Automated Security Proofs with Sequences of Games

Bruno Blanchet and David Pointcheval
CNRS, École Normale Supérieure, Paris

March 2006

Introduction: An automatic prover

We have implemented an automatic prover sound in the computational model:

- proves secrecy and non-injective correspondence properties.
- handles various cryptographic primitives: macs (message authentication codes), stream and block ciphers, public-key encryption, signatures, hash functions, . . .
- works for N sessions with an active adversary.
- gives a bound on the probability of an attack (exact security).

Produced proofs

As in Shoup's method, the proof is a sequence of games:

- The first game is the real protocol.
- One goes from one game to the next by syntactic transformations or by applying the definition of security of a cryptographic primitive.
  The difference of probability between consecutive games is bounded.
- The last game is "ideal": the security property can be read directly on it.
  (The advantage of the adversary is 0 for this game.)

A game is formalized as an extension of the pi calculus with function symbols and arrays.

Process calculus for games: terms

Inspired by:
– the calculus of [Lincoln, Mitchell, Mitchell, Scedrov]
– the calculus of [Laud, CCS'05]

Extension to arrays.

$M ::=$	terms
$\quad x, y, z, x[M_1, \ldots, M_n]$	variable
$\quad f(M_1, \ldots, M_n)$	function application

Function symbols f correspond to functions efficiently computable by deterministic Turing machines.

Process calculus for games: processes

$Q ::=$	input process
0	nil
$Q \mid Q'$	parallel composition
$!^{i \leq N} Q$	replication N times
$\mathbf{newChannel}\ c; Q$	restriction for channels
$c(x_1 : T_1, \ldots, x_m : T_m); P$	input
$P ::=$	output process
$\overline{c}\langle M_1, \ldots, M_m \rangle; Q$	output
$\mathbf{event}\ e(M); P$	event
$\mathbf{new}\ x : T; P$	random number generation (uniform)
$\mathbf{let}\ x : T = M\ \mathbf{in}\ P$	assignment
$\mathbf{if}\ M\ \mathbf{then}\ P\ \mathbf{else}\ P'$	conditional
$\mathbf{find}\ j \leq N\ \mathbf{suchthat}\ \mathbf{defined}(x[j], \ldots) \wedge M\ \mathbf{then}\ P\ \mathbf{else}\ P'$	array lookup

Process calculus for games: semantics

The semantics is purely probabilistic (no non-determinism).

A semantic configuration contains
– an output process (the process currently being executed)
– a set of input processes (waiting for messages)

When executing the output $\overline{c}\langle M_1, \dots, M_m\rangle; Q$:
– we randomly chose an input of the same arity on the same channel in the set of waiting input processes $c(x_1 : T_1, \dots, x_m : T_m); P$,
– store messages $M_1, \dots, M_m$ in variables $x_1, \dots, x_m$,
– schedule P for execution and add Q to the set of waiting input processes.

Arrays

Arrays replace lists often used in cryptographic proofs.

A variable defined under a replication is implicitly an array:

$$!^{i \le N} \mathbf{let}\ x = M\ \mathbf{in}\ \ldots$$

in fact defines $x[i]$, for i in $1, \ldots, N$. Under $!^{i \le N}$, we write x for $x[i]$.

Only variables with the current indexes can be assigned.

Variables may be defined at several places, but only one definition can be executed for the same indexes.
($\mathbf{if}\ \ldots\ \mathbf{then}\ \mathbf{let}\ x = M\ \mathbf{in}\ P\ \mathbf{else}\ \mathbf{let}\ x = M'\ \mathbf{in}\ P'$ is ok)

Arrays (continued)

find performs an array lookup:

$$
\begin{array}{l}
\ \ !^{i \leq N} \mathbf{let}\ x = M\ \mathbf{in}\ P \\
|\ !^{i' \leq N'} c(y:T) \mathbf{find}\ j \leq N\ \mathbf{suchthat}\ \mathbf{defined}(x[j]) \wedge y = x[j]\ \mathbf{then}\ \ldots
\end{array}
$$

Note that **find** is here used outside the scope of x.

This is the only way of getting access to values of variables in other sessions.

When several sessions satisfy the condition of the **find**,
the returned index is chosen randomly, with uniform probability.

Main notion of security: observational equivalence

Two processes (games) Q_1, Q_2 are observationally equivalent with probability p when an adversary running in time t has probability at most $p(t)$ of distinguishing them:

$$Q_1 \approx_p Q_2$$

In the formal definition, the adversary is represented by an acceptable evaluation context $C ::= [] \quad C \mid Q \quad Q \mid C \quad \textbf{newChannel } c; C$.

Observational equivalence is reflexive and symmetric.
If $Q_1 \approx_p Q_2$ and $Q_2 \approx_{p'} Q_3$ then $Q_1 \approx_{p+p'} Q_3$.

It is contextual: $Q_1 \approx_p Q_2$ implies $C[Q_1] \approx_{p'} C[Q_2]$
where C is any acceptable evaluation context running in time t_C and $p'(t) = p(t + t_C)$.

Proof technique

We transform a game into an observationally equivalent one using:

- observational equivalences $L \approx_p R$ that define primitives and are given as axioms, put inside a context:

$$Q_1 \approx_0 C[L] \approx_{p'} C[R] \approx_0 Q_2$$

- syntactic transformations: simplification, expansion of assignments, ...

Proof strategy: try all available observational equivalence axioms. When they fail, they may suggest syntactic transformations that make them succeed.

Example of the FDH signature

hash hash function (in the random oracle model)
$\mathsf{f}(pk, m)$ one-way trapdoor permutation, with inverse $\mathsf{mf}(sk, m)$.

We define a signature scheme as follows:
– signature $\mathsf{sign}(m, sk) = \mathsf{mf}(sk, \mathsf{hash}(m))$
– verification $\mathsf{check}(m, pk, s) = (\mathsf{f}(pk, s) = \mathsf{hash}(m))$

Our goal is to show that this signature scheme is secure against existential forgery under chosen message attacks.

FDH: security of a hash function (random oracle model)

A hash function is equivalent to a "random function":
a function that returns a new random number when it is a called on a new argument, and returns the same result when it is called on the same argument.

$!^{N_{\mathsf{h}}}\ (x : \mathit{bitstring})\ \longrightarrow\ \mathsf{hash}(x)\ [\mathit{all}]$

$\approx_0$

$!^{N_{\mathsf{h}}}\ (x : \mathit{bitstring})\ \longrightarrow$

$\quad \textbf{find}\ j \leq N_{\mathsf{h}}\ \textbf{suchthat defined}(x[j], r[j]) \wedge (x = x[j])$

$\quad \textbf{then}\ r[j]$

$\quad \textbf{else new}\ r : D;\ r$

FDH: permutation

mf is the inverse of f:

$$\forall r : seed, x : D; \mathsf{mf}(\mathsf{skgen}(r), \mathsf{f}(\mathsf{pkgen}(r), x)) = x$$

f is injective:

$$\forall k : pkey, x : D, x' : D; (\mathsf{f}(k, x) = \mathsf{f}(k, x')) = (x = x')$$

We can replace a uniformly distributed random number x with $\mathsf{f}(\mathsf{pkgen}(r), x')$ where x' is a uniformly distributed random number:

$\mathbf{new}\ x : D; (k : pkey) \rightarrow x$
$\approx_0$
$\mathbf{new}\ x : D; (k : pkey) \rightarrow \mathsf{f}(k, x)$

FDH: permutation

mf is the inverse of f:

$$\forall r : seed, x : D; \mathsf{mf}(\mathsf{skgen}(r), \mathsf{f}(\mathsf{pkgen}(r), x)) = x$$

f is injective:

$$\forall k : pkey, x : D, x' : D; (\mathsf{f}(k, x) = \mathsf{f}(k, x')) = (x = x')$$

We can replace a uniformly distributed random number x with $\mathsf{f}(\mathsf{pkgen}(r), x')$ where x' is a uniformly distributed random number:

$!^{N_\mathsf{k}}$ **new** $r : seed$; (
  $!^{N_0}$ $() \rightarrow \mathsf{pkgen}(r)$,
  $!^{N_\mathsf{f}}$ **new** $x : D$; $(!^{N_1}\ () \rightarrow \mathsf{mf}(\mathsf{skgen}(r), x), !^{N_2}\ () \rightarrow x))$

$\approx_0$

$!^{N_\mathsf{k}}$ **new** $r : seed$; (
  $!^{N_0}$ $() \rightarrow \mathsf{pkgen}(r)$,
  $!^{N_\mathsf{f}}$ **new** $x' : D$; $(!^{N_1}\ () \rightarrow x', !^{N_2}\ () \rightarrow \mathsf{f}(\mathsf{pkgen}(r), x')))$

FDH: one-wayness

LR (in the presence of an adversary that runs in time t) executes **event** bad with probability at most $\mathsf{Succ}^{\mathsf{ow}}_{\mathcal{P}}(t)$, where

$$LR = c(); \mathbf{new}\ r : seed; \mathbf{new}\ x : D; \overline{c}\langle \mathsf{pkgen}(r), \mathsf{f}(\mathsf{pkgen}(r), x)\rangle;$$
$$c(x' : D); \mathbf{if}\ x' = x\ \mathbf{then}\ \mathbf{event}\ \mathsf{bad}; \overline{c}\langle\rangle\ \mathbf{else}\ \overline{c}\langle\rangle$$

Event bad is executed when the adversary, given the public key $\mathsf{pkgen}(r)$ and the image of some x by f, manages to find x (without having the trapdoor).

FDH: one-wayness

$$
\begin{array}{l}
!^{N_{\mathsf{k}}}\ \mathbf{new}\ r : \mathit{seed};\ (\\
\quad !^{N_0}\ ()\ \rightarrow\ \mathsf{pkgen}(r),\\
\quad !^{N_{\mathsf{f}}}\ \mathbf{new}\ y : D;\ (\\
\qquad !^{N_1}\ ()\ \rightarrow\ \mathsf{f}(\mathsf{pkgen}(r), y),\\
\qquad !^{N_2}\ (x : D)\ \rightarrow\ (x = y),\\
\qquad !^{N_3}\ ()\ \rightarrow\ y))
\end{array}
$$

$$\approx_{N_{\mathsf{k}} \times N_{\mathsf{f}} \times \mathsf{Succ}^{\mathsf{ow}}_{\mathcal{P}}(t + (N_{\mathsf{k}} - 1)t_{\mathsf{pkgen}} + (N_{\mathsf{k}} N_{\mathsf{f}} - 1)t_{\mathsf{f}})}$$

$$
\begin{array}{l}
!^{N_{\mathsf{k}}}\ \mathbf{new}\ r : \mathit{seed};\ (\\
\quad !^{N_0}\ ()\ \rightarrow\ \mathsf{pkgen}'(r),\\
\quad !^{N_{\mathsf{f}}}\ \mathbf{new}\ y : D;\ (\\
\qquad !^{N_1}\ ()\ \rightarrow\ \mathsf{f}'(\mathsf{pkgen}'(r), y),\\
\qquad !^{N_2}\ (x : D)\ \rightarrow\ \mathbf{find}\ j \leq N_3\ \mathbf{suchthat}\ \mathbf{defined}(k[j]) \wedge \mathsf{true}\\
\qquad\qquad\qquad\qquad\quad \mathbf{then}\ x = y\ \mathbf{else}\ \mathsf{false},\\
\qquad !^{N_3}\ ()\ \rightarrow\ \mathbf{let}\ k : \mathit{bitstring} = \mathsf{mark}\ \mathbf{in}\ y))
\end{array}
$$

FDH: initial game

$start()$; **new** $r : seed$;
let $sk : skey = \mathsf{skgen}(r)$ **in**
let $pk : pkey = \mathsf{pkgen}(r)$ **in** $\overline{c0}\langle pk \rangle$;
((* hash oracle *)
$\quad !^{!\_27 \leq q_H}\ c4[!\_27](x : bitstring);\ \overline{c5[!\_27]}\langle \mathsf{hash}(x) \rangle$
| (* signature oracle *)
$\quad !^{!\_28 \leq q_S}\ c1[!\_28](m : bitstring);\ \overline{c2[!\_28]}\langle \mathsf{mf}(sk, \mathsf{hash}(m)) \rangle$
| (* forged signature? *)
$\quad c3(m' : bitstring, s : D)$;
$\quad$ **if** $(\mathsf{f}(pk, s) = \mathsf{hash}(m'))$ **then**
$\quad$ **find** $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then**
$\qquad \overline{0}$
$\quad$ **else**
$\qquad$ **event** bad
)

FDH step 1: apply the security of the hash function

Replace each occurrence of $\mathsf{hash}(M)$ with a lookup in the arguments of previous calls to hash.

– If M is found, return the same result as the previous result.

– Otherwise, pick a new random number and return it.

For instance, $\overline{c5[!\_27]}\langle \mathsf{hash}(x)\rangle$ is replaced with

$$
\begin{array}{l}
\textbf{find}\ \ \textbf{suchthat defined}(m', r\_30) \wedge (x = m')\ \textbf{then}\\
\quad \overline{c5[!\_27]}\langle r\_30\rangle\\
\textbf{orfind}\ @i1 \leq q_S\ \textbf{suchthat defined}(m[@i1], r\_32[@i1])\\
\quad \wedge (x = m[@i1])\ \textbf{then}\ \overline{c5[!\_27]}\langle r\_32[@i1]\rangle\\
\textbf{orfind}\ @i2 \leq q_H\ \textbf{suchthat defined}(x[@i2], r\_34[@i2])\\
\quad \wedge (x = x[@i2])\ \textbf{then}\ \overline{c5[!\_27]}\langle r\_34[@i2]\rangle\\
\textbf{else}\\
\quad \textbf{new}\ r\_34 : D;\ \overline{c5[!\_27]}\langle r\_34\rangle
\end{array}
$$

FDH step 2: simplify

(* forged signature? *)
$c3(m' : bitstring, s : D);$
find **suchthat defined**$(m', r\_30) \wedge (m' = m')$ **then**
  if $(\mathsf{f}(pk, s) = r\_30)$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else event** bad
orfind $@i5 \leq q_S$ **suchthat defined**$(m[@i5], r\_32[@i5]) \wedge (m' = m[@i5])$ **then**
  if $(\mathsf{f}(pk, s) = r\_32[@i5])$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else event** bad
orfind $@i6 \leq q_H$ **suchthat defined**$(x[@i6], r\_34[@i6]) \wedge (m' = x[@i6])$ **then**
  if $(\mathsf{f}(pk, s) = r\_34[@i6])$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else event** bad
else
  new $r\_30 : D;$
  if $(\mathsf{f}(pk, s) = r\_30)$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else event** bad

The red test always succeeds, so the blue part becomes $\overline{0}$.
$(m' = m')$ becomes true.

FDH step 3: substitute sk with its value

The variable sk is replaced with $\mathsf{skgen}(r)$, and the assignment $\mathbf{let}\ sk : skey = \mathsf{skgen}(r)$ is removed.

This transformation is advised in order to able to apply the permutation property.

FDH step 4: permutation

(* signature oracle *)

$!^{!\_28 \leq q_S}$

$c1[!\_28](m : bitstring);$

$\textbf{find}\ \ \textbf{suchthat defined}(m', r\_30) \wedge (m = m')\ \textbf{then}$

$\quad \overline{c2[!\_28]}\langle \mathsf{mf}(\mathsf{skgen}(r), r\_30)\rangle$

$\textbf{orfind}\ @i3 \leq q_S\ \textbf{suchthat defined}(m[@i3], r\_32[@i3]) \wedge (m = m[@i3])\ \textbf{then}$

$\quad \overline{c2[!\_28]}\langle \mathsf{mf}(\mathsf{skgen}(r), r\_32[@i3])\rangle$

$\textbf{orfind}\ @i4 \leq q_H\ \textbf{suchthat defined}(x[@i4], r\_34[@i4]) \wedge (m = x[@i4])\ \textbf{then}$

$\quad \overline{c2[!\_28]}\langle \mathsf{mf}(\mathsf{skgen}(r), r\_34[@i4])\rangle$

$\textbf{else}$

$\quad \textbf{new}\ r\_32 : D;$

$\quad \overline{c2[!\_28]}\langle \mathsf{mf}(\mathsf{skgen}(r), r\_32)\rangle$

$\textbf{new}\ r\_i : D$ becomes $\textbf{new}\ y\_i : D$,

$\mathsf{mf}(\mathsf{skgen}(r), r\_i)$ becomes y_i,

r_i becomes $\mathsf{f}(\mathsf{pkgen}(r), y\_i)$

FDH step 5: simplify

(* forged signature? *)
$c3(m' : bitstring, s : D);$
find **suchthat defined**$(y\_30) \wedge \mathsf{true}$ **then**
  if $(\mathsf{f}(pk, s) = \mathsf{f}(\mathsf{pkgen}(r), y\_30)))$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else event** bad
orfind $@i5 \leq q_S$ **suchthat defined**$(m[@i5], r\_32[@i5]) \wedge (m' = m[@i5])$ **then** $\overline{0}$
orfind $@i6 \leq q_H$ **suchthat defined**$(x[@i6], r\_34[@i6]) \wedge (m' = x[@i6])$ **then**
  if $(\mathsf{f}(pk, s) = \mathsf{f}(\mathsf{pkgen}(r), y\_34[@i6]))$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else event** bad
else
  new $y\_30 : D;$
  if $(\mathsf{f}(pk, s) = \mathsf{f}(\mathsf{pkgen}(r), y\_30))$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else event** bad

$\mathsf{f}(pk, s) = \mathsf{f}(\mathsf{pkgen}(r), y\_i)$ becomes $s = y\_i$,
knowing $pk = pkgen(r)$ and the injectivity of f:
$\forall k : pkey, x : D, x' : D; (\mathsf{f}(k, x) = \mathsf{f}(k, x')) = (x = x')$

FDH step 6: one-wayness

(* forged signature? *)
$c3(m' : bitstring, s : D);$
find suchthat defined$(y\_30) \land$ true **then**
  if $s = y\_30$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \land (m' = m[j])$ **then** $\bar{0}$ **else event** bad
orfind $@i5 \leq q_S$ **suchthat defined**$(m[@i5], r\_32[@i5]) \land (m' = m[@i5])$ **then** $\bar{0}$
orfind $@i6 \leq q_H$ **suchthat defined**$(x[@i6], r\_34[@i6]) \land (m' = x[@i6])$ **then**
  if $s = y\_34[@i6]$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \land (m' = m[j])$ **then** $\bar{0}$ **else event** bad
else
  new $y\_30 : D;$
  if $s = y\_30$ **then**
  find $j \leq q_S$ **suchthat defined**$(m[j]) \land (m' = m[j])$ **then** $\bar{0}$ **else event** bad

$s = y\_i$ becomes **find** $@j\_i \leq q_H$ **suchthat defined**$(k\_i[@j\_i]) \land$ true
then $s = y\_i$ **else** false,
In hash oracle, $\mathsf{f}(\mathsf{pkgen}(r), y\_i)$ becomes $\mathsf{f}'(\mathsf{pkgen}'(r), y\_i)$,
In signature oracle, y_i becomes **let** $k\_i : bitstring =$ mark **in** y_i.
Difference of probability: $(q_H + q_S + 1)\mathsf{Succ}^{\mathsf{ow}}_{\mathcal{P}}(t + t' + (q_H + q_S)t_{\mathsf{f}})$.

FDH step 7: simplify

(* forged signature? *)

$c3(m' : bitstring, s : D);$

find **suchthat** **defined**$(y\_30) \wedge \mathsf{true}$ **then**

  find $@j\_30 \leq q_H$ **suchthat** **defined**$(k\_30[@j\_30]) \wedge \mathsf{true}$ **then**

  if $s = y\_30$ **then**

  find $j \leq q_S$ **suchthat** **defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else** **event** bad

orfind $@i5 \leq q_S$ **suchthat** **defined**$(m[@i5], r\_32[@i5]) \wedge (m' = m[@i5])$ **then** $\overline{0}$

orfind $@i6 \leq q_H$ **suchthat** **defined**$(x[@i6], r\_34[@i6]) \wedge (m' = x[@i6])$ **then**

  find $@j\_34 \leq q_S$ **suchthat** **defined**$(k\_34[@j\_34]) \wedge (@i4[@j\_34] = @i6)$ **then**

  if $s = y\_34[@i6]$ **then**

  find $j \leq q_S$ **suchthat** **defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else** **event** bad

else

  new $y\_30 : D;$

  find $@j\_30 \leq q_H$ **suchthat** **defined**$(k\_30[@j\_30]) \wedge \mathsf{true}$ **then**

  if $s = y\_30$ **then**

  find $j \leq q_S$ **suchthat** **defined**$(m[j]) \wedge (m' = m[j])$ **then** $\overline{0}$ **else** **event** bad

The tests in red always succeed, so **event** bad disappears, which proves the desired property.

FDH step 7: simplify (2)

(* forged signature? *)
$c3(m' : bitstring, s : D);$
$\mathbf{find}\ \ \mathbf{suchthat}\ \mathbf{defined}(y\_30) \wedge \mathsf{true}\ \mathbf{then}$
$\quad \mathbf{find}\ @j\_30 \leq q_H\ \mathbf{suchthat}\ \mathbf{defined}(k\_30[@j\_30]) \wedge \mathsf{true}\ \mathbf{then}$
$\quad \mathbf{if}\ s = y\_30\ \mathbf{then}$
$\quad \mathbf{find}\ j \leq q_S\ \mathbf{suchthat}\ \mathbf{defined}(m[j]) \wedge (m' = m[j])\ \mathbf{then}\ \overline{0}\ \mathbf{else}\ \mathbf{event}\ \mathsf{bad}$

Definition of k_30:

$!^{!\_28 \leq q_S}$
$c1[!\_28](m : bitstring);$
$\mathbf{find}\ \ \mathbf{suchthat}\ \mathbf{defined}(m', y\_30) \wedge (m = m')\ \mathbf{then}$
$\quad \mathbf{let}\ k\_30 : bitstring = \mathsf{mark}\ \mathbf{in} \ldots$

When $k\_30[@j\_30]$ is defined, $m[@j\_30]$ is defined and $m[@j\_30] = m'$, so the test $j \leq q_S\ \mathbf{suchthat}\ \mathbf{defined}(m[j]) \wedge (m' = m[j])$ succeeds with $j = @j\_30$.
Same reasoning for the last red test.

FDH step 7: simplify (3)

(* forged signature? *)
$c3(m' : bitstring, s : D);$
$\ldots$
$\mathbf{orfind}\ @i6 \leq q_H\ \mathbf{suchthat\ defined}(x[@i6], r\_34[@i6]) \wedge (m' = x[@i6])\ \mathbf{then}$
$\quad \mathbf{find}\ @j\_34 \leq q_S\ \mathbf{suchthat\ defined}(k\_34[@j\_34]) \wedge (@i4[@j\_34] = @i6)\ \mathbf{then}$
$\quad \mathbf{if}\ s = y\_34[@i6]\ \mathbf{then}$
$\quad \mathbf{find}\ j \leq q_S\ \mathbf{suchthat\ defined}(m[j]) \wedge (m' = m[j])\ \mathbf{then}\ \overline{0}\ \mathbf{else\ event}\ \mathsf{bad}$

Definition of k_34:

$!^{!\_28 \leq q_S}$
$c1[!\_28](m : bitstring);$
$\ldots$
$\mathbf{orfind}\ @i4 \leq q_H\ \mathbf{suchthat\ defined}(x[@i4], y\_34[@i4]) \wedge (m = x[@i4])\ \mathbf{then}$
$\quad \mathbf{let}\ k\_34 : bitstring = \mathsf{mark}\ \mathbf{in} \ldots$

When $k\_34[@j\_34]$ is defined, $m[@j\_34]$ is defined and
$m[@j\_34] = x[@i4[@j\_34]] = x[@i6] = m'$
so the red test succeeds with $j = @j\_34$.

Conclusion

Hopefully a promising approach, but still some work to do:

- Extension to other cryptographic primitives, in particular Diffie-Hellman.
- Improvements to the simplification algorithm.
- Extension to other security properties: semantic security of the key, injective correspondences, . . .
- More experiments.

More information: http://www.di.ens.fr/~blanchet/cryptoc-eng.html