

A Short Tutorial on Proverif

Alfredo Pironti and Riccardo Sisto

Politecnico di Torino, Italy



Outline

- PART 1: how the tool works (Riccardo Sisto)
 - Context: Abstract modelling of security protocols (Dolev-Yao)
 - Proverif architecture
 - Specifying protocols and properties
 - Translation into Horn clauses and solving method
- PART 2: practice (Alfredo Pironti)
 - Running the tool
 - Case study: Verifying the SSH transport protocol with Proverif



Abstract (Dolev-Yao) Modelling of Security Protocols

- Data represented as terms of a term algebra (e.g. $\text{sencrypt}(\text{pair}(x, y), k)$)
- Active attackers
 - Can eavesdrop on public channels
 - Can delete, modify, send messages on public channels
 - Can generate new nonces
 - Can forge new messages by combining known data and their generated nonces by public operations (under the constraints of perfect cryptography)
 - Cannot forge or guess data in other ways



Perfect Cryptography

- Models the ideal properties of cryptography
- Example: perfect shared-key encryption
 - A cyphertext can be decrypted only if the right key is known
 - The encryption key cannot be deduced from the cyphertext
 - An encrypted message is sufficiently redundant so that the decryption algorithm can detect whether or not it has succeeded



Why Dolev-Yao Models?

- Resistance against Dolev-Yao attackers means the protocol is free from big logical errors
 - i.e. errors that can be exploited even under the simplifying assumptions of the model
- Dolev-Yao models are simple and can be analysed automatically
- Good for first (and fast) sanity checks (before more accurate analyses)



Automating the Analysis

- Key points:
 - Undecidability of main security properties with unbounded number of sessions (Durgin et al, JCS, 2004)
 - NP-completeness of checking main security properties with bounded number of sessions (Rusinowitch & Turani, TCS 2003)
- Possible approaches:
 - Analyse protocols with few sessions (e.g. by state exploration)
=> a result is eventually obtained but errors can be missed
 - Analyse protocols with unbounded sessions (by automated procedures that may not terminate or may not give a response)

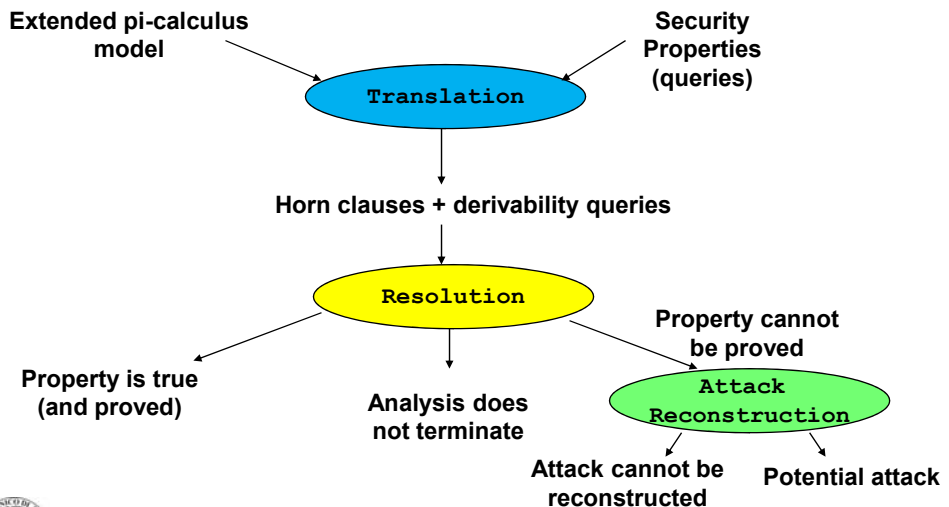


Proverif

- Tool for automated analysis of security protocols with Dolev-Yao models, developed by Bruno Blanchet (ENS, Paris)
- Protocol model expressed by a **process calculus** and then translated to a **logic program**
- Models unbounded sessions
- Automated resolution-based analysis of the logic program
- Note:
 - this tutorial just focuses on the main features of Proverif



Proverif Architecture



Specifying Protocols

- Input language:
 - pi calculus extended with cryptographic and data manipulation primitives
 - very close to applied pi-calculus (Abadi, Fournet, POPL'01)
- Each pi-calculus process models a protocol principal
 - Honest principals behave according to the protocol
 - Attacker can behave in any way



Extended Pi-Calculus: term Syntax

$M, N ::=$	terms
x, y, z	variable
a, b, c, k	name
$f(M_1, \dots, M_n)$	constructor application



Extended Pi-Calculus: process Syntax

$P, Q ::=$	processes	
$M\langle N \rangle.P$		output
$M(x).P$		input
0		nil
$P \mid Q$		parallel composition
$!P$		replication
$(\nu a)P$		restriction (new)
$\text{let } x=g(M_1, \dots, M_n) \text{ in } P \text{ else } Q$		destructor application
$\text{if } M = N \text{ then } P \text{ else } Q$		conditional
$\text{let } x=M \text{ in } P$		binding
$\text{event}(M).P$		event



Formal Semantics

- Process evolution defined operationally by a reduction relation on processes
- Destructor application defined by [rewriting rules](#)



Destructor Semantics

- Used to define the (ideal) properties of cryptographic and data manipulation primitives
- Example: modelling Shared-key encryption
 - Constructor: $\text{senc}(x,y)$ encrypts y under key x
 - Destructor: $\text{sdec}(x,y)$ decrypts y with key x
 - Rewrite rules: $\text{sdec}(x,\text{senc}(x,y)) \rightarrow y$
 - Proverif syntax:

```
fun senc/2.  
  reduc sdec(x, senc(x,y)) = y.
```



Process Semantics

- \mathcal{E} set of names
- \mathcal{P} multiset of processes
- If $\mathcal{E}=\{a_1,\dots,a_n\}$ and $\mathcal{P}=\{P_1,\dots,P_m\}$

semantic configuration \mathcal{E},\mathcal{P} corresponds to

$(\forall a_1)\dots(\forall a_n)(P_1 \mid \dots \mid P_m)$

- Initially, for process P , configuration is $\text{fn}(P), \{P\}$



Process Semantics

- $\mathcal{E}, \mathcal{P} \cup \{M \langle N \rangle . P, M(x) . Q\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{P, Q[N/x]\}$ (Red I/O)
- $\mathcal{E}, \mathcal{P} \cup \{0\} \rightarrow \mathcal{E}, \mathcal{P}$ (Red Nil)
- $\mathcal{E}, \mathcal{P} \cup \{P|Q\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{P, Q\}$ (Red Par)
- $\mathcal{E}, \mathcal{P} \cup \{!P\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{P, !P\}$ (Red Repl)
- $\mathcal{E}, \mathcal{P} \cup \{(v a)P\} \rightarrow \mathcal{E} \cup \{a'\}, \mathcal{P} \cup \{P[a'/a]\}$ with $a' \notin \mathcal{E}$ (Red Res)
- $\mathcal{E}, \mathcal{P} \cup \{\text{let } x = g(M_1, \dots, M_n) \text{ in } P \text{ else } Q\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{P[M'/x]\}$ (Red Destr1)
if $g(M_1, \dots, M_n) \rightarrow M'$
- $\mathcal{E}, \mathcal{P} \cup \{\text{let } x = g(M_1, \dots, M_n) \text{ in } P \text{ else } Q\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{Q\}$ (Red Destr2)
if there is no M' such that $g(M_1, \dots, M_n) \rightarrow M'$
- $\mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } P \text{ else } Q\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{P\}$ (Red Cond1)
- $\mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } P \text{ else } Q\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{Q\}$ (Red Cond2)
if $M \neq N$
- $\mathcal{E}, \mathcal{P} \cup \{\text{let } x = M \text{ in } P\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{P[M/x]\}$ (Red Bind)
- $\mathcal{E}, \mathcal{P} \cup \{\text{event}(M) . P\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{P\}$ (Red Event)



Example: Handshake Protocol

- Message 1 $S \rightarrow C:$ $\{\{k\}_{skS}\}_{pkC}$ k fresh
- Message 2 $C \rightarrow S:$ $\{s\}_k$

PS = $c(xpkC) . (v k) c \langle penc(xpkC, sign(skS, k)) \rangle .$
 $c(x) . \text{let } xs = sdec(k, x) \text{ in } 0$

PC = $c(y) . \text{let } y' = pdec(skC, y) \text{ in}$
if $checksign(xpkS, y') = ok$ then
let $xk = getmess(y')$ in
 $c \langle senc(xk, s) \rangle . 0$

$P = (v skS) (v skC) \text{ let } xpkS = pk(skS) \text{ in let } xpkC = pk(skC) \text{ in}$
 $c \langle xpkS \rangle . c \langle xpkC \rangle . (!PA \mid !PB)$



Specifying Properties: Secrecy

- **Intuitive property:** an attacker must not be able to get closed terms that are intended to be secret (e.g. names in the Handshake protocol)



Formal Specification of Secrecy

- S-Adversary: any closed process Q with $\text{fn}(Q) \subseteq S$
- Trace $T = \mathcal{E}, \rho_0 \rightarrow^* \mathcal{E}', \rho'$ **outputs** N iff T contains a reduction $\mathcal{E}, \rho \cup \{M \langle N \rangle . P, M(x). Q\} \rightarrow \mathcal{E}, \rho \cup \{P, Q[N/x]\}$ with $M \in S$
- Closed process P preserves the secrecy of N from S-Adversaries if
 $\forall S\text{-Adversary } Q, \forall T = \text{fn}(P) \cup S, \{P, Q\} \rightarrow^* \mathcal{E}', \rho'$
 T does not output N .



Translation into Horn Clauses (for secrecy verification)

- Definition of predicates:
 - $\text{attacker}(x)$ the attacker **may** have message x
 - $\text{message}(x,y)$ message y **may** appear on channel x
 - Remark: new names generated in different sessions (e.g. k in Handshake Protocol) are not differentiated
 - Partial differentiation is introduced by turning k into $k[xpkC]$
 - ⇒ Sessions that receive the same $xpkC$ are not differentiated
- General rule: a new name becomes function of all the messages received by the process before creating it



Horn Clauses for the Attacker

- For each $a \in S \cup \{b\}$ (b : fresh names created by attacker)
$$\text{attacker}(a[]) \quad (\text{Ri})$$
- For each public constructor $f(M_1, \dots, M_n)$
$$\text{attacker}(x_1) \wedge \dots \wedge \text{attacker}(x_n) \Rightarrow \text{attacker}(f(x_1, \dots, x_n)) \quad (\text{Rf})$$
- For each public destructor rule $g(M_1, \dots, M_n) \rightarrow M$
$$\text{attacker}(M_1) \wedge \dots \wedge \text{attacker}(M_n) \Rightarrow \text{attacker}(M) \quad (\text{Rg})$$
- $\text{message}(x,y) \wedge \text{attacker}(x) \Rightarrow \text{attacker}(y) \quad (\text{Rl})$
- $\text{attacker}(x) \wedge \text{attacker}(y) \Rightarrow \text{message}(x,y) \quad (\text{Rs})$



Horn Clauses for the Protocol

- For each message N the protocol can send on a channel M after having received messages N_1, \dots, N_n on channels M_1, \dots, M_n respectively

$\text{message}(M_1, N_1) \wedge \dots \wedge \text{message}(M_n, N_n) \Rightarrow \text{message}(M, N)$

- If all the channels are public, the clauses can be changed into

$\text{attacker}(N_1) \wedge \dots \wedge \text{attacker}(N_n) \Rightarrow \text{attacker}(N)$



Remark

- The number of times a message appears is ignored
 - The Horn clauses approximate the behaviour of the protocol with one that can send and receive each message an arbitrary number of times



Relating the two Models

Correctness (for secrecy):

- Let $R_{P,S}$ be the set of clauses modelling the protocol process P combined with S -Adversaries
- If there exist an S -Adversary Q and a trace $T = \text{fn}(P) \cup S, \{P, Q\} \rightarrow^* \mathcal{E}, \mathcal{P}'$ that outputs N , then the fact $\text{attacker}(N)$ is derivable from $R_{P,S}$

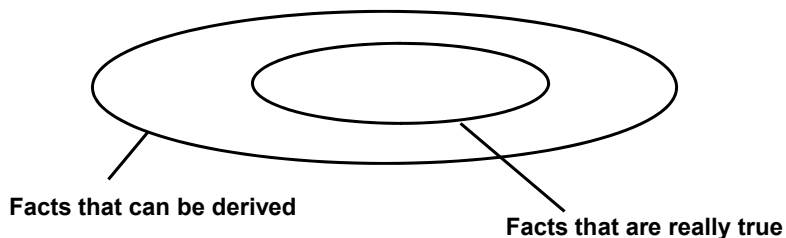
=> If one proves $\text{attacker}(N)$ is not derivable from $R_{P,S}$ one has proved P preserves the secrecy of N against S -Adversaries



Relating the two Models

- The logic theory described by the Horn clauses over-approximates the behavior of the real protocol
 - Freshness, repetition of send/receive

=> false positives are possible



Example

Process

(v privc)
(privc<s>. pubc<privc>. 0 | privc(x). 0)

preserves the secrecy of s against {pubc}-Adversaries

but Proverif cannot prove it, because the Proverif model corresponds to:

(v privc)
(! privc<s>. pubc<privc>. 0 | ! privc(x). 0)

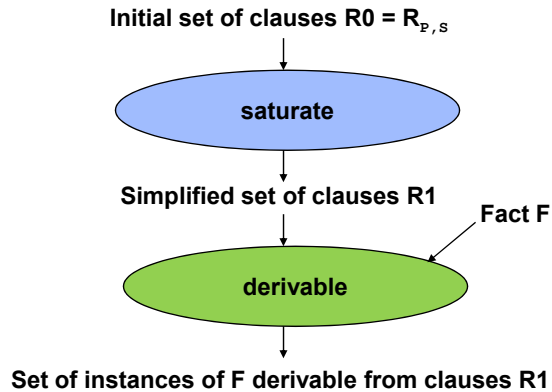


Resolution

- Approximations are key factor for automatic and efficient solution algorithm
- However, a standard Prolog engine would not terminate
 - Main problem: clauses like
$$\text{attacker}(\text{senc}(x,y)) \wedge \text{attacker}(x) \Rightarrow \text{attacker}(y)$$
generate bigger and bigger facts by SLD-resolution
- Proverif solves the problem by a custom (and clever) resolution algorithm (with free selection)



Resolution Algorithm



Phase 1 : Saturation

- Simplifies the set of clauses by
 - applying resolution to pairs of clauses with free selection (generate combined clauses $C \circ_{F_0} C'$ when F_0 is selected)
 - simplifying redundant clauses (function simplify())
 - eliminating subsumed clauses (function elim())
- Note: hypothesis attacker(x) is never selected



Saturation Algorithm

saturate(R_0) :

1. $R \leftarrow \emptyset$
2. $\forall C \in R_0, \quad R \leftarrow \text{elim}(\text{simplify}(C) \cup R)$
3. Repeat until a fixpoint is reached
 - $\forall C \in R$ such that the conclusion is selected
 - $\forall C' \in R, \forall F_0$ such that F_0 is selected and $C \circ_{F_0} C'$ is defined
 - $R \leftarrow \text{elim}(\text{simplify}(C \circ_{F_0} C') \cup R)$
4. Return $\{C \in R \mid \text{no hypothesis is selected in } C\}$



Simplifications

- Elimination of tautologies
- Elimination of duplicate hypotheses
- Elimination of hypotheses $\text{attacker}(x)$ when x does not appear elsewhere (always fulfilled)
- Elimination of hypotheses $\text{attacker}(N)$ when N is assumed to be secret (secrecy assumptions)



Correctness of Saturation

- Let F_{not} be a set of facts the user claims to be non-derivable (secrecy assumptions)
- If $\forall F' \in F_{\text{not}}$ no instance of F' is derivable from $\text{saturate}(R_0)$

then a closed fact F is derivable from R_0 iff F is derivable from $\text{saturate}(R_0)$



Phase 2 : Derivation Search

- Searches a derivation of a given fact F using a set of rules R by backward depth-first search:
 - $\text{derivable}(F, R) = \text{deriv}(F \Rightarrow F, \emptyset, R)$
 - $\text{deriv}(C, R', R) = \emptyset$ if $\exists C' \in R$ that subsumes C
 - $\text{deriv}(\Rightarrow F, R', R) = \{F\}$
 - $\text{deriv}(C, R', R) = \bigcup \{ \text{deriv}(\text{simplify}'(C' \circ_{F_0} C), \{C\} \cup R', R) \mid C' \in R, F_0 \text{ is selected and } C \circ_{F_0} C' \text{ is defined} \}$



Correctness of Derivation Search

- If $\forall F'' \in F_{\text{not}} \text{ derivable}(F'', R1) = \emptyset$

then a closed instance F' of F is derivable from $R1$ iff $\exists F'' \in \text{derivable}(F, R1)$ and a substitution σ such that $\sigma F'' = F'$



Summary of Resolution Algorithm

- Let F_{not} be a set of facts the user claims to be non-derivable (secrecy assumptions)
- $\text{solve}_{P,S}(F)$:
 1. $R1 = \text{saturate}(R_{P,S})$
 2. $\forall F' \in F_{\text{not}}$, if $\text{derivable}(F', R1) \neq \emptyset$, terminate with error
 3. Return $\text{derivable}(F, R1)$



Extension to Equational Theories

- Proverif can handle some equational theories
- Useful for modelling the special properties of some cryptosystems
- Example: Diffie-Hellman key generation
 - We need to model the equality $g^{x^y} \bmod p = g^{y^x} \bmod p$
 - We can define two constructors $f(v,w) = w^v \bmod p$,
 $f'(z)=g^z \bmod p$ and the equality $f(y,f'(x)) = f(x,f'(y))$



Extension to Equational Theories

- Proverif automatically translates equations into (non-deterministic) rewrite rules for function symbols
 - Example: for Diffie-Hellman key generation functions the equality $f(y,f'(x)) = f(x,f'(y))$ is translated into the rewrite rules
$$f(y,f'(x)) \rightarrow f(x,f'(y)) \qquad f(x,y) \rightarrow f(x,y)$$
- The solving algorithm is adapted to perform unification modulo the equational theory by reducing it to syntactic unification via the rewriting rules
- The solving algorithm may not terminate (it has been shown to work well with the Diffie-Hellman equality)



Correspondence Properties

- Specify order relationships that bind trace events
- Can be used to specify authentication (e.g. agreement)



Example: Authentication in the Handshake Protocol

```
PS= c(xpkC). (v k)
     event(bS(xpkS,xpkC,k)).
     c<penc(xpkC, sign(skS, k))>.
     c(x). let xs=sdec(k,x) in 0
```

```
PC= c(y). let y'=pdec(skC,y) in
     if checksign(xpkS, y')=ok then let xk=getmess(y') in
     event(eC(xpka,xpkb,xk)).
     c<senc(xk,s)>. 0
```

- In each trace, if event $eC(x,y,z)$ occurs, event $bS(x,y,z)$ must have occurred before
 $(ev:eC(x,y,z) ==> ev:bS(x,y,z)).$



Verification of Correspondences

- Logic theory is extended with new predicates:
 - $\text{event}(M)$ $\text{event}(M)$ **may** have been executed
 - $\text{m-event}(M)$ $\text{event}(M)$ **must** have been executed
- Correspondence $\text{ev:eC}(x,y,z) \implies \text{ev:bS}(x,y,z)$ can be proved by proving that in each trace $\text{event}(\text{eC}(x,y,z))$ **may** have been executed only if $\text{event}(\text{bS}(x,y,z))$ **must** have been executed



Verification of Correspondences

- Fresh names generated in different sessions now are differentiated (by instrumenting the protocol with session identifiers)
 - k is turned into $k[xpkC,i]$ where i identifies the session
- Clauses are generated for event actions: For each $\text{event}(N)$ the protocol can execute after having received messages N_1, \dots, N_n on channels M_1, \dots, M_n respectively

$\text{message}(M_1, N_1) \wedge \dots \wedge \text{message}(M_n, N_n) \implies \text{event}(N)$



Verification of Correspondences

- An m-event(N') hypothesis is added to a clause for message output when event event(N') occurs before the output

– $\text{message}(M_1, N_1) \wedge \dots \wedge \text{message}(M_n, N_n) \wedge \text{m-event}(N') \Rightarrow \text{message}(M, N)$

- m-event(N) predicates occur only as hypotheses and are never selected during resolution



Modified Derivation Search

- $\text{derive}(F, R)$ now returns a **set of clauses** without selectable hypotheses (a clause may have m-event() hypotheses)
- The clauses $H \Rightarrow F' \in \text{derivable}(F, R)$ are derived from the rules in R and have F' that is an instance of F
 - $\text{deriv}(C, R', R) = \emptyset$ if $\exists C' \in R$ that subsumes C
 - $\text{deriv}(C, R', R) = \{C\}$ if C has no selectable hypothesis
 - $\text{deriv}(C, R', R) = U\{ \text{deriv}(\text{simplify}'(C' \circ_{F_0} C), \{C\}UR', R) \mid C' \in R, F_0 \text{ is selected and } C \circ_{F_0} C' \text{ is defined} \}$



Example

- Verification of $ev:e(x_1, \dots, x_n) \implies ev:e'(x_1, \dots, x_n)$
 - Let $R_{P,S}$ be the set of clauses modelling the protocol process P combined with S -Adversaries
 - If $\forall C \in \text{solve}_{P,S}(\text{event}(e(x_1, \dots, x_n)))$ such that $C = H \rightarrow \text{event}(e(p_1, \dots, p_n))$ we have $m\text{-event}(e(p_1, \dots, p_n)) \in H$, then P satisfies the correspondence $ev:e(x_1, \dots, x_n) \implies ev:e'(x_1, \dots, x_n)$ against S -Adversaries



Injective Correspondences

- $ev:e(x_1, \dots, x_n) \implies ev:e'(x_1, \dots, x_n)$ is true even when the same execution of event $e'(x_1, \dots, x_n)$ corresponds to more executions of event $e(x_1, \dots, x_n)$
- Injective correspondence
 $ev:e(x_1, \dots, x_n) \implies evinj:e'(x_1, \dots, x_n)$

requires that each occurrence of event $e(x_1, \dots, x_n)$ corresponds to a **distinct** occurrence of event $e'(x_1, \dots, x_n)$

- In order to prove injective correspondences Proverif adds **session identifiers** to $m\text{-event}()$ predicates



General Correspondences

- Correspondences can be combined together to form more complex queries:
 - $ev:e(x1,x2) \implies (evinj:e2(x1,x2) \implies evinj:e1(x1))$
 - $ev:e(x1,x2) \implies (evinj:e2(x1,x2) \implies evinj:e1(x1)) \mid (evinj:e4(x1,x2) \implies evinj:e3(x1))$
- The query
 - $ev:e(x1,x2)$ means event $e(x1,x2)$ is never executed



Observational Equivalence

- Informally, two processes are observationally equivalent if an attacker cannot distinguish them.
- Example: $(\nu k) M\langle H(k)\rangle.0 \approx (\nu k) M\langle k\rangle.0$
- Example: $(\nu k) M\langle H(k),k\rangle.0 \not\approx (\nu k) M\langle k,k\rangle.0$

because they can be distinguished by the attacker

$M(x)$. let $l = \text{fst}(x)$ in let $r = \text{snd}(x)$ in if $l = H(r)$ then success else fail



Strong Secrecy

- Observational equivalence can be used to give a stronger definition of secrecy, closer to non-interference
- Intuitive meaning: strong secrecy of a secret is preserved when an adversary cannot see any difference when the secret changes
- Strong secrecy takes into account implicit flows (e.g. different observable behaviours depending on a test on the secret). Example:

let $x = \text{sdec}(s, k)$ in $M \langle N \rangle . 0$ else $M \langle N' \rangle . 0$



Strong Secrecy: Formal Definition

- Process P preserves the strong secrecy of its free variables iff for all closed substitutions σ and σ' of domain $\text{fv}(P)$, $\sigma P \approx \sigma' P$



Observational Equivalence: Formal Definition

- C (evaluation context): an expression built from $[], C|P, P|C, (\nu a)C$
- $P \Downarrow M$ (P emits on M): $P \rightarrow^* C[M \langle N \rangle . Q]$ and C does not bind M
- \approx (Observational equivalence): the largest symmetric relation R on closed processes such that $P R Q$ implies
 - If $P \Downarrow M$ then $Q \Downarrow M$
 - If $P \rightarrow P'$ then $\exists Q'$ such that $Q \rightarrow^* Q'$ and $P' R Q'$
 - $C[P] R C[Q]$ for any C



Verification of Observational Equivalence

- Proverif supports verification of observational equivalence between processes that differ only by some terms
- Example: $(\nu k) (\nu k') M \langle H(\text{choice}[k, k']) \rangle . 0$

can be defined to verify that

$$(\nu k) M \langle H(k) \rangle . 0 \approx (\nu k') M \langle H(k') \rangle . 0$$

i.e. that $(\nu k) M \langle H(k) \rangle . 0$ preserves strong secrecy of k



Verification of Observational Equivalence

- Proverif tries to prove a stronger condition that implies observational equivalence
 - Intuitively: each reduction step proceeds uniformly for all the values of choice terms
- The logic theory is extended by introducing the predicate $\text{testunif}(x,y)$
 - Intuitively: $\text{testunif}(M,M')$ is true if the existence of a substitution that unifies M and M' depends on the values of the choice terms



Termination

- The saturation algorithm may not terminate
- Termination has been proved for a class of tagged protocols (Blanchet, Podelski, TCS 2005):
 - Limited set of primitives (including all the main ones)
 - No private channels
 - Crypto functions always applied to tagged data (with different tags for each occurrence of each function)
 - Tags always checked on application of destructors
 - No else in destructor applications
 - Atomic keys



Example: Handshake Protocol

- Message 1 $S \rightarrow C:$ $\{T0, \{T1, k\}_{skS}\}_{pkC}$ k fresh
- Message 2 $C \rightarrow S:$ $\{T2, s\}_k$

PS= $c(xpkC). (v k) c\langle(T0, penc(xpkC, sign(skS, k)))\rangle.$
 $c(x). let y = sdec(k, x) in let fst(y) = T2 in xs = snd(xst) in 0$

PC= $c(y). let y' = pdec(skC, y) in let fst(y') = T0 in$
 $let y'' = snd(y') in$
 $if checksign(xpkS, y'') = ok then$
 $let z = getmess(y'') in let fst(z) = T1 in let xk = snd(z) in$
 $c\langle senc(xk, s) \rangle. 0$



Attack Reconstruction

- Let us assume we have found a derivation D of a fact F that prevents building the desired proof
- Attack Reconstruction looks for *an attack trace compatible with D* .
- Method:
 - Use a trace semantics restricted (i.e. driven) by D
 - Exhaustively search an attack in the (finite) set of traces of the restricted semantics



Attack Reconstruction

- If Attack Reconstruction finds an attack trace T , then T is a real attack (counterexample found).
 - Does not work with equivalence verification (e.g. strong secrecy)
 - For injective correspondences the reconstructed trace is not necessarily a real attack
- If no attack trace is found by Attack Reconstruction on found derivations, an attack (not included in the restricted semantics) may still exist



References

- **Verification of Secrecy**
 - M. Abadi, B. Blanchet, “Analyzing Security Protocols with Secrecy Types and Logic Programs”, JACM 52(1):102-146 (Jan 2005)
- **Verification of Correspondences**
 - B. Blanchet, “Automatic Verification fo Correspondences for Security Protocols”, JCS, 17(4):363-434 (Dec 2009).
- **Verification of Strong Secrecy and Observational Equivalences and Support for Equational Theories**
 - B. Blanchet, M. Abadi, C. Fournet, “Automated Verification of Selected Equivalences for Security Protocols”, Journal of Logic and Algebraic Programming, 75(1):3-51 (Feb-Mar 2008).



References

- **Termination for Tagged Protocols**

- B. Blanchet, A. Podelski, “Verification of Cryptographic Protocols: Tagging Enforces Termination”, Theoretical Computer Science, 333(1-2):67-90 (March 2005).

- **Attack Reconstruction**

- X. Allamigeon, B. Blanchet, “Reconstruction of attacks against cryptographic protocols”, Proc. 18th IEEE CSFW (2005), pp. 140-154.

