From Secrecy to Authenticity in Cryptographic Protocols

Bruno Blanchet
Département d'Informatique,
École Normale Supérieure
and
Max-Planck-Institut für Informatik

October 2002

Introduction

Verifier for security properties of cryptographic protocols

- Fully automatic
- For an unbounded number of sessions
- Efficient (no state space explosion)

In this talk, we focus on authenticity.

Informal definition of authenticity

Authenticity means:

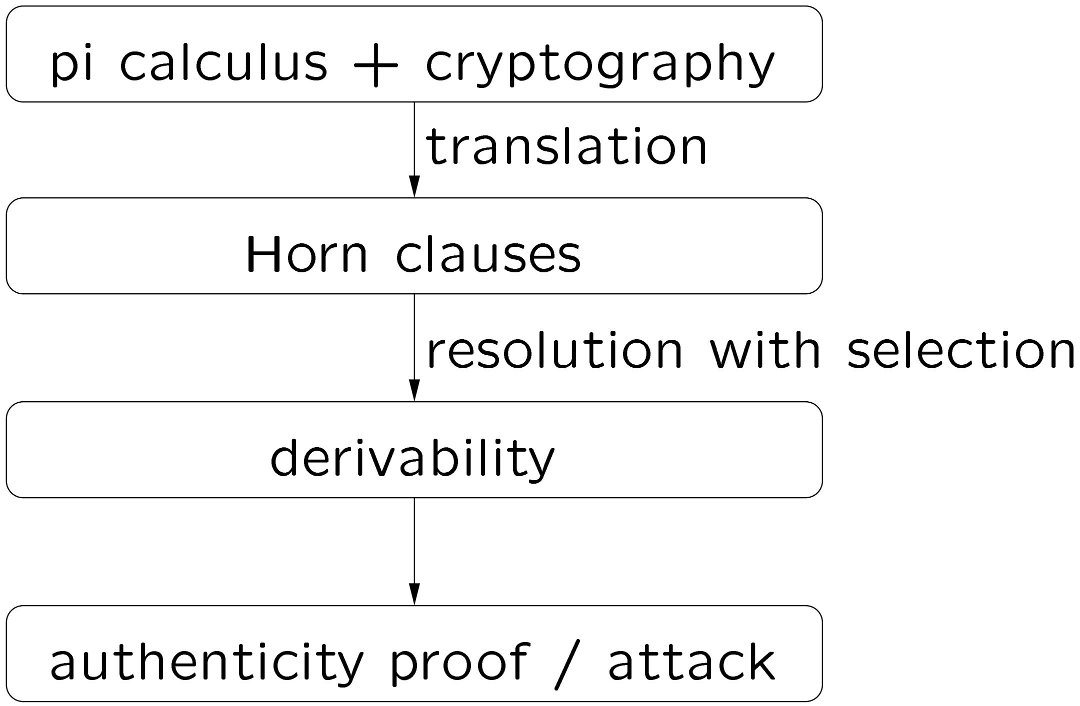
if A thinks he talks to B
then he really talks to B.

Authenticity can be defined by correspondence assertions
(Woo and Lam):

If A executes `end` (A thinks he talks to B),
then B must have executed `begin` (B has started a run with A).

Our solution

Extend a previous technique for secrecy.



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
$\quad 0 \quad P \mid Q \quad !P \quad (\nu a)P$	
$\quad \mathit{begin}(M).P$	begin event
$\quad \mathit{end}(M).P$	end event

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 reductions $g(M_1, \ldots, M_n) \rightarrow M$.

Examples of constructors and destructors

Public-key encryption:

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

- Destructor: Decryption $\mathsf{pdecrypt}(M', N)$

$$\mathsf{pdecrypt}(\mathsf{pencrypt}(M, \mathsf{pk}(N)), N) \rightarrow M.$$

We can also handle shared-key encryption, signatures, hash functions, a simple model of Diffie-Hellman key agreements, ...

Process calculus (continued)

To remember executed events, we add $begin\_ex(M)$ and $end\_ex(M)$:

$$begin(M).P \rightarrow begin\_ex(M) \mid P$$
$$end(M).P \rightarrow end\_ex(M) \mid P$$

Executed events do not appear in the initial process.
A process without executed events is a starting process.

Formal definition of secrecy

S = finite set of names (initial knowledge of the adversary).

The closed process Q is an S-adversary $\Leftrightarrow$ $\mathit{fn}(Q) \subseteq S$ and Q does not contain events.

$\boxed{\text{The secret } M \text{ cannot be output on a public channel}}$

The process P outputs M on c $\Leftrightarrow$
$P \, (\rightarrow \cup \equiv)^* \, \overline{c}\langle M \rangle.R \mid R'$ for some processes R and R'.

The closed process P preserves the secrecy of M from S $\Leftrightarrow$
$\forall S$-adversary $Q, \forall c \in S, P \mid Q$ does not output M on c.

Formal definition of authenticity

S = finite set of names (initial knowledge of the adversary).

The closed process Q is an S-adversary $\Leftrightarrow$ $fn(Q) \subseteq S$ and Q does not contain events.

$\boxed{\text{If } end(M) \text{ has been executed then } begin(M) \text{ has been executed}}$

The closed starting process P satisfies
non-injective agreement with respect to S-adversaries $\Leftrightarrow$
$\forall S$-adversary Q, $\forall P'$ such that $P \mid Q \, (\rightarrow \cup \equiv)^* \, P'$, $\forall M$,
if $end\_ex(M)$ occurs in P', then $begin\_ex(M)$ also occurs in P'.

Formal definition of authenticity

S = finite set of names (initial knowledge of the adversary).

The closed process Q is an S-adversary $\Leftrightarrow$ $fn(Q) \subseteq S$ and Q does not contain events.

$\boxed{\text{If } end(M) \text{ has been executed } n \text{ times then } begin(M) \text{ has been executed at least } n \text{ times}}$

The closed starting process P satisfies
injective agreement with respect to S-adversaries $\Leftrightarrow$
$\forall S$-adversary Q, $\forall P'$ such that $P \mid Q \, (\rightarrow \cup \equiv)^* \, P'$, $\forall M$,
the number of occurrences of $end\_ex(M)$ in P' is at most the number of occurrences of $begin\_ex(M)$ in P'.

Example (simplified Woo-Lam public key)

$$\begin{array}{llll} \text{Message 1.} & A \rightarrow B: & pk_A & \\ \text{Message 2.} & B \rightarrow A: & b & b \text{ fresh} \\ \text{Message 3.} & A \rightarrow B: & \{pk_A, pk_B, b\}_{sk_A} & \end{array}$$

$$(\nu sk_A)(\nu sk_B) let\ pk_A = \mathsf{pk}(sk_A)\ in\ let\ pk_B = \mathsf{pk}(sk_B)\ in$$
$$\overline{c}\langle pk_A\rangle\overline{c}\langle pk_B\rangle.$$

$$\begin{array}{lll} (A) & & !\ c(x\_pk_B).begin(x\_pk_B).\overline{c}\langle pk_A\rangle.c(x\_b). \\ & & \quad \overline{c}\langle \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A)\rangle \\ (B) & \mid & !\ c(x\_pk_A).(\nu b)\overline{c}\langle b\rangle.c(m). \\ & & \quad if\ (x\_pk_A, pk_B, b) = \mathsf{checksign}(m, x\_pk_A)\ then \\ & & \quad if\ x\_pk_A = pk_A\ then\ end(pk_B) \end{array}$$

Overview of the proof technique

Our technique overapproximates occurrences of events.

We can overapproximate occurrences of end events,
but not of $begin$ events.

We assume that $begin(M)$ did not occur,
and we show that $end(M)$ cannot not occur.

Back to the example

Forbid the execution of $begin(pk_B)$:

$$
\begin{array}{lll}
 & & (\nu sk_A)(\nu sk_B) let\ pk_A = \mathsf{pk}(sk_A)\ in\ let\ pk_B = \mathsf{pk}(sk_B)\ in \\
 & & \overline{c}\langle pk_A\rangle \overline{c}\langle pk_B\rangle. \\
(A) & & !\ c(x\_pk_B).if\ x\_pk_B = pk_B\ then\ 0\ else\ \overline{c}\langle pk_A\rangle.c(x\_b). \\
 & & \quad \overline{c}\langle \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A)\rangle \\
(B) & | & !\ c(x\_pk_A).(\nu b)\overline{c}\langle b\rangle.c(m). \\
 & & \quad if\ (x\_pk_A, pk_B, b) = \mathsf{checksign}(m, x\_pk_A)\ then \\
 & & \quad if\ x\_pk_A = pk_A\ then\ end(pk_B)
\end{array}
$$

... and show that $end(pk_B)$ cannot be executed.

Back to the example

Forbid the execution of $begin(pk_B)$:

$$
\begin{array}{lll}
& & (\nu sk_A)(\nu sk_B) let\ pk_A = \mathsf{pk}(sk_A)\ in\ let\ pk_B = \mathsf{pk}(sk_B)\ in \\
& & \overline{c}\langle pk_A\rangle \overline{c}\langle pk_B\rangle. \\
(A) & & !\ c(x\_pk_B).if\ \ x\_pk_B = pk_B\ then\ 0\ else\ \overline{c}\langle pk_A\rangle.c(x\_b). \\
& & \quad \overline{c}\langle \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A)\rangle \\
(B) & | & !\ c(x\_pk_A).(\nu b)\overline{c}\langle b\rangle.c(m). \\
& & \quad if\ \ (x\_pk_A, pk_B, b) = \mathsf{checksign}(m, x\_pk_A)\ then \\
& & \quad if\ \ x\_pk_A = pk_A\ then\ \overline{c}\langle auth(pk_B)\rangle
\end{array}
$$

... and show that $end(pk_B)$ cannot be executed.
that is, show that $auth(pk_B)$ is kept secret.

A proof of authenticity from secrecy

Assume that no bound names occur in events.

Let $P = C[\mathit{begin}(M).P_1, \mathit{end}(M').P_2]$ be a closed starting process, and assume that these are the only begin and end events in P.

Let $P'(N) = C[\mathit{if}\ M = N\ \mathit{then}\ 0\ \mathit{else}\ P_1, \overline{c}\langle \mathit{auth}(M')\rangle.P_2]$.

Assume that for all closed terms N, $P'(N)$ preserves the secrecy of $\mathit{auth}(N)$ from S (with $c \in S$).
Then P satisfies non-injective agreement with respect to S-adversaries.

How to handle the general case

When events contain bound names:

- Bound names in N may not be in scope at the $begin(M)$.
  $\Rightarrow$ we cannot write $if\ M = N$ instead of $begin(M)$
  $\Rightarrow$ represent names by functions.

- Names created in different sessions must always be distinguished, otherwise the analysis thinks that the $else$ branch of the test may always be executed.
  $\Rightarrow$ use session identifiers

- For injective agreement, relate corresponding sessions of A and B $\Rightarrow$ use session identifiers

Instrumentation

- Add session identifiers to replications:

$$!P \text{ becomes } !^{i \geq n} P$$

$!^{i \geq n} P$ means $P\{n/i\} \mid P\{n+1/i\} \mid P\{n+2/i\} \mid \ldots$.

- Add patterns (types) to restrictions:

$$(\nu a)P \text{ becomes } (\nu a{:}p)P$$

Fresh name $a \rightsquigarrow$ function $p = a[x_1, \ldots, x_n, i_1, \ldots, i_{n'}]$ of all variables (in particular inputs and sessions identifiers) bound above a (skolemization).

$\rightarrow$ distinguish bound names created in different sessions.

Instrumentation of the example

$$(\nu sk_A{:}sk_A[])(\nu sk_B{:}sk_B[]) let\ pk_A = \mathsf{pk}(sk_A)\ in\ let\ pk_B = \mathsf{pk}(sk_B)\ in$$
$$\overline{c}\langle pk_A\rangle \overline{c}\langle pk_B\rangle.$$

$$(A) \qquad !^{i_A \geq 0}\ c(x\_pk_B).begin(x\_pk_B).\overline{c}\langle pk_A\rangle.c(x\_b).$$
$$\overline{c}\langle \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A)\rangle$$

$$(B) \qquad |\quad !^{i_B \geq 0}\ c(x\_pk_A).(\nu b{:}b[x\_pk_A, i_B])\overline{c}\langle b\rangle.c(m).$$
$$if\ \ (x\_pk_A, pk_B, b) = \mathsf{checksign}(m, x\_pk_A)\ \ then$$
$$if\ \ x\_pk_A = pk_A\ \ then\ \ end(pk_B)$$

Translation pi + crypto $\rightarrow$ Horn clauses

A protocol is translated into a set of Horn clauses using 4 predicates:

$\mathsf{mess}(p, p')$	the message p' may be sent on the channel p
$\mathsf{attacker}(p)$	the adversary may have p
$\mathsf{begin}(p)$	the event $begin(p)$ may have happened
$\mathsf{end}(p)$	the event $end(p)$ may have happened

$F_1 \wedge \ldots \wedge F_n \rightarrow F$ means that, if F_1 and ... and F_n are true, then F is true.

Translation: attacker rules

For each $a \in S$, $\mathsf{attacker}(a[])$ (Init)

$\mathsf{attacker}(b[i])$ where $b \notin fn(P_0)$ and (νb) does not occur in P_0

(Name gen)

For each constructor f of arity n,

$\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \rightarrow \mathsf{attacker}(f(x_1, \ldots, x_n))$ (Constr)

For each destructor g, for each reduction $g(M_1, \ldots, M_n) \rightarrow M$ in $\mathrm{def}(g)$,

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

(Destr)

$\mathsf{mess}(x, y) \wedge \mathsf{attacker}(x) \rightarrow \mathsf{attacker}(y)$ (Listen)

$\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \rightarrow \mathsf{mess}(x, y)$ (Send)

Translation: protocol rules

ρ: environment (variables, names $\mapsto$ terms)
h: hypothesis (events that must be executed before reaching the current process)

- $[\![M(x).P]\!]\rho h = [\![P]\!](\rho[x \mapsto x'])(h \wedge \mathsf{mess}(\rho(M), x'))$
  x' new variable

- $[\![\overline{M}\langle N\rangle.P]\!]\rho h = [\![P]\!]\rho h \cup \{h \rightarrow \mathsf{mess}(\rho(M), \rho(N))\}$

- $[\![\mathit{begin}(M).P]\!]\rho h = [\![P]\!]\rho(h \wedge \mathsf{begin}(\rho(M)))$

- $[\![\mathit{end}(M).P]\!]\rho h = [\![P]\!]\rho h \cup \{h \rightarrow \mathsf{end}(\rho(M))\}$

- $[\![!^{i \geq 0} P]\!]\rho h = [\![P]\!](\rho[i \mapsto i'])h$ $\quad$ i' new variable

- $[\![(\nu a : p)P]\!]\rho h = [\![P]\!](\rho[a \mapsto \rho(p)])h$

Example: protocol rules (initialization)

Process:

$(\nu sk_A : sk_A[])(\nu sk_B : sk_B[]) let\ pk_A = \mathsf{pk}(sk_A)\ in\ let\ pk_B = \mathsf{pk}(sk_B)\ in$
$\overline{c}\langle pk_A\rangle.\overline{c}\langle pk_B\rangle \ldots$

Rules:

$$\mathsf{attacker}(\mathsf{pk}(sk_A[]))$$

$$\mathsf{attacker}(\mathsf{pk}(sk_B[]))$$

Note: $\mathsf{mess}(c, M)$ is equivalent to $\mathsf{attacker}(M)$ because c is public

Example: protocol rules (for A)

Process:

$$(A) \qquad !^{i_A \geq 0}\, c(x\_pk_B).begin(x\_pk_B).\overline{c}\langle pk_A\rangle.c(x\_b).$$
$$\overline{c}\langle \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A)\rangle$$

Rules:

$$\mathsf{attacker}(x\_pk_B) \wedge \mathsf{begin}(x\_pk_B) \rightarrow \mathsf{attacker}(\mathsf{pk}(sk_A[]))$$

Example: protocol rules (for A)

Process:

$$(A) \qquad !^{i_A \geq 0}\ c(x\_pk_B).begin(x\_pk_B).\overline{c}\langle pk_A\rangle.c(x\_b).$$
$$\overline{c}\langle \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A)\rangle$$

Rules:

$$\mathsf{attacker}(x\_pk_B) \wedge \mathsf{begin}(x\_pk_B) \rightarrow \mathsf{attacker}(\mathsf{pk}(sk_A[]))$$

$$\mathsf{attacker}(x\_pk_B) \wedge \mathsf{begin}(x\_pk_B) \wedge \mathsf{attacker}(x\_b)$$
$$\rightarrow \mathsf{attacker}(\mathsf{sign}((\mathsf{pk}(sk_A[]), x\_pk_B, x\_b), sk_A[]))$$

Example: protocol rules (for B)

Process:

$$(B) \qquad !^{i_B \geq 0}\, c(x\_pk_A).(\nu b : b[x\_pk_A, i_B])\overline{c}\langle b\rangle.c(m).$$
$$\quad \mathit{if}\ \ (x\_pk_A, pk_B, b) = \mathsf{checksign}(m, x\_pk_A)\ \ \mathit{then}$$
$$\quad \mathit{if}\ \ x\_pk_A = pk_A\ \ \mathit{then}\ \ \mathit{end}(pk_B)$$

Rules:

$\mathsf{attacker}(x\_pk_A) \rightarrow \mathsf{attacker}(b[x\_pk_A, i_B])$

Example: protocol rules (for B)

Process:

$$(B) \qquad !^{i_B \geq 0}\, c(x\_pk_A).(\nu b : b[x\_pk_A, i_B])\overline{c}\langle b\rangle.c(m).$$
$$\quad \mathit{if}\ \ (x\_pk_A, pk_B, b) = \mathsf{checksign}(m, x\_pk_A)\ \ \mathit{then}$$
$$\quad \mathit{if}\ \ x\_pk_A = pk_A\ \ \mathit{then}\ \ \mathit{end}(pk_B)$$

Rules:

$\mathsf{attacker}(x\_pk_A) \rightarrow \mathsf{attacker}(b[x\_pk_A, i_B])$

$\mathsf{attacker}(\mathsf{pk}(sk_A[])) \wedge \mathsf{attacker}(\mathsf{sign}((\mathsf{pk}(sk_A[]), \mathsf{pk}(sk_B[]), b[\mathsf{pk}(sk_A[]), i_B]), sk_A[]))$
$\quad \rightarrow \mathsf{end}(\mathsf{pk}(sk_B[]))$

Proof of authenticity

Rules for the protocol and the adversary: $B_{P_0,S}$.
Closed facts defining begin: B_b.

Theorem:

Assume that, for any B_b,

if $\mathsf{end}(T)$ is derivable from $B_{P_0,S} \cup B_b$, then $\mathsf{begin}(T) \in B_b$.

Then P_0 satisfies non-injective agreement with respect to S-adversaries.

Solving algorithm (for secrecy)

Selection function $sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) \subseteq \{F_1, \ldots, F_n\}$.

$$sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) = \begin{cases} \emptyset \text{ if } \forall i \in \{1, \ldots, n\}, F_i = \mathsf{attacker}(x) \\ \{F_i\} \text{ or } \emptyset \text{ different from } \mathsf{attacker}(x) \end{cases}$$

Resolution

$$\frac{R = F_1 \wedge \ldots \wedge F_n \rightarrow F \qquad R' = F'_1 \wedge \ldots \wedge F'_{n'} \rightarrow F'}{\sigma F_1 \wedge \ldots \wedge \sigma F_n \wedge \sigma F'_2 \wedge \ldots \wedge \sigma F'_{n'} \rightarrow \sigma F'}$$

where σ is the most general unifier of F and F'_1,
$sel(R) = \emptyset$, and $F'_1 \in sel(R')$.
until a fixpoint is reached.
saturate($B_{P_0,S}$) $= \{R \mid sel(R) = \emptyset\}$.

Solving algorithm (for authenticity)

Selection function $sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) \subseteq \{F_1, \ldots, F_n\}$.

$$sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) = \begin{cases} \emptyset \text{ if } \forall i \in \{1, \ldots, n\}, F_i = \mathsf{attacker}(x) \text{ or } \mathsf{begin}(p) \\ \{F_i\} \text{ or } \emptyset \text{ different from } \mathsf{attacker}(x) \text{ and } \mathsf{begin}(p) \end{cases}$$

Resolution (same as before)

F can be derived from $B_{P_0,S} \cup B_b$ if and only if it can be derived from saturate$(B_{P_0,S}) \cup B_b$.

Proof of authenticity

Rules for the protocol and the adversary: $B_{P_0,S}$.

Theorem:

Assume that,

for each $R \in \mathrm{saturate}(B_{P_0,S})$ such that $R = H \to \mathsf{end}(p)$,

we have $\mathsf{begin}(p) \in H$.

Then P_0 satisfies non-injective agreement with respect to S-adversaries.

Back to the example

The only rule $R \in \mathsf{saturate}(B_{P_0,S})$ such that $R = H \rightarrow \mathsf{end}(p)$ is:

$$\mathsf{begin}(\mathsf{pk}(sk_B[])) \rightarrow \mathsf{end}(\mathsf{pk}(sk_B[]))$$

so we have proved non-injective agreement.

Intuition for injective agreement

- Add session identifiers to end events $end(i, M)$
  Distinguish in which session the event happens.

- Also add them to begin events, but the session identifier of the end event is not directly visible at begin, so $begin(\text{getsession}(\rho), M)$.
  ρ is the whole environment visible at the begin event.
  getsession(ρ) extracts a subterm from ρ.

- Moving $begin$ can increase ρ, so make the proof easier.

Automatically find the position of $begin$ and the value of getsession so that we have agreement.

Example

If the example, if we move the begin event:

$$(A) \qquad !^{i_A \geq 0}\ c(x\_pk_B).\overline{c}\langle pk_A\rangle.c(x\_b).begin(x\_pk_B)\ldots$$

we obtain

$$\mathsf{begin}(\mathsf{getsession}(x\_b \mapsto b[\mathsf{pk}(sk_A[]), i_B], x\_pk_B \mapsto \mathsf{pk}(sk_B[])), \mathsf{pk}(sk_B[]))$$
$$\rightarrow \mathsf{end}(i_B, \mathsf{pk}(sk_B[]))$$

so we have injective agreement, because the environment ρ contains i_B.

Termination

The saturation algorithm does not always terminate,
but we are proving that it terminates for tagged protocols

That is, when each encryption, signature, ... is distinguished from others by a constant tag c_i

$$\{c_i, M_1, ..., M_n\}_K$$

- Large class of protocols
- Easy to add tags
- Good design practice

Experimental results

Pentium III 1GHz

NS=Needham-Schroeder WL=Woo-Lam	Time (ms)	Cases with attacks A	 B
NS public key	25	None	All[Lowe]
NS public key corrected	16	None	None
WL public key	4		All[Durante]
WL public key corrected	6		None
WL shared key (with tags)	6		All[Anderson]
WL shared key corrected (tags)	5		None
Simpler Yahalom, unidirectional	29	Key	None
Simpler Yahalom, bidirectional	101	All[Syverson]	None
Otway-Rees	62	Key[BAN]	Inj, Key[BAN]
Simpler Otway-Rees	10	All[Paulson]	All[Paulson]
Main mode of Skeme	67	None	None

Conclusion

- An automatic and efficient verifier.
- Can handle an unbounded number of sessions
- Can verify several notions of authenticity
  (non-injective, injective agreement, and even recentness).