Automated Verification of Selected Equivalences for Security Protocols

Bruno Blanchet
CNRS, École Normale Supérieure, Paris

Martín Abadi
University of California, Santa Cruz

Cédric Fournet
Microsoft Research, Cambridge

December 2005

Introduction

Analysis of cryptographic protocols:

- Powerful automatic tools for proving properties on behaviors (traces) of protocols (secrecy of keys, correspondences).

- Many important properties can be formalized as process equivalences, not as properties on behaviors:
  - secrecy of a boolean x in $P(x)$: $P(\text{true}) \approx P(\text{false})$
  - the process P implements an ideal specification Q: $P \approx Q$

  Equivalences are usually proved by difficult, long manual proofs. Already much research on this topic, using in particular sophisticated bisimulation techniques (e.g., Boreale et al).

Equivalences as properties of behaviors (1)

Goal: extend tools designed for proving properties of behaviors (here ProVerif) to the proof of process equivalences.

- We focus on equivalences between processes that differ only by the terms they contain, e.g., $P(\mathsf{true}) \approx P(\mathsf{false})$.

  Many interesting equivalences fall into this category.

- We introduce biprocesses to represent pairs of processes that differ only by the terms they contain.

  $P(\mathsf{true})$ and $P(\mathsf{false})$ are variants of a biprocess $P(\mathsf{diff}[\mathsf{true}, \mathsf{false}])$.

  The variants give a different interpretation to $\mathsf{diff}[\mathsf{true}, \mathsf{false}]$, true for the first variant, false for the second one.

Equivalences as properties of behaviors (2)

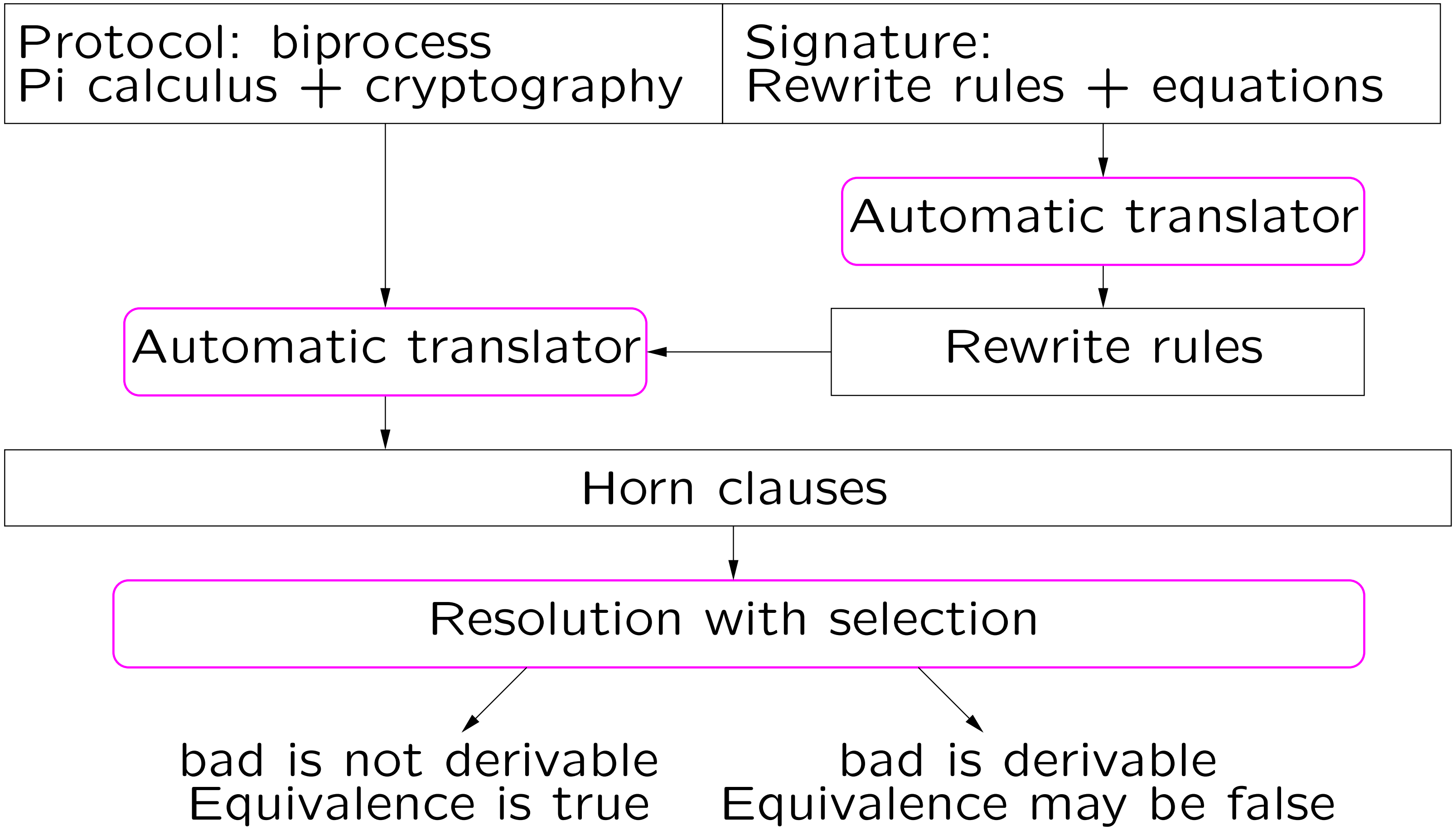
- We introduce a new operational semantics for biprocesses:

  A biprocess reduces when both variants reduce in the same way and after reduction, they still differ only by terms (so can be written using diff).

- We establish $P(\mathsf{true}) \approx P(\mathsf{false})$ by reasoning on behaviors of $P(\mathsf{diff}[\mathsf{true}, \mathsf{false}])$:

  If, for all reachable configurations, both variants reduce in the same way, then we have equivalence.

Overview of the verification method



The process calculus

Extension of the pi-calculus with function symbols for 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
$D ::=$	term evaluations
$\quad M$	term
$\quad$ eval $h(D_1, \ldots, D_n)$	function evaluation
$P, Q, R ::=$	processes
$\quad M(x).P$	input
$\quad \overline{M}\langle N \rangle.P$	output
$\quad let\ x = D\ in\ P\ else\ Q$	term evaluation
$\quad \mathbf{0} \quad P \mid Q \quad !P \quad (\nu a)P$	

Representation of cryptographic primitives

Two possible representations:

- When success/failure is visible: destructors with rewrite rules

  constructor $sencrypt$

  destructor $sdecrypt(sencrypt(x, y), y) \rightarrow x$

  The $else$ clause of the term evaluation is executed when no rewrite rule of some destructor applies.

- When success/failure is not visible: equations

  $sdecrypt(sencrypt(x, y), y) = x$

  $sencrypt(sdecrypt(x, y), y) = x$

The treatment of equations is one the main contributions of this work.

Semantics

$D \Downarrow M$ when the term evaluation D evaluates to M.
Uses rewrite rules of destructors and equations.

$\equiv$ transforms processes so that reduction rules can be applied.

Main reduction rules:

$\overline{N}\langle M\rangle.Q \mid N'(x).P \ \rightarrow \ Q \mid P\{M/x\}$ (Red I/O)
if $\Sigma \vdash N = N'$

$let\ x = D\ in\ P\ else\ Q \rightarrow P\{M/x\}$ (Red Fun 1)
if $D \Downarrow M$

$let\ x = D\ in\ P\ else\ Q \rightarrow Q$ (Red Fun 2)
if there is no M such that $D \Downarrow M$

Observational equivalences and biprocesses

Two processes P and Q are observationally equivalent ($P \approx Q$) when the adversary cannot distinguish them.

A biprocess P is a process with diff.
$\mathsf{fst}(P) =$ the process obtained by replacing $\mathsf{diff}[M, M']$ with M.
$\mathsf{snd}(P) =$ the process obtained by replacing $\mathsf{diff}[M, M']$ with M'.

P satisfies observational equivalence when $\mathsf{fst}(P) \approx \mathsf{snd}(P)$.

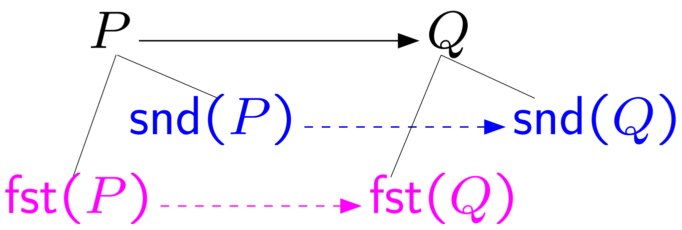
Semantics of biprocesses

A biprocess reduces when both variants of the process reduce in the same way.

$\overline{N}\langle M\rangle.Q \mid N'(x).P \;\rightarrow\; Q \mid P\{M/x\}$ (Red I/O)
if $\Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N')$ and $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$

$\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q \rightarrow P\{\mathsf{diff}[M_1, M_2]/x\}$ (Red Fun 1)
if $\mathsf{fst}(D) \Downarrow M_1$ and $\mathsf{snd}(D) \Downarrow M_2$

$\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q \rightarrow Q$ (Red Fun 2)
if there is no M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ and
there is no M_2 such that $\mathsf{snd}(D) \Downarrow M_2$



Proof of observational equivalence using biprocesses

Let P_0 be a closed biprocess.

If for all configurations P reachable from P_0 (in the presence of an adversary), both variants of P reduce in the same way, then P_0 satisfies observational equivalence.

Formalizing the adversary

Let P_0 be a closed biprocess.

If for all configurations P reachable from P_0 (in the presence of an adversary), both variants of P reduce in the same way, then P_0 satisfies observational equivalence.

An adversary is represented by a plain evaluation context (evaluation context without diff), so:

If, for all plain evaluation contexts C and reductions $C[P_0] \rightarrow^* P$, both variants of P reduce in the same way, then P_0 satisfies observational equivalence.

Formalizing “reduce in the same way”

The biprocess P is uniform when
$\mathsf{fst}(P) \to Q_1$ implies $P \to Q$ for some biprocess Q with $\mathsf{fst}(Q) \equiv Q_1$,
and symmetrically for $\mathsf{snd}(P) \to Q_2$.

If, for all plain evaluation contexts C and reductions $C[P_0] \to^* P$,
the biprocess P is uniform,
then P_0 satisfies observational equivalence.

Result

Let P_0 be a closed biprocess.

Suppose that, for all plain evaluation contexts C, all evaluation contexts C', and all reductions $C[P_0] \rightarrow^* P$,

1. the (Red I/O) rules apply in the same way on both variants.

   if $P \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$, then $\Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N')$ if and only if $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$,

2. the (Red Fun) rules apply in the same way on both variants.

   if $P \equiv C'[\mathit{let}\ x = D\ \mathit{in}\ Q\ \mathit{else}\ R]$, then there exists M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ if and only if there exists M_2 such that $\mathsf{snd}(D) \Downarrow M_2$.

Then P_0 satisfies observational equivalence.

Example: Non-deterministic encryption

Non-deterministic public-key encryption is modeled by an equation:

$$dec(enc(x, pk(s), a), s) = x$$

Without knowledge of the decryption key, ciphertexts appear to be unrelated to the plaintexts.

Ciphertexts are indistinguishable from fresh names:

$$(\nu s)(\overline{c}\langle pk(s)\rangle \mid !c'(x).(\nu a)\overline{c}\langle \mathsf{diff}[enc(x, pk(s), a), a]\rangle)$$

satisfies equivalence.

This equivalence can be proved using the previous result, and verified automatically by ProVerif.

Treatment of equations

We translate equations into rewrite rules, much easier to handle (and already handled in ProVerif).

This approach has limitations: it cannot handle associative function symbols, such as XOR.

Its main advantage is its ease of implementation and efficiency: we can still use syntactic unification for resolution.

Treatment of equations

We say that a set of terms $\mathcal{M}$ is in "normal form" relatively to the rewriting system $\mathcal{S}$ and an equational theory Σ if and only if:
– all terms in $\mathcal{M}$ are irreducible by $\mathcal{S}$,
– and for all N_1, N_2 subterms of terms of $\mathcal{M}$, if $\Sigma \vdash N_1 = N_2$ then $N_1 = N_2$.

(Terms may have several normal forms, all equal modulo Σ.)

The rewrite rules must be such that all normal forms of each term are reached by applying exactly one rewriting step for each function symbol (that is, by evaluating each function symbol).

Example: encryption

Σ with equations:

$$sdecrypt(sencrypt(x, y), y) = x$$
$$sencrypt(sdecrypt(x, y), y) = x$$

is translated into the rewrite rules:

$$sdecrypt(x, y) \rightarrow sdecrypt(x, y) \qquad sencrypt(x, y) \rightarrow sencrypt(x, y)$$
$$sdecrypt(sencrypt(x, y), y) \rightarrow x \qquad sencrypt(sdecrypt(x, y), y) \rightarrow x$$

Each term has a single normal form, irreducible by $sdecrypt(sencrypt(x, y), y) \rightarrow x$ and $sencrypt(sdecrypt(x, y), y) \rightarrow x$.

Example: Diffie-Hellman

Σ with equation:

$$(\mathtt{b}\,\hat{}\,x)\,\hat{}\,y = (\mathtt{b}\,\hat{}\,y)\,\hat{}\,x$$

is translated into the rewrite rules:

$$\mathtt{b} \longrightarrow \mathtt{b} \qquad x\,\hat{}\,y \longrightarrow x\,\hat{}\,y \qquad (\mathtt{b}\,\hat{}\,x)\,\hat{}\,y \longrightarrow (\mathtt{b}\,\hat{}\,y)\,\hat{}\,x$$

Terms may have several normal forms: applying ^ to $\mathtt{b}\,\hat{}\,x$ and y yields two normal forms of $(g\,\hat{}\,x)\,\hat{}\,y$: $(g\,\hat{}\,x)\,\hat{}\,y$ and $(g\,\hat{}\,y)\,\hat{}\,x$.

Result

We have built algorithms that compute rewrite rules from a set of equations.

We have shown that, for each trace with equations, there is a corresponding trace with rewrite rules, and conversely.

Then we obtain a result for proving equivalences using rewrite rules instead of equations.

(See formal details in the paper.)

Translation into clauses

As in our previous work, we translate the protocol and the adversary into a set of Horn clauses.

The predicates differ in order to translate behaviors of biprocesses instead of processes:

$F ::=$	facts
$\quad \mathrm{att}'(M, M')$	the attacker has M (resp. M')
$\quad \mathrm{msg}'(M_1, M_2, M'_1, M'_2)$	message M_2 on channel M_1 (resp. M'_2 on M'_1)
$\quad \mathrm{input}'(M, M')$	input on M (resp. M')
$\quad \mathrm{nounif}(M, M')$	M and M' do not unify modulo Σ
$\quad \mathrm{bad}$	the property may be false

Magenta arguments for the first variant of the biprocess, blue ones for the second variant.

Translation into clauses: nounif

The predicate nounif is used to detect when the application of a destructor fails.

Let M and M' be closed terms.
M and M' may contain special constants $\mathrm{g}, \mathrm{g}', \ldots$ in a set $GVar$.

nounif(M, M') holds when there is no closed substitution σ with domain $GVar$ such that $\Sigma \vdash \sigma M = \sigma M'$.

Example: with a destructor $sdecrypt(sencrypt(x, y), y) \rightarrow x$,
nounif$((x, y), (sencrypt(\mathrm{g}, \mathrm{g}'), \mathrm{g}'))$ holds when
the destructor application $sdecrypt(x, y)$ fails.

Clauses for the attacker (1)

For each $a \in fn(P_0)$, $\mathrm{att}'(a, a)$

For some b that does not occur in P_0, $\mathrm{att}'(b[x], b[x])$

For each function h, for each pair of rewrite rules $h(M_1, \ldots, M_n) \to M$ and $h(M'_1, \ldots, M'_n) \to M'$ in $\mathrm{def}_{\Sigma'}(h)$ (after renaming of variables),

$$\mathrm{att}'(M_1, M'_1) \wedge \ldots \wedge \mathrm{att}'(M_n, M'_n) \to \mathrm{att}'(M, M')$$

$\mathrm{msg}'(x, y, x', y') \wedge \mathrm{att}'(x, x') \to \mathrm{att}'(y, y')$

$\mathrm{att}'(x, x') \wedge \mathrm{att}'(y, y') \to \mathrm{msg}'(x, y, x', y')$

$\mathrm{att}'(x, x') \to \mathrm{input}'(x, x')$

Clauses for the attacker (2): detecting attacks

Condition 1: If a communication succeeds in the first variant and fails in the second one, we may have an attack.

$$\mathsf{input}'(x, x') \wedge \mathsf{msg}'(x, z, y', z') \wedge \mathsf{nounif}(x', y') \rightarrow \mathsf{bad}$$

Condition 2: If a destructor application succeeds in the first variant and fails in the second one, we may have an attack.

For each rewrite rule $g(M_1, \ldots, M_n) \rightarrow M$ in $\mathsf{def}_{\Sigma'}(g)$,

$$\bigwedge_{g(M'_1,\ldots,M'_n)\rightarrow M' \text{ in } \mathsf{def}_{\Sigma'}(g)} \mathsf{nounif}((x'_1, \ldots, x'_n), GVar((M'_1, \ldots, M'_n)))$$

$$\wedge\, \mathsf{att}'(M_1, x'_1) \wedge \ldots \wedge \mathsf{att}'(M_n, x'_n) \rightarrow \mathsf{bad}$$

plus symmetric clauses, where we swap the arguments of att' and input', and the first and third arguments of msg'.

Example of clauses for primitives

For non-deterministic encryption $dec(enc(x, pk(s), a), s) = x$:

Rewrite rules for dec:

$$dec(x, y) \to dec(x, y) \qquad dec(enc(x, pk(y), z), y) \to x$$

Clause for enc:

$$\mathsf{att}'(x, x') \wedge \mathsf{att}'(y, y') \wedge \mathsf{att}'(z, z') \to \mathsf{att}'(enc(x, y, z), enc(x', y', z'))$$

Clauses for dec:

$$\mathsf{att}'(x, x') \wedge \mathsf{att}'(y, y') \to \mathsf{att}'(dec(x, y), dec(x', y'))$$
$$\mathsf{att}'(enc(x, pk(y), z), x') \wedge \mathsf{att}'(y, y') \to \mathsf{att}'(x, dec(x', y'))$$
$$\mathsf{att}'(x, enc(x', pk(y'), z')) \wedge \mathsf{att}'(y, y') \to \mathsf{att}'(dec(x, y), x')$$
$$\mathsf{att}'(enc(x, pk(y), z), enc(x', pk(y'), z')) \wedge \mathsf{att}'(y, y') \to \mathsf{att}'(x, x')$$

Clauses for the protocol (1)

If a participant A receives $M_1, \ldots, M_n$ on channels $C_1, \ldots, C_n$ in the first variant, $M'_1, \ldots, M'_n$ on $C'_1, \ldots, C'_n$ in the second variant, respectively, then

– outputs M on C in the first variant, M' on C' in the second one:

$$\mathsf{msg}'(C_1, M_1, C'_1, M'_1) \wedge \ldots \wedge \mathsf{msg}'(C_n, M_n, C'_n, M'_n) \rightarrow \mathsf{msg}'(C, M, C', M')$$

– executes an input on C in the first variant, C' in the second one:

$$\mathsf{msg}'(C_1, M_1, C'_1, M'_1) \wedge \ldots \wedge \mathsf{msg}'(C_n, M_n, C'_n, M'_n) \rightarrow \mathsf{input}'(C, C')$$

Clauses for the protocol (2): detecting attacks

If a participant A receives $M_1, \ldots, M_n$ on channels $C_1, \ldots, C_n$ in the first variant, $M'_1, \ldots, M'_n$ on $C'_1, \ldots, C'_n$ in the second variant, respectively, then

– executes $let\ x = \mathsf{eval}\ g(\mathsf{diff}[T_1, T'_1], \ldots, \mathsf{diff}[T_{n'}, T'_{n'}])\ in$:
for each rewrite rule $g(N_1, \ldots, N_{n'}) \rightarrow N$ in $\mathsf{def}_{\Sigma'}(g)$,

$$\mathsf{msg}'(\sigma C_1, \sigma M_1, C'_1, M'_1) \wedge \ldots \wedge \mathsf{msg}'(\sigma C_n, \sigma M_n, C'_n, M'_n) \\ \wedge \mathsf{fails}(g(N'_1, \ldots, N'_n)) \rightarrow \mathsf{bad}$$

where σ is the most general unifier of $N_1, \ldots, N_{n'}$ and $T_1, \ldots, T_{n'}$ and $\mathsf{fails}(g(T'_1, \ldots, T'_{n'})) = \bigwedge_{g(N'_1, \ldots, N'_{n'}) \rightarrow N' \in \mathsf{def}_{\Sigma'}(g)} \mathsf{nounif}((T'_1, \ldots, T'_{n'}), GVar(N'_1, \ldots, N'_{n'}))$

Plus the symmetric rule where we swap the two variants of the biprocess.

Example: some generated clauses

The biprocess of the non-deterministic encryption example:

$$(\nu s)(\overline{c}\langle pk(s)\rangle \mid !c'(x).(\nu a)\overline{c}\langle \mathsf{diff}[enc(x, pk(s), a), a]\rangle)$$

yields the clauses:

$$\mathsf{msg}'(c, pk(s), c, pk(s))$$
$$\mathsf{msg}'(c', x, c', x') \rightarrow \mathsf{msg}'(c, enc(x, pk(s), a[i, x]), c, a[i, x'])$$

The first clause corresponds to the output of the public key $pk(s)$.

The second clause corresponds to the other output.

Resolution algorithm

Theorem 1 *If* bad *is not a logical consequence of the clauses, then P_0 satisfies observational equivalence.*

We determine whether bad is a logical consequence of the clauses using a resolution-based algorithm.

This algorithm uses domain-specific simplification steps
(for predicate nounif in particular, using unification
modulo the equational theory of Σ).

Basic resolution algorithm

Resolution with free selection:

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

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

The selection function avoids selecting:
– nounif (handled by specific simplification steps)
– bad (otherwise, we conclude that bad may be derivable)
– att$'(x, x')$ (for termination)

Simplification steps for nounif (examples)

Unification modulo Σ can be computed thanks the rewrite rules built from the equations.

Using unification modulo Σ,

– in the Diffie-Hellman example, $H \wedge \text{nounif}(x\hat{\ }y, x'\hat{\ }y') \rightarrow C$ becomes $H \wedge \text{nounif}((x, y), (x', y')) \wedge \text{nounif}((x, x'), (\mathtt{b}\hat{\ }y', \mathtt{b}\hat{\ }y)) \rightarrow C$.

– without equational theory,
$H \wedge \text{nounif}((sencrypt(x', y'), z'), (sencrypt(\mathtt{g}, y), \mathtt{g})) \rightarrow C$
becomes $H \wedge \text{nounif}((x', y', z'), (\mathtt{g}, y, \mathtt{g})) \rightarrow C$
which becomes $H \wedge \text{nounif}((y', z'), (y, x')) \rightarrow C$.

Scenarios with several stages

Many protocols can be broken into stages.
Example: a protocol discloses a session key after the conclusion of a session.

- We introduce a stage prefix $t : P$, which means that P executes in stage t.

- We have predicates att'_t, msg'_t, input'_t for each stage.

  We use att'_t, msg'_t, input'_t when translating P in $t : P$.

  The clauses for the attacker are replicated for each att'_t.

  $\mathsf{att}'_t(x, x') \rightarrow \mathsf{att}'_{t+1}(x, x')$.

Application: Weak secrets

The process P protects a password w against off-line guessing attacks when the following biprocess satisfies equivalence:

$$(\nu w)(0 : P \mid 1 : (\nu w')\overline{c}\langle \mathsf{diff}[w, w']\rangle)$$

In stage 0, the protocol interacts with the adversary (on-line stage).

In stage 1, the adversary receives either the password w or an unrelated fresh value w', but no longer interacts with the protocol (off-line stage).

The adversary can verify that he guessed the password correctly if and only if he can distinguish these two variants.

We can prove this property using our technique (done for 4 versions of EKE).

Other applications

Some other applications:

- Authenticity: We can formalize authenticity as an equivalence and prove it (for the Wide-Mouth Frog protocol).

- JFK: We can show that the encrypted messages of JFK are equivalent to fresh names, with our technique plus the property that observational equivalence is contextual.

Total runtime: 45 s on a Pentium M 1.8 GHz.

Conclusion

Contributions:

- Fully automatic proof of some process equivalences.
- Treatment of cryptographic primitives represented by equations.

Implementation and more information at

`http://www.di.ens.fr/~blanchet/obsequi/`