Verifying security protocols with ProVerif and StatVerif

Mark D. Ryan
University of Birmingham

FOSAD summer school, Bertinoro, Italy
3–7 September 2012

L1: Security protocols and verification
L2: ProVerif language, tool, theory
L3: Example: electronic voting
L4: Example: stateful systems

Outline

What are security protocols?

Security protocols (also known as cryptographic protocols) are distributed procedures that employ cryptography to achieve a security goal.

Example participating agents:

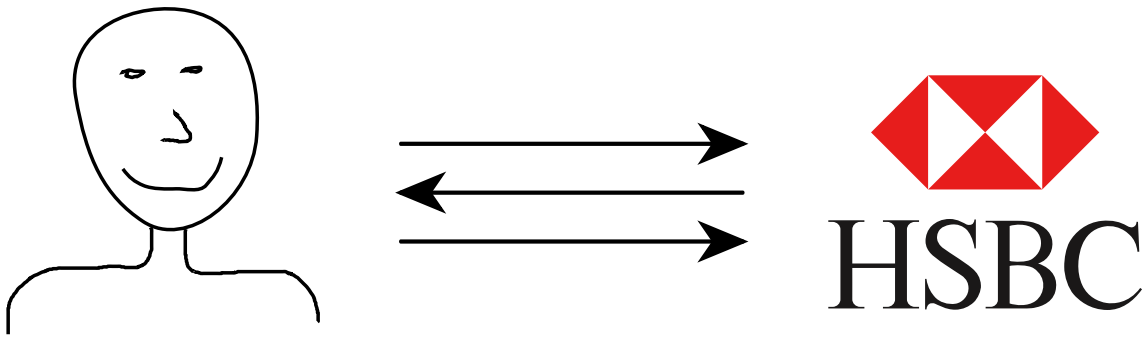
- Client *and* Server
- Application *and* TPM
- VM_1 *and* VMM
- Voter *and* Collector
- Alice *and* Bob

Example security goals:

- Authentication
- Key agreement
- Secure communication
- Privacy
- Attestation

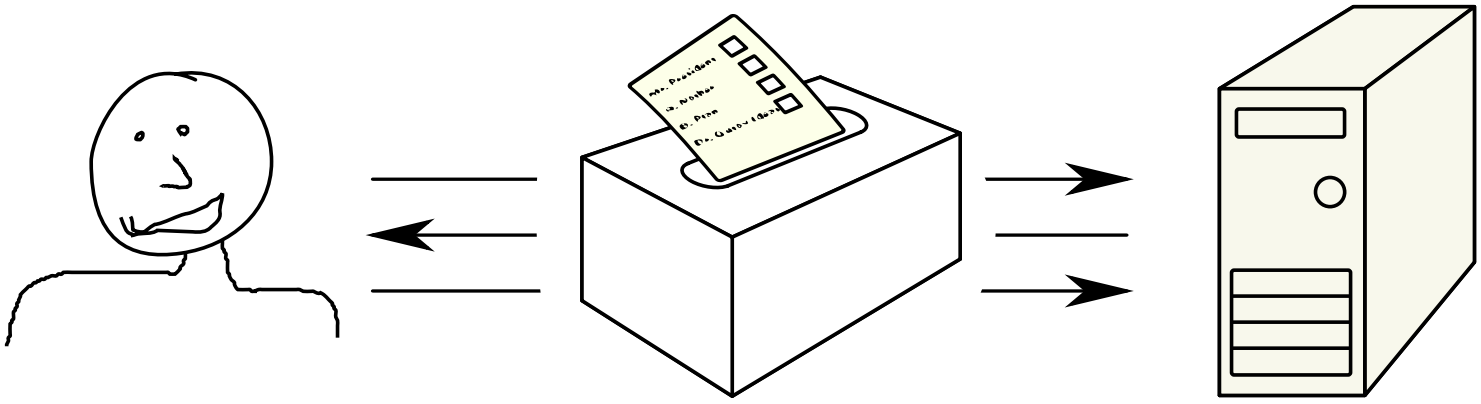
Protocols are usually simple, but often subtle. That makes them ideal for automated reasoning.

Example: customer $\leftrightarrow$ bank



- Mutual authentication
  - Q: How does the bank authenticate itself to the customer?
- Non-repudiation

Example: voter $\leftrightarrow$ administrator

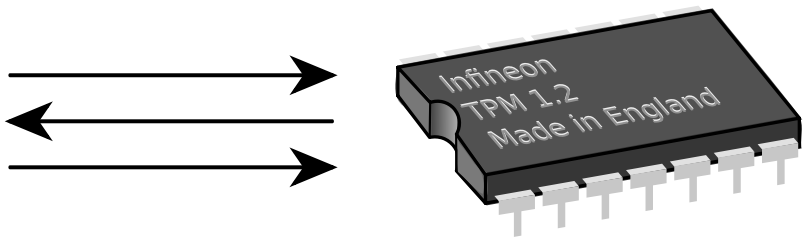


- Mutual authentication
- Incoercibility, verifiability
  - Anything else?

Example: user process $\leftrightarrow$ TPM

```
#include "stdio.h"
...

int main(int argc, char **argv)
{
     ...
     y = tpm_decrypt(tpm, x);
     ...
     return 0;
}
```



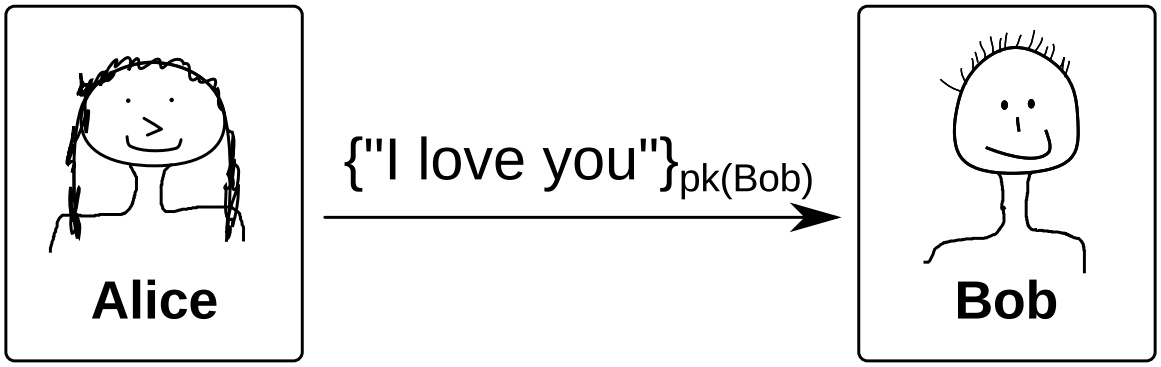
- Mutual authentication
  - In a stateful context

Outline

Approaches to verifying protocols

Computational approach (a.k.a. "provable security")	**Symbolic approach** (a.k.a. "formal methods security")
• data are bitstrings • proof is reduction to a "hardness assumption" (e.g. the RSA assumption, which expresses hardness of factorisation)	• data are terms in a calculus • proof is a proof in that calculus

Computational approach example



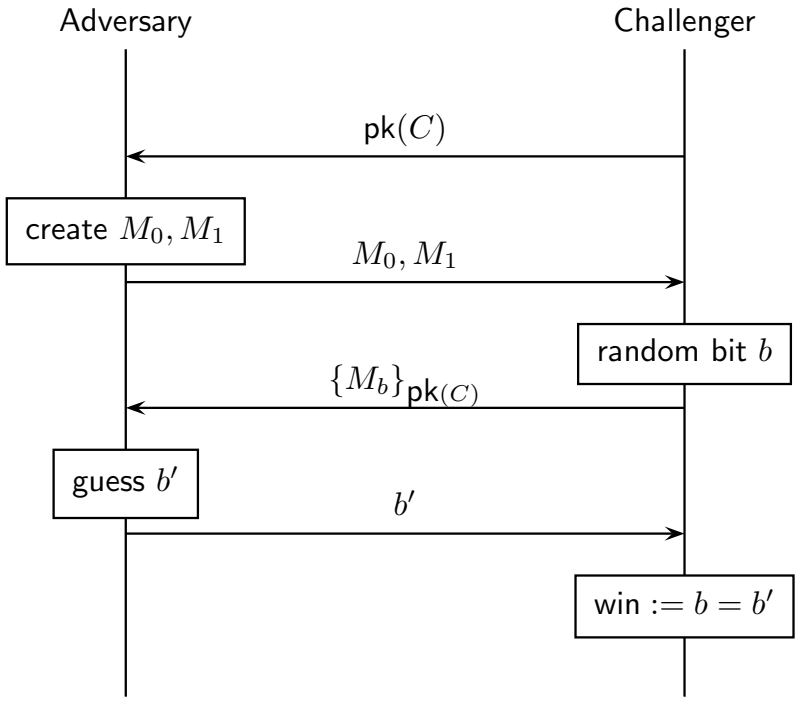
They've decided to use RSA-OAEP, so Bob has created a secret key (d, n) and a public key (e, n) such that $n = p \times q$; p, q are prime; and $d \times e = 1 \pmod{\phi(n)}$. Alice chooses a random r and sends $(x \,||\, y)^e \pmod{n}$, where

$$x = \text{“I love you”}00\ldots0 \oplus G(r)$$
$$y = H(x) \oplus r$$

and G, H are suitable hash functions.

Definition

The pubic key encryption scheme $\{\cdot\}_{pk(\cdot)}$ is IND-CPA secure if the adversary has no better than 0.5 chance of winning this game.

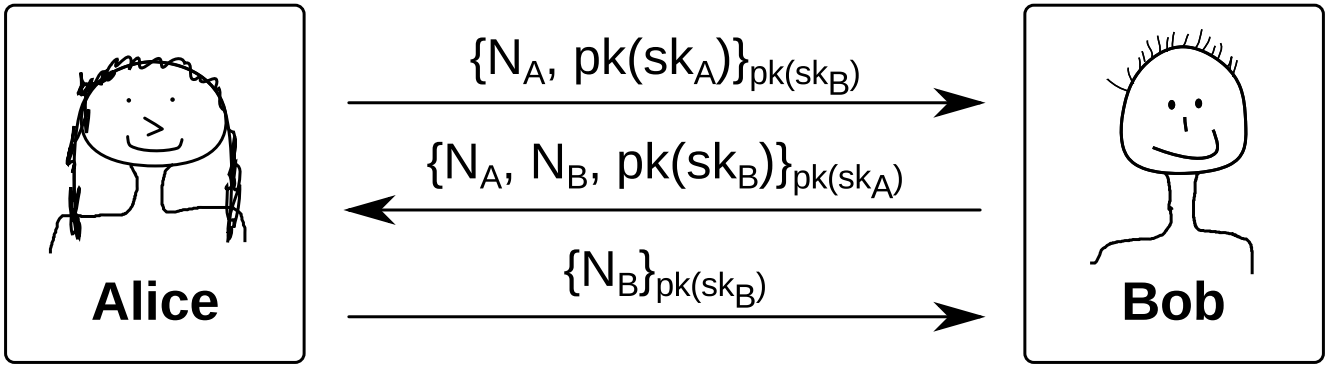


Theorem (Random oracle model)

If the RSA assumption holds (roughly, prime factorisation is hard) then RSA-OAEP is IND-CPA secure. (In fact, it's IND-CCA2 secure.)

Symbolic approach example

Consider the Needham-Schroeder-Lowe protocol.



where N_A and N_B are nonces chosen by Alice and Bob respectively, and Alice and Bob have the correct values for each other's public keys.

We don't know exactly what the encryption scheme is but we assume the functions $enc(\cdot,\cdot)$, $dec(\cdot,\cdot)$ and $pk(\cdot)$ which are related by the equation

$$dec(k, enc(pk(k), x)) = x$$

and no other equation.

Consider the term algebra $(N, pk, enc, dec, \langle\cdot,\cdot\rangle, fst, snd)$ where N is a set of names, $A, B, sk_A, sk_B \in N$, and there are three equations, namely:

$$fst(\langle x, y\rangle) = x$$
$$snd(\langle x, y\rangle) = y$$
$$dec(x, enc(pk(x), y)) = y$$

Suppose Alice is an oracle that, on input x, outputs $enc(y, z)$ where $y = snd(snd(dec(sk_A, x)))$ and $z = fst(snd(dec(sk_A, x)))$, and Bob is an oracle that, on input x, chooses a random r and outputs $enc(y, z)$ where $y = snd(dec(sk_B, x))$ and $z = (fst(dec(sk_B, x)), r, pk(sk_B))$. Also, Alice willingly chooses a value r and creates $enc(x, (r, pk(sk_A)))$ for any x at any time.
Suppose an attacker can:

- intercept and inject messages;
- apply any function to any message he knows;
- use the oracles Alice and Bob any number of times in any way;
- knows names $a, b, c, \ldots, A$ and B but not sk_A and sk_B.

Claim

- The attacker can't obtain the names sk_A, sk_B, N_A, N_B.
- Alice successfully executes her protocol iff Bob successfully executes his protocol, and on the same data.

Comparing computational and symbolic approaches

- Which do you find more convincing?
- Which might be more scalable or automatable?

Computational vs. symbolic approaches

Computational	Symbolic
more concrete	more abstract
more exact	more idealistic
manual	more automatable
small scale	larger scale

Two views of verification

Provable security vs. Formal methods

- Provable security provides stronger promises
- But, *"proofs are so turgid that other specialists don't even read them"* [KoblitzMenezes'04]
- Formal methods are simpler, specifications are nicer and automated support is available, but it is more abstract
- Both methods are based on a model, and therefore may fail to detect certain kinds of attack
- There is a gap between the abstract formal model and the actual implementation

Reconciling two views of cryptography

- Computational soundness: in some circumstances, one can prove that a proof in the symbolic model is sound in the computational model.

Outline

Symbolic approach: some methods/tools

ProVerif	B. Blanchet, and others, 2001-
AVISPA	Large EU project and team, 2005-
Scyther	C. Cremers, 2008-
Casper/FDR	G. Lowe, 1998
AKISS	Kremer, Chadha, 2012-
Athena	D. Song, 1999
NRL	C. Meadows, 1994
Isabelle/HOL	L. Paulson, G. Bella, 1998-

ProVerif syntax: terms

$L, M, N, \ldots ::=$

$a, b, c, \ldots$	name
$x, y, z, \ldots$	variable
$f(M_1, \ldots, M_l)$	function application

Equational theory

Suppose we have defined nullary function *ok*, unary function *pk*, binary functions *dec*, *senc*, *sdec*, *sign* and ternary functions *enc* and *checksign*.

$$\begin{aligned}
&\text{equation } \mathit{sdec}(x, \mathit{senc}(x, y)) &&= y \\
&\text{equation } \mathit{dec}(x, \mathit{enc}(\mathit{pk}(x), r, y)) &&= y \\
&\text{equation } \mathit{checksign}(\mathit{pk}(x), \mathit{sign}(x, y)) &&= \mathit{ok} \\
&\text{equation } \mathit{getmess}(\mathit{sign}(x, y)) &&= y
\end{aligned}$$

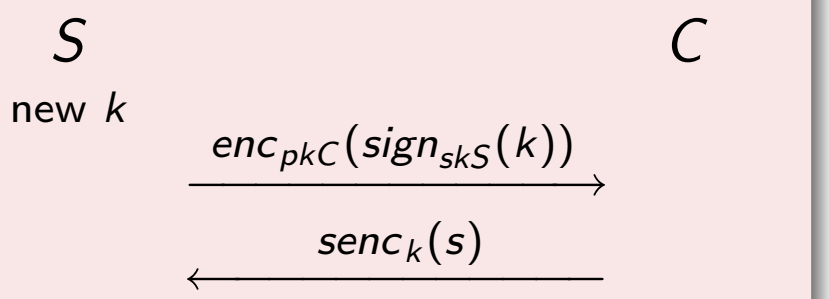
ProVerif syntax: processes

$P, Q, R, \ldots ::=$

0	null process
$P \mid Q$	parallel composition
$!P$	replication
$new\ n; P$	name restriction
$in(M, x); P$	message input
$out(M, N); P$	message output
$if\ M = N\ then\ P\ else\ Q$	conditional
$let\ x = g(M_1, \ldots, M_n)\ in\ P\ else\ Q$	destructor application

Example: handshake protocol

Handshake protocol



C knows S's public key. S is willing to talk to any C (does not know their public keys in advance). They want to agree a session key; they communicate on a channel that is controlled by the attacker.

Intended properties:

1. Secrecy: The value s is known only to $\mathcal{C}$ and $\mathcal{S}$.
2. Authentication of $\mathcal{S}$: if $\mathcal{C}$ reaches the end of the protocol with session key k, then $\mathcal{S}$ proposed k for use by $\mathcal{C}$.
3. Authentication of $\mathcal{C}$: if $\mathcal{S}$ reaches the end of the protocol and she believes she has session the key k with $\mathcal{C}$, then $\mathcal{C}$ was indeed her interlocutor and she has session k.

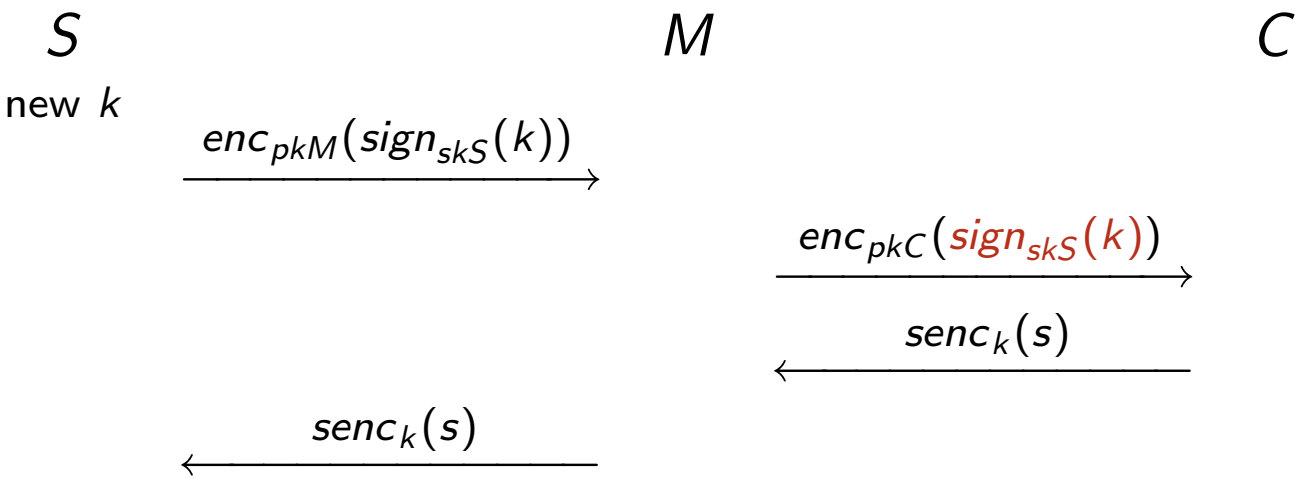
Attacker model

We model a very powerful attacker, with "Dolev-Yao" capabilities:

- it completely controls the communication channels, so it is able to record, alter, delete, insert, redirect, reorder, and reuse past or current messages, and inject new messages. (The *network* is the attacker.)
- manipulate data in arbitrary ways, including applying crypto operations provided it has the necessary keys.
- It controls dishonest participants.

"It's always better to assume the worst. Assume your adversaries are better than they are. Assume science and technology will soon be able to do things they cannot yet. Give yourself a margin for error. Give yourself more security than you need today." - Bruce Schneier

Handshake protocol attack



Intended properties:

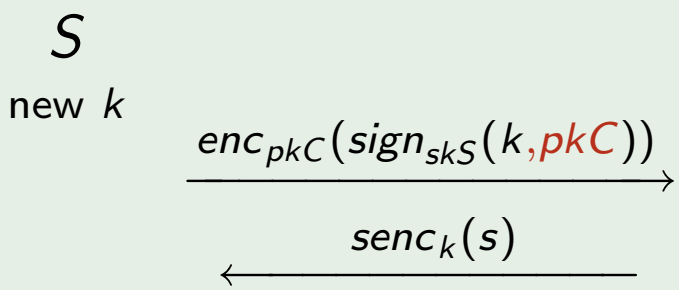
1. Secrecy: The value s is known only to $\mathcal{C}$ and $\mathcal{S}$.
2. Authentication of $\mathcal{S}$: if $\mathcal{C}$ reaches the end of the protocol with session key k, then $\mathcal{S}$ proposed k for use by $\mathcal{C}$.
3. Authentication of $\mathcal{C}$: if $\mathcal{S}$ reaches the end of the protocol and she believes she has session the key k with $\mathcal{C}$, then $\mathcal{C}$ was indeed her interlocutor and she has session k.

Handshake protocol fixed

The attack is avoided by making the package the initiator sends include the identity of the respondent.

The three properties hold of the revised protocol, but not for the original one.

Revised handshake protocol

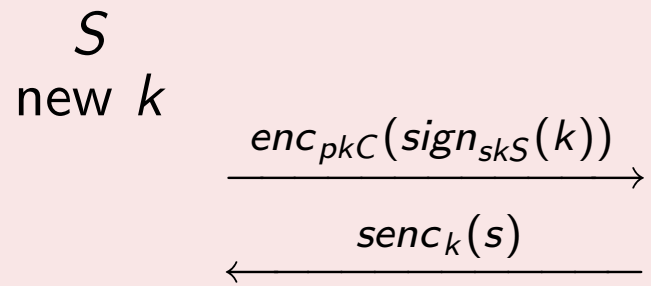


Our aim is to be able to automatically establish these facts.

Coding protocols as processes

Original handshake protocol:

Handshake protocol



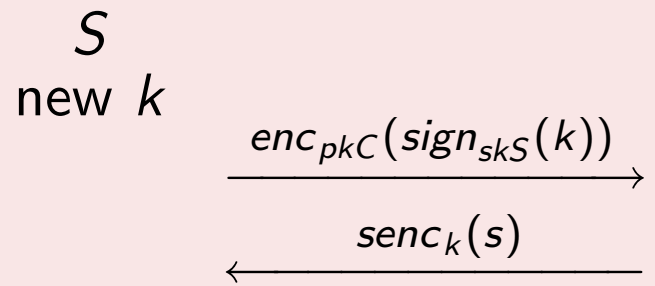

```
let Server =
  in (ch, pkC');
  new k;
  out (ch, enc(pkC', sign(skS, k)));
  in (ch, m);
  let x = sdec(k, m) in
  0.
```

Coding protocols as processes

Original handshake protocol:

```
let Client =
  in (ch, m);
  let sig = dec(skC, m) in
  if checksign(pkS, sig) = ok then
  let k' = getmess(sig) in
  new s;
  out (ch, senc(k', s)).
```

Handshake protocol



The handshake protocol in full - 1

```
free ch.

(* Public key cryptography *)
fun pk/1.
fun enc/2.  fun dec/2.
equation dec(x, enc(pk(x), y)) = y.

(* Signatures *)
fun sign/2.  fun checksign/2.  fun getmess/1.  fun ok/0.
equation checksign(pk(x), sign(x,y)) = ok.
equation getmess(sign(x,y)) = y.

(* Shared-key cryptography *)
fun senc/2.  fun sdec/2.
equation sdec(senc(x,y),x) = y.

query attacker:s.
```

The handshake protocol in full - 2

```
let Server =
    in (ch, pkC');
    new k;
    out (ch, enc(pkC', sign(skS, k)));
    in (ch, m);
    let x = sdec(k, m) in 0.

let Client =
    in (ch, m);
    let sig = dec(skC, m) in
    if checksign(pkS, sig) = ok then
    let k' = getmess(sig) in
    new s;
    out (ch, senc(k', s)).

process
    new skC; new skS;
    let pkC = pk(skC) in out (ch, pkC);
    let pkS = pk(skS) in out (ch, pkS);
    (!Client) | (!Server)
```

Output from ProVerif

```
Starting query not attacker:s_19[m = v_97,!1 = v_98]
goal reachable: attacker:x_297 -> attacker:s_19[m = enc(pk(skC_12[]),
    sign(skS_13[],k_21[pkC' = pk(x_297),!1 = @sid_298])),!1 = v_299]

new skC_12 creating skC_12_327 at {1}
new skS_13 creating skS_13_328 at {2}
out(ch, pk(skC_12_327)) at {4}
out(ch, pk(skS_13_328)) at {6}
in(ch, pk(a_324)) at {15}in copy a_325
new k_21 creating k_21_329 at {16}in copy a_325
out(ch, enc(pk(a_324),sign(skS_13_328,k_21_329))) at {17}in copy a_325
in(ch, enc(pk(skC_12_327),sign(skS_13_328,k_21_329))) at {8}in copy a_326
new s_19 creating s_19_330 at {12}in copy a_326
out(ch, senc(getmess(dec(skC_12_327,enc(pk(skC_12_327),
    sign(skS_13_328,k_21_329)))),s_19_330)) at {13}in copy a_326
The attacker has the message s_19_330.

A trace has been found.
RESULT not attacker:s_19[m = v_97,!1 = v_98] is false.
```

Three kinds of property

- Weak secrecy
  (attacker cannot derive a value)
- Correspondence properties
  (e.g. authentication properties)
- Equivalence properties
  (attacker can't detect difference between instances of a system)
  — strong secrecy

Correspondence properties I

By annotating processes with events $\overline{f}\langle M \rangle$, relationships between the order of events and their parametrisation M can be studied.

Annotated server process

```
let Server =
 in (ch, pkC');
 new k;
 event startedS(pair(pkC',k));
 out (ch, enc(pkC',sign(skS,k)));
 in (ch, m);
 if pkC' = pkC then
 event compS(k).
```

- event startedS(pair(pkC',k)) means *Server* started the protocol with *Client* having pub key pkC', and k is the session key.
- event compS(k) means *Server* completed the protocol with session key k.

Since event $compS(k)$ is under a conditional it can only occur when the protocol completes with *Client*.

Correspondence properties II (Handshake protocol)

```
let Server =
  in (ch, pkC');
  new k;
  event startedS(pair(pkC',k));
  out (ch, enc(pkC', sign(skS, k ) ));
  in (ch, m);
  if pkC' = pkC then
  event compS(k).

let Client =
  in (ch, m);
  let sig = dec(skC, m) in
  if checksign(pkS, sig) = ok then
  let k' = getmess(sig) in
  event startedC(k');
  out (ch, senc(k', s));
  event compC(pair(pkC,k')).
```

Authentication properties

Authentication of server to client:

$$\mathtt{ev}:\mathtt{compC}(\mathtt{x},\mathtt{y}) ==> \mathtt{ev}:\mathtt{startedS}(\mathtt{x},\mathtt{y}).$$

Authentication of client to server:

$$\mathtt{ev}:\mathtt{compS}(\mathtt{y}) ==> \mathtt{ev}:\mathtt{startedC}(\mathtt{y}).$$

Equivalence properties

```
free ch.
fun pk/1. fun enc/3. fun dec/2.
equation dec(x, enc(pk(x), y, z)) = z.

let Alice =
    new n;
    let msg = choice[enc(pk(skB), n, I_love_you), n] in
    out (ch, msg).

process
    new skB; out (ch, pk(skB)); !Alice
```

Process calculus semantics

Applied pi calculus

Mobile values, new names and secure communication
[Abadi/Fournet 2001]

ProVerif calculus (theory)

An efficient cryptographic protocol verifier based on Prolog rules
[Blanchet 2001]

Automatic verification of correspondences for security protocols
[Blanchet 2008]

ProVerif calculus (tool)

The ProVerif tool

Semantics of ProVerif calculus

Semantic configuration: $E, \mathcal{P}$

E: a finite set of names currently in use
$\mathcal{P}$: a finite multiset of closed processes

Transitions $\quad E, \mathcal{P} \rightarrow E', \mathcal{P}'$

Semantics of ProVerif calculus

$E, \mathcal{P} \cup \{0\} \rightarrow E, \mathcal{P}$ (Red Nil)

$E, \mathcal{P} \cup \{!P\} \rightarrow E, \mathcal{P} \cup \{P, !P\}$ (Red Repl)

$E, \mathcal{P} \cup \{P \mid Q\} \rightarrow E, \mathcal{P} \cup \{P, Q\}$ (Red Par)

$E, \mathcal{P} \cup \{\nu a.P\} \rightarrow E \cup \{a'\}, \mathcal{P} \cup \{P\{^{a'}/_{a}\}\}$ (Red Res)

where $a' \notin E$

$E, \mathcal{P} \cup \{\overline{N}\langle M\rangle.Q, N(x).P\} \rightarrow E, \mathcal{P} \cup \{Q, P\{^{M}/_{x}\}\}$ (Red I/O)

$E, \mathcal{P} \cup \{\mathit{let}\ x = g(M_1, \ldots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{P\{^{M'}/_{x}\}\}$

if $g(M_1, \ldots, M_n) \rightarrow M'$ (Red Destr 1)

$E, \mathcal{P} \cup \{\mathit{let}\ x = g(M_1, \ldots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{Q\}$ (Red Destr 2)

if there exists no M' such that $g(M_1, \ldots, M_n) \rightarrow M'$

$E, \mathcal{P} \cup \{\mathit{if}\ M = M\ \mathit{then}\ P\ \mathit{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{P\}$ (Red Cond 1)

$E, \mathcal{P} \cup \{\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{Q\}$ (Red Cond 2)

if $M \neq N$

$E, \mathcal{P} \cup \{\mathit{event}(M).P\} \rightarrow E, \mathcal{P} \cup \{P\}$ (Red Event)

Predicate *attacker*(·)

The ProVerif tool performs logical reasoning (called resolution) on logic formulae (called Horn clauses) that describe knowledge obtained by the attacker.
The principal predicate in the Horn clauses is

$$attacker(\cdot)$$

attacker:M means that the attacker has the means to botain the term M. The attacker's means are:

- his own private computation, represented by applying functions to terms he already has;
- the "services" (or oracle) represented by protocol participants, which accept data as input and produce new data as output.

Computations by attacker

Declaring the constructors and destructors

```
fun pk/1. fun enc/3.
reduc dec(x, enc(pk(x), y, z)) = z
```

produces the clauses

```
attacker:x -> attacker:pk(x);
attacker:x & attacker:y & attacker:z
    -> attacker:enc(x, y, z);
attacker:x & attacker:enc(pk(x), y, z))
    -> attacker:z;
```

Services offered by participants

```
let Server =
  in (ch, x);
  new n;
  out (ch, enc(k, (x,n) ));
```

produces the clause

```
attacker:x → attacker:enc(k[], (x,n[x = x]));
```

Services offered by participants

```
let Client =
    in (ch, m);
    let sig = dec(skC, m) in
    if checksign(pkS, sig) = ok then
    let k' = getmess(sig) in
    new s;
    out (ch, senc(k', s)).
```

produces the clause

```
attacker:enc(pk(skC[]), sign(skS[], y))
    -> attacker:senc(y,
        s[m = enc(pk(skC[]), sign(skS[], y))]);
```

The predicate *message*$(\cdot, \cdot)$

attacker:N means N can be made available on a public channel.
message:M, N means N can be made available on a channel called M.
Inputs and outputs on private channels are encoded using *message*.
These clauses link the two:

```
message:x,y & attacker:x -> attacker:y
attacker:x & attacker:y -> message:x,y
```

Equations vs. reducs (constructors/destructors)

When should we use equations and when should we use reducs?

AES cipher

```
equation dec(k,enc(k,m)) = m.
equation enc(k,dec(k,m)) = m.

att:x & att:y -> att:enc(x,y)
att:x & att:y -> att:dec(x,y)
att:x & att:enc(x,y) -> att:y
att:x & att:dec(x,y) -> att:y
```

AES scheme

```
reduc dec(k,enc(k,v,m)) = m.
reduc iv(enc(k,v,m)) = v.

att:x & att:y & att:z
                -> att:enc(x,y,z)
att:x & att:enc(x,y,z) -> att:z
att:enc(x,y,z) -> att:y
```

Equations vs. reducs (process)

reduc/equation `dec(x,enc(x,y)) = y.`

```
new k; new s;
in (c, x);
let y = dec(k, x) in
out (c, s)
```

reduc: `att:enc(k[],y) -> att:s[]`

equation:
```
att:enc(k[],y) -> att:s[]
att:x & att:y -> att:dec(x,y)
att:x -> att:s[]
```

Saturation

Definitions

- A *rule (or* Horn clause) is a set of hypothesis facts and a conclusion fact, written $F_1, \ldots, F_n \rightarrow F$.
- A fact is *unselectable if it is of the form attacker*(x)*, for some variable* x*.*
- *A selection function sel* : *rules* $\rightarrow$ *facts satisfies:*

$$sel(F_1, \ldots, F_n \rightarrow F) = \begin{cases} \emptyset & \text{if } F_1, \ldots, F_n \text{ are unselectable} \\ \{F_i\} & F_i \text{ is selectable,} \\ & \text{where } 1 \leq i \leq n \end{cases}$$

Algorithm

First phase: saturation

saturate$(\mathcal{R}_0) =$

- $\mathcal{R} \leftarrow \emptyset$.
  For each $R \in \mathcal{R}_0, \mathcal{R} \leftarrow \mathit{elim}(\mathit{simplify}(R) \cup \mathcal{R})$.
- Repeat until a fixpoint is reached
  for each $R \in \mathcal{R}$ such that $\mathrm{sel}(R) = \emptyset$,
  for each $R' \in \mathcal{R}$, for each $F_0 \in \mathrm{sel}(R')$ such that $R \circ_{F_0} R'$ is defined,
  $\mathcal{R} \leftarrow \mathit{elim}(\mathit{simplify}(R \circ_{F_0} R') \cup \mathcal{R})$.
- Return $\{R \in \mathcal{R} \mid \mathrm{sel}(R) = \emptyset\}$.

Second phase: backwards depth-first search

$$\mathrm{deriv}(R, \mathcal{R}, \mathcal{R}_1) = \begin{cases} \emptyset & \text{if } \exists R' \in \mathcal{R}, R' \sqsupseteq R \\ \{R\} & \text{otherwise, if } \mathrm{sel}(R) = \emptyset \\ \bigcup\{\mathrm{deriv}(\mathit{simplify}'(R' \circ_{F_0} R), & \\ \quad \{R\} \cup \mathcal{R}, \mathcal{R}_1) \mid R' \in \mathcal{R}_1, & \\ \quad F_0 \in \mathrm{sel}(R) \text{ such that} & \\ \quad R' \circ_{F_0} R \text{ is defined}\} & \text{otherwise} \end{cases}$$

$\mathrm{derivable}(F, \mathcal{R}_1) = \mathrm{deriv}(F \Rightarrow F, \emptyset, \mathcal{R}_1)$

Correctness

Correctness

If $\mathit{attacker}(M)$ cannot be derived from $\mathit{clauses}(P)$ then M is secret in P.

ProVerif abstractions

- Processes are treated as if they can be executed multiple times.

$$Clauses(new\ \tilde{n}; P) \approx Clauses(new\ \tilde{n}; !P)$$

- A value output on a channel is treated as if output multiple times.
- New values are not necessarily new.

$$new\ p; (!(new\ n; in(c, x); out(p, (x, n)) \mid in(p, (=a, y)); in(p, (=b, =y)); out(c, secret)))$$

These abstractions are safe (i.e. they increase the power of the attacker; they do not lose attacks), but they can produce false attacks.

ProVerif tricks (non-termination)

- Put *new* before $in(\cdot, \cdot)$.
- Replace *new n*
  with $in(pubch, n)$
- Replace $new\ n; \ldots\ if\ x_n <> T\ then \ldots$
  with $new\ n_0; let\ n = prop(n_0)\ in \ldots if\ deprop(x_n)\ then \ldots$
  where we have the reduc $deprop(prop(x)) = x$.
- Don't have queries with variables:
  Replace `query attacker:M.`
  with `attacker:M -> mybad:;`
  `query mybad:.`
- Look at infinite series of clauses being produced. Consider inserting a clause that subsumes them all.
- Consider fiddling with `nounif`.

ProVerif tricks (false attacks)

General

- Put new after $in(\cdot, \cdot)$.
- Use different private channels for different purposes. (Can't test occurrences on private channels, but can test presence.)

Equivalences

- Swapping for equivalences (see later).
  $out(c, choice[a, b]) \mid out(c, choice[b, a])$
- Skolem term depends on private data.
  $new\ p; (in(c, y); new\ n; out(p, (choice[const, y], n))) \mid (in(p, x); new\ m; out(c, m))$

Equivalence properties

```
free ch.
fun pk/1. fun enc/3. fun dec/2.
equation dec(x, enc(pk(x), y, z)) = z.

let Alice =
    new n;
    let msg = choice[enc(pk(skB), n, I_love_you), n] in
    out (ch, msg).

process
    new skB; out (ch, pk(skB)); !Alice
```

Predicate *attacker2*$(\cdot, \cdot)$

Predicate `attacker2:M,N`

Using the biprocess, the attacker can obtain M from the left process and N from the right process.

Simple equivalence example

Process:

```
param verboseClauses = short.

free ch, vote.

(* Public key cryptography *)
fun pk/1.
fun enc/3. fun dec/2.
equation dec(x, enc(pk(x), y, z) ) = z.

let Arole =
    new n;
    let msg = choice[ enc(pk(skB), n, vote), n ] in
    out (ch, msg).

process
    new skB; let pkB=pk(skB) in out (ch, pk(skB));
    ! Arole
```

Clauses:

```
Clause 0: att2:x,y -> att2:pk(x),pk(y)
Clause 1: att2:x,x' & att2:y,y' & att2:z,z' -> att2:enc(x,y,z),enc(x',y',z')
Clause 2: att2:x,x' & att2:y,y' -> att2:dec(x,y),dec(x',y')
Clause 3: att2:u,x & att2:v,enc(pk(x),y,z) -> att2:dec(u,v),z
Clause 4: att2:x,u & att2:enc(pk(x),y,z),v -> att2:z,dec(u,v)
Clause 5: att2:x,x' & att2:enc(pk(x),y,z),enc(pk(x'),y',z') -> att2:z,z'
Clause 13: att2:vote[],vote[]
Clause 14: att2:ch[],ch[]
Clause 15: att2:new_name[!att = v_73],new_name[!att = v_73]
Clause 16: x <> z & att2:x,y & att2:x,z -> bad:
Clause 17: x <> z & att2:y,x & att2:z,x -> bad:
Clause 18: att2:pk(skB[]),pk(skB[])
Clause 19: att2:enc(pk(skB_7[]),n_9[!1 = sid_101],vote[]),n_9[!1 = sid_101]
```

Simple non-equivalence example

Process:

```
param verboseClauses = short.

free ch, a,b.

let Arole = out (ch, choice[a,b]).
let Brole = out (ch, choice[b,a]).

process
    Arole | Brole
```

Clauses:

```
Clause 7: attacker2:b[],b[]
Clause 8: attacker2:a[],a[]
Clause 9: attacker2:ch[],ch[]
Clause 10: attacker2:new_name[!att = v_25],new_name[!att = v_25]
Clause 11: x <> y & attacker2:x,z & attacker2:y,z -> bad:
Clause 12: x <> y & attacker2:z,x & attacker2:z,y -> bad:
Clause 13: attacker2:a[],b[]
Clause 14: attacker2:b[],a[]
```

Outline

Applied pi calculus

Applied pi calculus: grammar

Terms

$$
\begin{array}{lll}
L, M, N, T, U, V ::= & & \\
\quad a, b, c, k, m, n, s, t, r, \ldots & & \text{name} \\
\quad x, y, z & & \text{variable} \\
\quad g(M_1, \ldots, M_l) & & \text{function}
\end{array}
$$

Equational theory

Suppose we have defined nullary function *ok*, unary function *pk*, binary functions *dec*, *senc*, *sdec*, *sign* and ternary functions *enc* and *checksign*.

$$
\begin{array}{ll}
\text{equation } \mathit{sdec}(x, \mathit{senc}(x, y)) & = y \\
\text{equation } \mathit{dec}(x, \mathit{enc}(\mathit{pk}(x), r, y)) & = y \\
\text{equation } \mathit{checksign}(\mathit{pk}(x), y, \mathit{sign}(x, y)) & = \mathit{ok}
\end{array}
$$

Equations to model the cryptography

1. *Encryption and signatures*

$$sdec(x, senc(x, y)) = y$$
$$dec(x, enc(pk(x), r, y)) = y$$
$$checksign(pk(x), sign(x, y)) = ok$$

2. *Blind signatures*

$$unblind(r, sign(x, blind(r, y))) = y$$

3. *Designated verifier proof of re-encryption*
The term $dvp(x, rencrypt(r, x), r, pkv)$ represents a proof designated for the owner of pkv that x and $rencrypt(x, r)$ have the same plaintext.

$$checkdvp(dvp(x, rencrypt(r, x), r, pkv), x, rencrypt(r, x), pkv) = ok$$
$$checkdvp(dvp(x, y, z, skv), x, y, pk(skv)) = ok$$

4. *Zero knowledge proofs of knowledge. . .*

Applied pi calculus: grammar

Processes

$P, Q, R, \ldots ::=$	processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction
$u(x)$	message input
$\overline{u}\langle M \rangle$	message output
if $M = N$ *then* P *else* Q	conditional

$A, B, C, \ldots ::=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{^{M}/_{x}\}$	active substitution

Example

$$\nu k.(\overline{c}\langle senc(k, a)\rangle.\overline{c}\langle senc(k, b)\rangle \mid \{^{h(k)}/_{x}\})$$

Machine-readable syntax

Math. syntax	Machine syntax
0	`0`
$P \mid Q$	`P	Q`
$!P$	`!P`
$\nu n.P$	`new n; P`
$u(x).P$	`in(u,x); P`
$\overline{u}\langle M \rangle.P$	`out(u,M); P`
$\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$	`if M = N then P else Q`
$\nu x.(\{^{M}/_{x}\} \mid P)$	`let x = M in P`

Applied pi calculus: Operational semantics I

Par-0	A	$\equiv$	$A \mid 0$
Par-A	$A \mid (B \mid C)$	$\equiv$	$(A \mid B) \mid C$
Par-C	$A \mid B$	$\equiv$	$B \mid A$
Repl	$!P$	$\equiv$	$P \mid !P$
New-0	$\nu n.0$	$\equiv$	0
New-C	$\nu u.\nu w.A$	$\equiv$	$\nu w.\nu u.A$
New-Par	$A \mid \nu u.B$	$\equiv$	$\nu u.(A \mid B)$
			where $u \notin fv(A) \cup fn(A)$
Alias	$\nu x.\{^M/_x\}$	$\equiv$	0
Subst	$\{^M/_x\} \mid A$	$\equiv$	$\{^M/_x\} \mid A\{^M/_x\}$
Rewrite	$\{^M/_x\}$	$\equiv$	$\{^N/_x\}$
			where $M =_E N$

Applied pi calculus: Operational semantics II

$$\textsc{Comm} \qquad \overline{c}\langle M\rangle.P \mid c(x).Q \;\rightarrow\; P \mid Q\{^{M}/_{x}\}$$

$$\textsc{Then} \qquad \mathit{if}\ N = N\ \mathit{then}\ P\ \mathit{else}\ Q \;\rightarrow\; P$$

$$\textsc{Then} \qquad \mathit{if}\ L = M\ \mathit{then}\ P\ \mathit{else}\ Q \;\rightarrow\; Q$$

for ground terms L, M where $L \neq_E M$

Syntactic secrecy

- Secrecy of M is preserved if an adversary cannot construct M from the outputs of the protocol.
- Formalise the adversary as a process I running in parallel. If I cannot output M, then secrecy is preserved.

Can-output

P *can output* the term M if there exists an evaluation context $C[\_]$ and channel $c \notin bn(C)$ and process R such that the reduction $P \rightarrow^* C[\overline{c}\langle M \rangle.R]$ with no alpha-renaming of the names in $fn(M)$.

Syntactic secrecy

A closed plain process P preserves the *syntactic secrecy* of M, if there is no plain processes I where $(fn(I) \cup bn(I)) \cap bn(P) = \emptyset$ such that $P \mid I$ can output M.

Security properties

The applied pi calculus can model the following:

- Reachability properties (e.g. secrecy)
- Correspondence assertions (e.g. authentication)
- Observational equivalence (e.g. strong secrecy, for instance, ballot secrecy)

Examples:

- *Certified email* [AbadiBlanchet05]
- *Privacy* properties [DelauneKremerRyan09], and *election verifiability* properties [SmythRyanKremer10] in e-voting
- *Trusted computing protocols* [ChenRyan09,MukhamedovGordonRyan09], and *attestation* protocols [SmythRyanChen07,Backes08]
- *Web services interoperability* [BhargavanFournetGordonTse]
- *Integrity of file systems* on untrusted storage [ChaudhuriBlanchet08]

Applied pi calculus: operational semantics III

Labelled semantics: $A \xrightarrow{\alpha} B$

- $A \xrightarrow{c(M)} B$ means that the process A performs an input of the term M from the environment on the channel c, and the resulting process is B.
- $A \xrightarrow{\overline{c}\langle u \rangle} B$ means that the process A outputs the free u (which may be a variable, or a channel name).
- $A \xrightarrow{\nu u.\overline{c}\langle u \rangle} B$ means A outputs u that is restricted in A, and becomes free in B. Again, u is a channel name or a variable representing a term.

Applied pi calculus: operational semantics IV

IN
$$c(x).P \xrightarrow{c(M)} P\{^M/_x\}$$

OUT-ATOM
$$\overline{c}\langle u\rangle.P \xrightarrow{\overline{c}\langle u\rangle} P$$

OPEN-ATOM
$$\frac{A \xrightarrow{\overline{c}\langle u\rangle} A' \qquad u \neq c}{\nu u.A \xrightarrow{\nu u.\overline{c}\langle u\rangle} A'}$$

SCOPE
$$\frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

PAR
$$\frac{A \xrightarrow{\alpha} A' \qquad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

STRUCT
$$\frac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}$$

Operational semantics: example

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\overline{c}\langle x\rangle.P \xrightarrow{\overline{c}\langle x\rangle} P}\ \textsc{Out-Atom}}{\overline{c}\langle x\rangle.P \mid \{M/x\} \xrightarrow{\overline{c}\langle x\rangle} P \mid \{M/x\}}\ \textsc{Par}}{\nu x.(\overline{c}\langle x\rangle.P \mid \{M/x\}) \xrightarrow{\nu x.\overline{c}\langle x\rangle} P \mid \{M/x\}}\ \textsc{Open-Atom}}{\overline{c}\langle M\rangle.P \equiv \nu x.(\overline{c}\langle x\rangle.P \mid \{M/x\}) \xrightarrow{\nu x.\overline{c}\langle x\rangle} P \mid \{M/x\} \equiv P \mid \{M/x\}}\ \textsc{Struct}$$

Correspondence properties III

Correspondence property

A *correspondence property* is a formula of the form:

$$\overline{f}\langle M\rangle \rightsquigarrow \overline{g}\langle N\rangle.$$

A correspondence property asserts if event f has been executed then the event g must have been previously executed and any relationship between the event parameters must be satisfied.

Validity of correspondence property

Let E be an equational theory, and A_0 an extended process. We say that A_0 *satisfies the correspondence property* $\overline{f}\langle M\rangle \rightsquigarrow \overline{g}\langle N\rangle$ if for all execution paths

$$A_0 \rightarrow^* \xrightarrow{\alpha_1} \rightarrow^* A_1 \rightarrow^* \xrightarrow{\alpha_2} \rightarrow^* \cdots \rightarrow^* \xrightarrow{\alpha_n} \rightarrow^* A_n,$$

and all index $i \in \mathbb{N}$, substitution σ and variable e such that $\alpha_i = \nu\, e.\overline{f}\langle e\rangle$ and $e\varphi(A_i) =_E M\sigma$, there exists $j \in \mathbb{N}$ and e' such that $\alpha_j = \nu\, e'.\overline{g}\langle e'\rangle$, $e'\varphi(A_j) =_E N\sigma$ and $j < i$.

Equivalence properties

Equivalence defines indistinguishability between two processes and allows us to consider properties that cannot be expressed as secrecy or correspondence properties.

Example: electronic voting

We cannot model vote privacy as syntactic secrecy, because

- the identies of the voters are not secret;
- the votes (names of candidates) are not secret.

What is secret is the link between the voter and the vote.

Privacy in electronic voting

A voting protocol respects privacy if

$$S[V_A\{^a/_v\} \mid V_B\{^b/_v\}] \approx S[V_A\{^b/_v\} \mid V_B\{^a/_v\}].$$

Observational equivalence

We write $A \Downarrow c$ when A can evolve to a process that can send a message on c, that is, when $A \to^* C[\overline{c}\langle M \rangle.P]$ for some term M and some evaluation context $C[\_]$ that does not bind c.

Observational equivalence

Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A \mathcal{R} B$ implies:

1. if $A \Downarrow c$, then $B \Downarrow c$.
2. if $A \to^* A'$ then, for some B', we have $B \to^* B'$ and $A' \mathcal{R} B'$;
3. $C[A] \mathcal{R} C[B]$ for all closing evaluation contexts $C[\_]$.

The definition universally quantifies over evaluation contexts to capture all possible adversary behaviour. This makes the definition of observational equivalence hard to use in practice.

Labelled bisimilarity I

Labelled bisimilarity is more suitable for reasoning. It relies on an equivalence relation between frames; intuitively, two frames are statically equivalent if no 'test' $M = N$ can tell them apart

Static equivalence

Two closed frames $\varphi \equiv \nu\, \tilde{m}.\sigma$ and $\psi \equiv \nu\, \tilde{n}.\tau$ are statically equivalent, denoted $\varphi \approx_s \psi$, if $\mathrm{dom}(\varphi) = \mathrm{dom}(\psi)$ and for all terms M, N such that $(\tilde{m} \cup \tilde{n}) \cap (\mathrm{fn}(M) \cup \mathrm{fn}(N)) = \emptyset$, we have $M\sigma =_E N\sigma$ holds if and only if $M\tau =_E N\tau$ holds.

Examples

- $\nu\, m.\{^m/_x\} \approx_s \nu\, n.\{^n/_x\}$; they are structurally equivalent.
- $\nu\, m.\{^m/_x\} \approx_s \nu\, n.\{^{hash(n)}/_x\}$.
- $\{^m/_x\} \not\approx_s \{^{hash(m)}/_x\}$. LHS satisfies $x = m$.
- $\nu\, s.\{^{pair(s,s)}/_x\} \not\approx_s \nu\, s.\{^s/_x\}$.
  LHS satisfies $pair(fst(x), snd(x)) = x$.

Labelled bisimilarity II

Static equivalence examines the current state of the processes (as represented by their frames), and not the processes' dynamic behaviour (that is, the ways in which they may execute in the future). The dynamic part is captured as follows.

Labelled bisimilarity

Labelled bisimilarity ($\approx_l$) is the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $A \mathcal{R} B$ implies:

1. $A \approx_s B$;
2. if $A \to A'$ then $B \to^* B'$ and $A' \mathcal{R} B'$ for some B';
3. if $A \xrightarrow{\alpha} A'$ and $\mathrm{fv}(\alpha) \subseteq \mathrm{dom}(A)$ and $\mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset$; then $B \to^* \xrightarrow{\alpha} \to^* B'$ and $A' \mathcal{R} B'$ for some B'.

Abadi & Fournet state that observational equivalence and labelled bisimilarity coincide.