrypt only if one has the decryption key.

Example

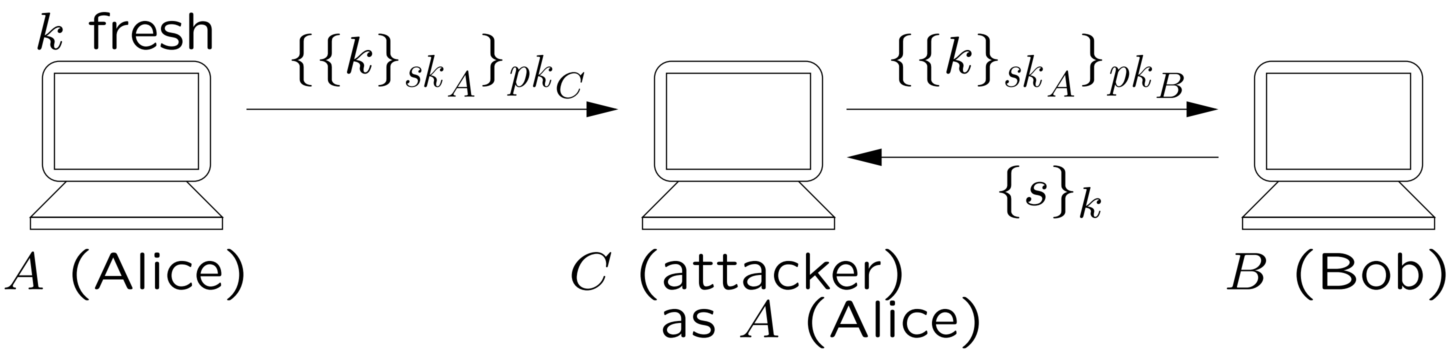
Denning-Sacco key distribution protocol [Denning, Sacco, Comm. ACM, 1981] (simplified)



The goal of the protocol is that the key k should be a secret key, shared between A and B. So s should remain secret.

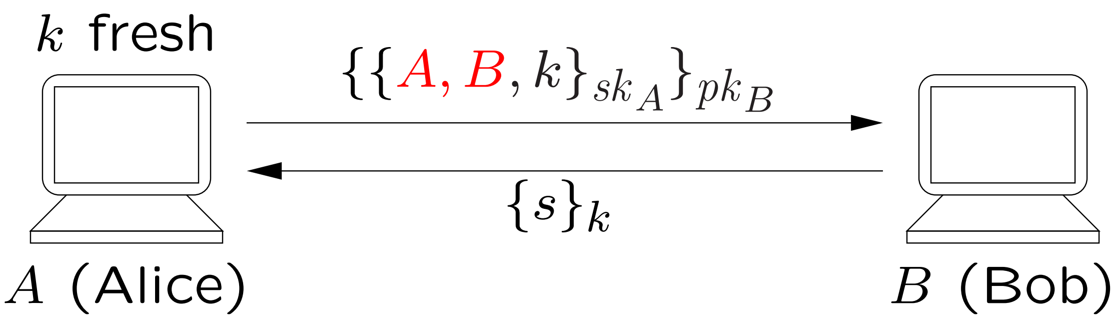
The attack

The (well-known) attack against this protocol.



The attacker C impersonates A and obtains the secret s.

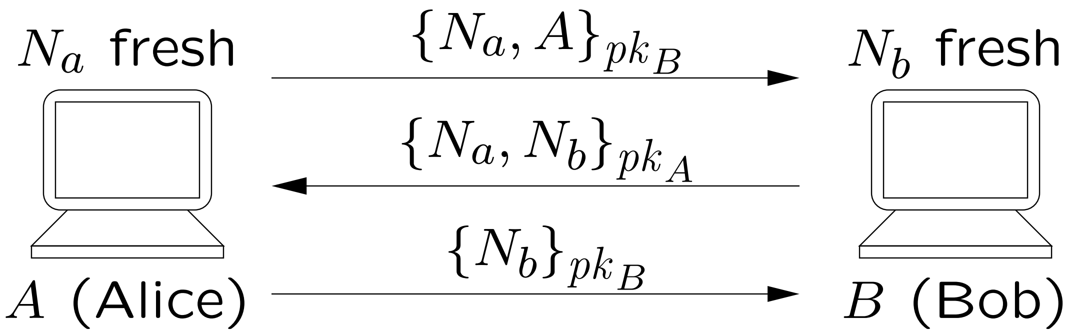
The corrected protocol



Now C cannot impersonate A because in the previous attack, the first message is $\{\{A, C, k\}_{sk_A}\}_{pk_B}$, which is not accepted by B.

Another example

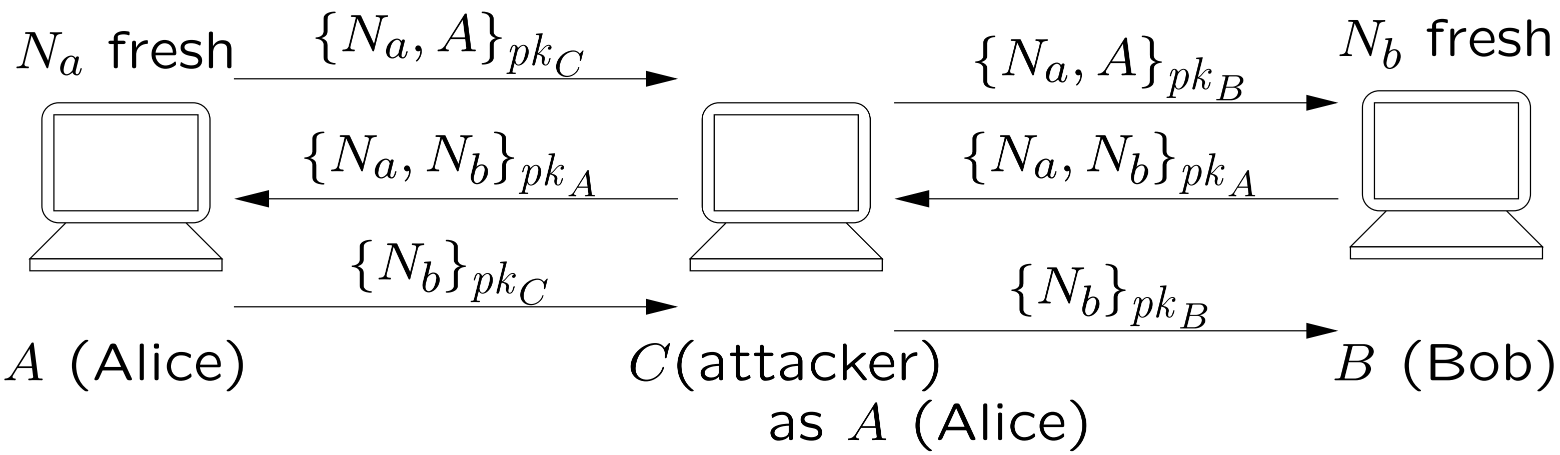
Needham-Schroeder public key protocol [Needham, Schroeder, Comm. ACM, 1978]



N_a and N_b should be shared secrets between A and B. At the end of the protocol, B should be convinced that he talked to A and conversely.

The attack

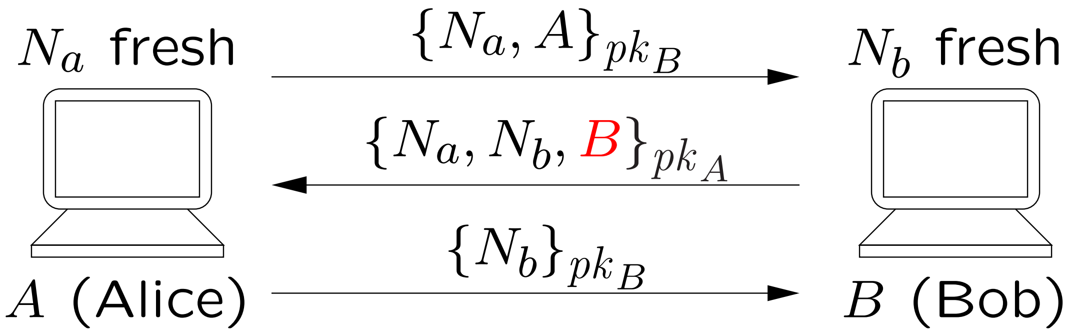
Lowe [Lowe, TACAS'96] found a man-in-the-middle attack against the Needham-Schroeder protocol:



C impersonates A, and obtains N_a and N_b. B thinks he talks to A, whereas in fact he talks to C.

The corrected protocol

Lowe suggested the following corrected version:



We can check that it prevents the previous attack.

Practical examples

- SSH (Secure SHell)
- SSL (Secure Socket Layer), renamed TLS (Transport Layer Security), `https://`
- IPsec
- SET (Secure Electronic Transaction), e-commerce protocol.

Why cryptographic protocols ?

- Their design is error prone.

- Errors are not detected by testing:
  they appear only in the presence of an adversary.

- Errors can have serious consequences.

Goals of protocols

- Secrecy: The attacker cannot have a message s.

- Authentication: If B thinks he talks to A then A thinks she talks to B.

- Key exchange: establish a shared secret key between A and B.

- Fair contract signing.

- ...

Difficulties of protocol verification

Cryptographic protocols are infinite state:

- The attacker can create messages of unbounded size.
- Unbounded number of sessions of the protocol.

With the assumption of a bounded number of sessions, protocols are still decidable.

They are undecidable in the general case.

Solutions to undecidability

To solve an undecidable problem, we can

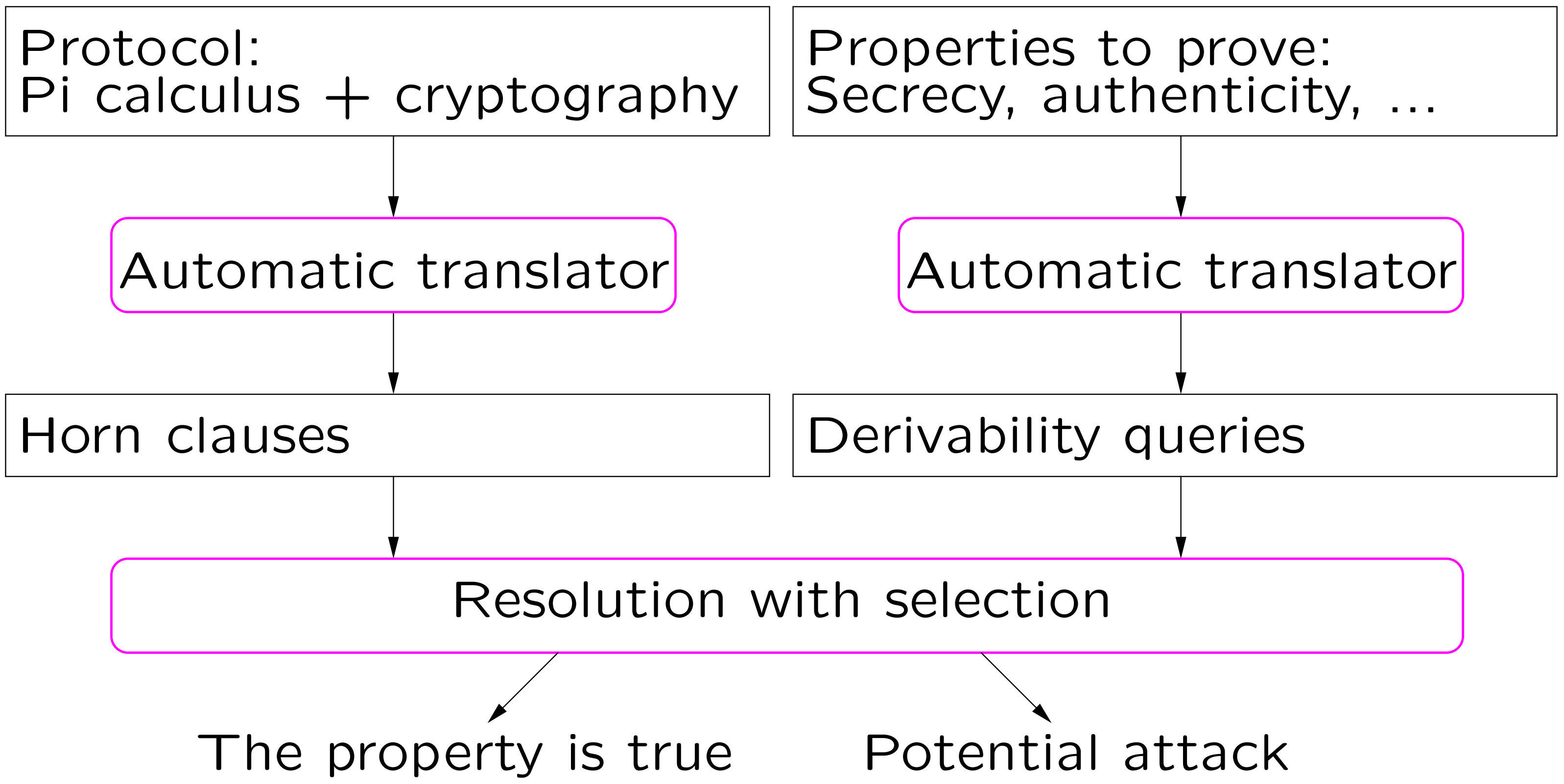
- Use approximations, abstraction.
- Terminate on a restricted class.

We do both, using a very precise abstraction.

Outline

- Introduction to cryptographic protocols
- ProVerif and the Horn clause verification method
- Demo and experimental results
- Conclusion and future work

The Horn clause verification method



Features of the Horn clause verification method

- Fully automatic.

- Efficient: small examples verified in less than 0.1 s; complex ones in a few minutes.

- Very precise: no false attack in our tests for secrecy and authentication.

- For unbounded number of sessions and message space.

- For a wide range of cryptographic primitives.

Syntax of the process calculus

Pi calculus + cryptographic primitives

$M, N ::=$	terms
$\quad x, y, z$	variable
$\quad a, b, c, k, s$	name
$\quad f(M_1, \ldots, M_n)$	constructor application
$P, Q ::=$	processes
$\quad \overline{M}\langle N \rangle.P$	output
$\quad M(x).P$	input
$\quad \mathit{let}\ x = g(M_1, \ldots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q$	destructor application
$\quad \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$	conditional
$\quad 0 \quad P \mid Q \quad !P \quad (\nu a)P$	

Constructors and destructors

Two kinds of operations:

- Constructors f are used to build terms
  $f(M_1, \ldots, M_n)$

- Destructors g manipulate terms
  $let\ x = g(M_1, \ldots, M_n)\ in\ P\ else\ Q$
  Destructors are defined by rewrite rules $g(M_1, \ldots, M_n) \rightarrow M$.

Examples of constructors and destructors

Shared-key encryption:

- Constructor: Shared-key encryption $\mathsf{encrypt}(M, N)$.
- Destructor: Decryption $\mathsf{decrypt}(M', N)$

$$\mathsf{decrypt}(\mathsf{encrypt}(M, N), N) \rightarrow M.$$

Examples of constructors and destructors (continued)

Signature:

- Constructors: Signature $\mathsf{sign}(M, N)$.
  Public key generation $\mathsf{pk}(N)$.

- Destructors: Signature checking $\mathsf{checksign}(M', N')$

$$\mathsf{checksign}(\mathsf{sign}(M, N), \mathsf{pk}(N)) \rightarrow M.$$

  Message extraction $\mathsf{getmess}(M')$

$$\mathsf{getmess}(\mathsf{sign}(M, N)) \rightarrow M.$$

We represent in the same way public-key encryption, hash functions, ...

Example: The Denning-Sacco protocol

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

$$\begin{array}{lll} & & (\nu sk_A)(\nu sk_B) let\ pk_A = \mathsf{pk}(sk_A)\ in\ let\ pk_B = \mathsf{pk}(sk_B)\ in \\ & & \overline{c}\langle pk_A\rangle\overline{c}\langle pk_B\rangle. \\ (A) & & !\, c(x\_pk_B).(\nu k)\overline{c}\langle \mathsf{pencrypt}(\mathsf{sign}(k, sk_A), x\_pk_B)\rangle. \\ & & \quad c(x).let\ s = \mathsf{decrypt}(x, k)\ in\ 0 \\ (B) & \mid & !\, c(y).let\ y' = \mathsf{pdecrypt}(y, sk_B)\ in \\ & & \quad let\ k = \mathsf{checksign}(y', pk_A)\ in\ \overline{c}\langle \mathsf{encrypt}(\mathsf{s}, k)\rangle \end{array}$$

The Horn clause representation

The main predicate used by the Horn clause representation of protocols is attacker:

$$\mathsf{attacker}(M) \quad \text{means} \quad \text{``the attacker may have } M\text{''}.$$

We can model actions of the adversary and of the protocol participants thanks to this predicate.

Processes are automatically translated into Horn clauses (joint work with Martín Abadi).

Example - Cryptographic primitives

Shared-key encryption:

- Encryption $\mathsf{encrypt}(m, k)$.
  $\mathsf{attacker}(m) \wedge \mathsf{attacker}(k) \longrightarrow \mathsf{attacker}(\mathsf{encrypt}(m, k))$

- Decryption $\mathsf{decrypt}(\mathsf{encrypt}(m, k)), k) \rightarrow m$.
  $\mathsf{attacker}(\mathsf{encrypt}(m, k)) \wedge \mathsf{attacker}(k) \longrightarrow \mathsf{attacker}(m)$

General treatment of primitives

- Constructors $f(M_1, \ldots, M_n)$
  $\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \longrightarrow \mathsf{attacker}(f(x_1, \ldots, x_n))$

- Destructors $g(M_1, \ldots, M_n) \longrightarrow M$
  $\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \longrightarrow \mathsf{attacker}(M)$

Example - Coding of a protocol

Upon receipt of a message of the form $\mathsf{pencrypt}(\mathsf{sign}(y, sk_A), pk_B)$, B replies with $\mathsf{encrypt}(\mathsf{s}, y)$:

$$\mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(y, sk_A), pk_B)) \longrightarrow \mathsf{attacker}(\mathsf{encrypt}(\mathsf{s}, y))$$

The attacker sends $\mathsf{pencrypt}(\mathsf{sign}(y, sk_A), pk_B)$ to B, and intercepts his reply $\mathsf{encrypt}(\mathsf{s}, y)$.

General coding of a protocol

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

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

Sound approximations

- The freshness of keys, nonces is partially modeled.
- The state of principals is partially modeled.
- The number of times a message appears is ignored, only the fact that is has appeared is taken into account.

These approximations help for efficient verification
with infinite state spaces,
with any number of protocol runs.

Security properties

- Secrecy:
  If $\mathsf{attacker}(M)$ cannot be derived from the clauses, then M is secret.

  The term M cannot be built by an attacker.

  The resolution algorithm will determine whether a given fact can be derived from the clauses.

  Remark: Soundness and completeness are swapped.
  The resolution prover is complete
  (If $\mathsf{attacker}(M)$ is derivable, it finds a derivation.)
  $\Rightarrow$ The protocol verifier is sound
  (If it proves secrecy, then secrecy is true.)

Security properties (extensions)

- Correspondence assertions:
  ProVerif can also prove properties of the form:
  - If an event e has happened, then some other events $e_1, \ldots, e_n$ must have happened.
  - For each execution of event e, *distinct* events $e_1, \ldots, e_n$ have been executed.

  These properties formalize in particular authentication.

Security properties (extensions)

- Process equivalences

  $P \approx Q$ when the adversary cannot distinguish P from Q.

  Process equivalences can formalize various security properties. Example: P implements an ideal specification Q.

  ProVerif can check:

  - Strong secrecy: $P(x) \approx P(y)$ for all x, y.
  - Equivalences between processes that differ only by terms they contain (joint work with Martín Abadi and Cédric Fournet)

Determining derivability

The natural tool for determining derivability of a fact from clauses is resolution.

Basic SLD-resolution (as in Prolog) would never terminate:

$$\mathsf{attacker}(\mathsf{encrypt}(m, k)) \wedge \mathsf{attacker}(k) \rightarrow \mathsf{attacker}(m)$$

yields a loop.

Resolution with free selection

$$\frac{R = H \rightarrow F \qquad R' = F'_1 \wedge H' \rightarrow F'}{\sigma H \wedge \sigma H' \rightarrow \sigma F'}$$

where σ is the most general unifier of F and F'_1,
F and F'_1 are selected.

The selection function selects:
– a hypothesis not of the form $\mathsf{attacker}(x)$ if possible,
– the conclusion otherwise.

Key idea: avoid resolving on facts $\mathsf{attacker}(x)$.

Termination (joint work with Andreas Podelski)

The technique does not always terminate,
but terminates on tagged protocols.

Each cryptographic primitive application is identified by a constant tag:

$$\mathsf{encrypt}((\mathsf{c}_0, M), K)$$

- Simple syntactic annotation.
- Retains the intended behavior of the protocol.
- Prevents type flaw attacks.
- Good design practice. Used in real protocols (SSH).

Other possible variants

- Ordered resolution with selection [Weidenbach, 1999].

- Ordered resolution with factorization and splitting [Comon, Cortier, 2003]
  Terminates on clauses with at most one variable.
  Protocols which blindly copy at most one term.

Reconstruction of attacks
(joint work with Xavier Allamigeon)

In general, a derivation of $\mathsf{attacker}(M)$ manifests an attack against the secrecy of M.

We can reconstruct such an attack from the derivation:
we explore the traces that correspond to the derivation.

The attack reconstruction fails when the derivation does not correspond to an attack (false attack).

Outline

- Introduction to cryptographic protocols
- ProVerif and the Horn clause verification method
- Demo and experimental results
- Conclusion and future work

Experimental results

Pentium M, 1.8 GHz.

Protocol	Result	Time
Simple protocols of the literature:		ms
Needham-Schroeder public key	Attack [Lowe]	82
Needham-Schroeder public key corrected	Secure	53
Needham-Schroeder shared key	Attack [Denning]	226
Needham-Schroeder shared key corrected	Secure	109
Denning-Sacco	Attack [AN]	19
Denning-Sacco corrected	Secure	16
Otway-Rees	Secure	66
Otway-Rees, variant of Paulson, 98	Attack [Paulson]	112
Yahalom	Secure	40
Main mode of Skeme	Secure	191
More complex case studies:		
Certified email (with M. Abadi)	Secure	2 s
Certified email + SSH (with M. Abadi)	Secure	6 min
JFK (with M. Abadi and C. Fournet)	Secure	96 s

Outline

- Introduction to cryptographic protocols
- ProVerif and the Horn clause verification method
- Demo and experimental results
- Conclusion and future work

Conclusion

- A lot of effort (and a lot of progress!) on the verification of cryptographic protocols.

- We can now have fully automatic proofs of cryptographic protocols for an unbounded number of sessions.

- Logic is a very powerful tool to represent systems formally.

Other aspects of cryptographic protocol verification

- Model more primitives, using equations
- Model more features: timestamps, ...
  (for example, Delzanno and Ganty, TACAS'04)
- Prove more properties automatically: denial of service, ...
- Proofs of protocols in the computational model of cryptography (studied by many people)
- Proofs of implementations of protocols (in C, Java, ...)
  (started by Goubault-Larrecq and Parennes, VMCAI'05)