Verifying Cryptographic Protocols in Applied Pi Calculus

Mark Ryan   Ben Smyth

M.D.Ryan@cs.bham.ac.uk
research@bensmyth.com

Cryptoforma

7th April 2010

UNIVERSITY OF BIRMINGHAM

Cryptographic protocols

A cryptographic protocol is a distributed procedure that employs cryptography to achieve a security goal.

Examples of participating agents:

- Client *and* Server
- Application *and* TPM
- VM_1 *and* VMM
- VM_1 *and* VM_2
- VoterAgent *and* Collector
- Alice *and* Bob
- *n* parties agreeing a contract signature

Cryptographic protocols

A cryptographic protocol is a distributed procedure that employs cryptography to achieve a security goal.

Examples of "security goal":

- Authentication
- Key agreement
- Secure communication
- Privacy
- Confidentiality management
- Attestation
- Non-repudiation
- Fair exchange
- Contract signing
- Secure storage
- Access control
- Voting

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

Example: handshake protocol

Handshake protocol

S		C
new k		
	$\xrightarrow{\mathrm{enc}_{pkC}(\mathrm{sign}_{skS}(k))}$	
	$\xleftarrow{\mathrm{senc}_k(s)}$	

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.

Handshake protocol attack

S $\qquad$ M $\qquad$ C

new k

$S \xrightarrow{\mathsf{enc}_{pkM}(\mathsf{sign}_{skS}(k))} M$

$M \xrightarrow{\mathsf{enc}_{pkC}(\mathsf{sign}_{skS}(k))} C$

$M \xleftarrow{\mathsf{senc}_{k}(s)} C$

$S \xleftarrow{\mathsf{senc}_{k}(s)} M$

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

S $\qquad\qquad\qquad\qquad\qquad$ C

new k

$$\xrightarrow{\mathsf{enc}_{pkC}(\mathsf{sign}_{skS}(k, pkC))}$$

$$\xleftarrow{\mathsf{senc}_{k}(s)}$$

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

Example: Needham-Schroeder public key protocol

NSPK protocol

A		B
new N_A		new N_B
	$\xrightarrow{\mathrm{enc}_{pkB}(N_A, pkA)}$	
	$\xleftarrow{\mathrm{enc}_{pkA}(N_A, N_B)}$	
	$\xrightarrow{\mathrm{enc}_{pkB}(N_B)}$	

As before, A and B know each other's public keys, and want to agree a session key for private communication. They communicate on a channel which is controlled by the attacker.

- If Alice has completed the protocol, apparently with Bob, then Bob has completed the protocol with her.
- If Bob has completed the protocol, apparently with Alice, then Alice has completed the protocol with him.
- Messages sent encrypted with the agreed key (based on N_A, N_B) remain secret.

NSPK protocol fixed

The protocol (invented in 1978) was found to be flawed in 1995. The attack is avoided similarly as before, by including identity information in an encrypted package.

Revised NSPK

A		B
new N_A		new N_B
	$\xrightarrow{\mathsf{enc}_{pkB}(N_A, pkA)}$	
	$\xleftarrow{\mathsf{enc}_{pkA}(N_A, N_B, pkB)}$	
	$\xrightarrow{\mathsf{enc}_{pkB}(N_B)}$	

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

Verifying cryptographic protocols

“Provable/computational security”

1. Computationally bounded (polynomial) attacker
2. Exact cryptographic operations on bitstrings
3. Bitstring (more concrete) model
4. Prove difficulty of violating security property is equivalent to solving a *hard* problem

“Formal/symbolic methods”

1. Idealised (worst case) attacker
2. Idealised (best case) perfect cryptography
3. Symbolic (more abstract) model of protocol
4. Prove impossibility of violating security property within the model

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]
- Furthermore, they fail to detect certain kinds of attack [Meadows'03, KoblitzMenezes'04, SmythRyanChen'07]
- Formal methods are simpler, specifications are nicer and automated support is available
- Caveat: gulf between abstract formal model and real world specification (and the actual implementation)

Reconciling two views of cryptography

- [AbadiRogaway'00], [PfitzmannSchunterWaidner'00], [Warinschi'05], [Blanchet'07]
- EPSRC (UK) funded CryptoForma network (EP/G069875/1)

Applied pi calculus and ProVerif

- The applied pi calculus is a language for describing concurrent processes and their interactions
  - Developed explicitly for modelling security protocols
  - Similar to spi calculus; with more general cryptography
- ProVerif is a leading software tool for automated reasoning
  - Takes applied pi processes and reasons about observational equivalence, correspondence assertions and secrecy

History of applied pi calculus and ProVerif

1970s: Milner's *Calculus of Communicating Systems* (CCS)

1989: Milner *et al.* extend CCS to *pi calculus*

1999: Abadi & Gordon introduce *spi calculus*, variant of pi

2001: Abadi & Fournet generalise spi to *applied pi calculus*

2000s: Blanchet develops *ProVerif* to enable automated reasoning for applied pi calculus processes

Applied pi calculus: Grammar

Terms

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

Equational theory

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

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

Applied pi calculus: Grammar

Processes

$P, Q, R ::=$	processes	$A, B, C ::=$	extended processes
0	null process	P	plain process
$P \mid Q$	parallel comp.	$A \mid B$	parallel comp.
$!P$	replication	$\nu\, n.A$	name restriction
$\nu\, n.P$	name restriction	$\nu\, x.A$	variable restriction
$u(x).P$	message input	$\{^{M}/_{x}\}$	active substitution
$\overline{u}\langle M\rangle.P$	message output		
$\text{if } M = N \text{ then } P \text{ else } Q$	cond'nl		

Example

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

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`
$\text{if } M = N \text{ then } P \text{ else } Q$	`if M=N then P else Q`
$\nu x.(\{^{M}/_{x}\} \mid P)$	`let x=M in P`

Applied pi calculus: Operational semantics I

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

Applied pi calculus: Operational semantics II

COMM	$\overline{c}\langle x\rangle.P \mid c(x).Q \rightarrow P \mid Q$
THEN	$\text{if } N = N \text{ then } P \text{ else } Q \rightarrow P$
ELSE	$\text{if } L = M \text{ then } P \text{ else } Q \rightarrow Q$
	for ground terms L, M where $L \neq_E M$

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

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

Operational semantics: example

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

Operational semantics: example

The process

$$A \hat{=} \nu\, s.(c(x).\text{if } x = s \text{ then } \overline{c}\langle i\_got\_s \rangle)$$

can never output i_got_s, because no term input as x can be equal to the 'new' s created by the process.

More precisely, there is no sequence of reductions

$$A \rightarrow^{*} \xrightarrow{\alpha} \rightarrow^{*} \cdots \rightarrow^{*} \xrightarrow{\alpha} \rightarrow^{*} B \mid \{{}^{i\_got\_s}/_{y}\}$$

for some process B and variable y.

Operational semantics: example

Consider A'':

$$A'' \hat{=} \nu\, s.(\overline{c}\langle \mathsf{senc}(k,s)\rangle.c(x).\text{if } x = s \text{ then } \overline{c}\langle i\_got\_s\rangle)$$

This test can succeed; the process can output i_got_s, as shown by the following execution:

$$\begin{array}{rcl}
A'' & \xrightarrow{\nu\, y.\overline{c}\langle y\rangle} & \nu\, s.(c(x).\text{if } x = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,s)}/_{y}\}) \\
 & \xrightarrow{c(\mathsf{sdec}(k,y))} & \nu\, s.(\text{if } \mathsf{sdec}(k,y) = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,s)}/_{y}\}) \\
 & \equiv & \nu\, s.(\text{if } \mathsf{sdec}(k,\mathsf{senc}(k,s)) = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,s)}/_{y}\}) \\
 & \equiv & \nu\, s.(\text{if } s = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,s)}/_{y}\}) \\
 & \rightarrow & \nu\, s.(\overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,s)}/_{y}\}) \\
 & \xrightarrow{\nu\, z.\overline{c}\langle z\rangle} & \nu\, s.(\{^{\mathsf{senc}(k,s)}/_{y}\} \mid \{^{i_got_s}/_{z}\}) \\
 & \equiv & \nu\, s.(\{^{\mathsf{senc}(k,s)}/_{y}\}) \mid \{^{i_got_s}/_{z}\}
\end{array}$$

Five steps to verification

1. Write equations to capture cryptographic primitives
   - For example: $\mathtt{dec}(x, \mathtt{enc}(x, y)) = y$
2. Decide which partipants are honest/dishonest
3. Model the honest parties as processes. Example:

```
processA =
   new k;
   in(c, m);
   out(c, sign(k, m));
```

4. Model the intended security property
   - as a reachability property
   - as a correspondence property
   - as an observational equivalence property
5. Evaluate the complete model using ProVerif and/or hand reasoning

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 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

Equations to model the cryptography

1. *Encryption and signatures*

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

2. *Blind signatures*

```
unblind(r, sign( x, blind(r,y) ) )  =  sign(x,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...*

Coding protocols as processes

Original handshake protocol:

```
let Server =
  in (ch, pkC');

  new k;
  out (ch, enc(pkC', sign(skS, k ) ));

  in (ch, m);
  0.
```

Handshake protocol

S C

new k

$\xrightarrow{\mathsf{enc}_{pkC}(\mathsf{sign}_{skS}(k))}$

$\xleftarrow{\mathsf{senc}_k(s)}$

The handshake protocol in full

```
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.
```

The handshake protocol in full 2

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

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

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];

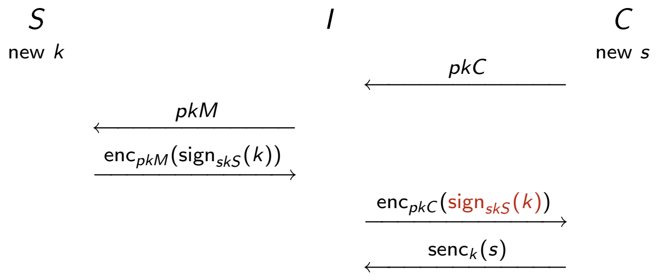
Syntactic secrecy

- 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.

Syntactic secrecy

A closed plain process P preserves the *syntactic secrecy* of M, if for all plain processes I where $\mathrm{fn}(I) \cap \mathrm{bn}(P) = \emptyset$, there is no evaluation context $C[\_]$ with channel $c \notin \mathrm{bn}(C)$ and process R such that $P \mid I \rightarrow^* C[\overline{c}\langle M \rangle.R]$.

Syntactic secrecy (Handshake protocol example)



- C publishes her public key
- I starts a session with S
- I learns $\mathrm{sign}_{skS}(k)$ and k
- I replays $\mathrm{sign}_{skS}(k)$ in a session with S
- I is able to output secrect s

Adversary process I

in (c, xPK);
out (c, pkM);
in (c, y);
let sig = $dec_{skM}(y)$ in
out (c, $enc_{xPK}(sig)$);
in (c, z);
out (c, $sdec_{getmsg(sig)}(z)$)

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

- event startedS(pair(pkB',k)) means A started the protocol with interlocutor having pub key pkB', and k is the session key.
- event compS(k) means A completed the protocol with session key k.

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

Correspondence properties II

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 $j < i$.

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

Authentication properties

Client wants authentication of server:

$$\overline{compC}\langle pair(x,y)\rangle \rightsquigarrow \overline{startedS}\langle pair(x,y)\rangle.$$

Server wants authentication of client that started session:

$$\overline{compS}\langle y\rangle \rightsquigarrow \overline{startedC}\langle y\rangle$$

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

Classically modelled as observational equivalences between two slightly different processes P_1 and P_2, but

- changing the identity does not work, as identities are revealed
- changing the vote does not work, as the votes are revealed at the end

$\hookrightarrow$ consider two honest voters and swap their votes

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 $A \rightarrow^* C[\overline{c}\langle M \rangle.P]$ 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 $A \mathrel{\mathcal{R}} B$ implies:

1. if $A \Downarrow c$, then $B \Downarrow c$.
2. if $A \rightarrow^* A'$ then, for some B', we have $B \rightarrow^* B'$ and $A' \mathrel{\mathcal{R}} B'$;
3. $C[A] \mathrel{\mathcal{R}} C[B]$ 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 $A \mathrel{\mathcal{R}} B$ implies:

1. $A \approx_s B$;
2. if $A \rightarrow A'$ then $B \rightarrow^* B'$ and $A' \mathrel{\mathcal{R}} B'$ 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 $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A' \mathrel{\mathcal{R}} B'$ for some B'.

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

Weak secrets I

Weak secret

A secret is *weak* if it is low entropy, and therefore potentially easily guessable by an attacker. Typically, human-memorable secrets are weak.

Offline dictionary attack

A secret value in a protocol is vulnerable to offline *dictionary attack* (also called *guessing attack*) if an attacker could confirm the correctness of a large number of guesses of the secret on the basis of data he receives in a single session.

Example: A webmail login program should be such that the password is not vulnerable to offline dictionary attack!

Weak secrets II

The correct value of the secret "looks the same" as the incorrect value.

Guessing attacks on frames

Let $\varphi \equiv \nu n.\varphi'$ be a frame. We say that φ is *resistant to guessing attacks* against n if, and only if, $\nu n.(\varphi' \mid \{^n/_x\}) \approx_s \nu n'.\nu n.(\varphi' \mid \{^{n'}/_x\})$ where n' is a fresh name and x is a variable such that $x \notin \mathrm{dom}(\varphi)$.

Guessing attacks on processes

Let A be a process and $n \in \mathrm{bn}(A)$. We say that A is *resistant to guessing attacks* against n if, for every process B such that $A(\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^* B$, then we have that $\varphi(B)$ is resistant to guessing attacks against n.

Weak secrets III

TPM authentication:

$$
\begin{array}{rcl}
P & \triangleq & \nu\, s.(!P_A \mid !P_B) \\
P_A & \triangleq & \nu\, n.\overline{c}\langle(\!|\mathsf{comm},\ n,\ \mathsf{mac}(s, (\!|\mathsf{comm}, n|\!))|\!)\rangle \\
P_B & \triangleq & c(x).\mathrm{if}\ \mathsf{3rd}(x) = \mathsf{mac}(s, (\!|\mathsf{1st}(x), \mathsf{2nd}(x)|\!))\ \mathrm{then}\ \overline{c}\langle\mathsf{resp}\rangle
\end{array}
$$

where

$$
\begin{array}{rcl}
(\!|M_1, \ldots, M_n|\!) & = & \mathsf{pair}(M_1, \mathsf{pair}(M_2, \mathsf{pair}(\ldots, \mathsf{pair}(M_n, *)\ldots))) \\
\mathsf{1st}(M) & = & \mathrm{fst}(M) \\
\mathsf{2nd}(M) & = & \mathrm{fst}(\mathrm{snd}(M)) \\
\mathsf{3rd}(M) & = & \mathrm{fst}(\mathrm{snd}(\mathrm{snd}(M)))
\end{array}
$$

Weak secrets IV

$$P \triangleq \nu s.(!P_A \mid !P_B)$$

$$P_A \triangleq \nu n.\overline{c}\langle(\!|\mathsf{comm},\ n,\ \mathsf{mac}(s, (\!|\mathsf{comm}, n|\!))|\!)\rangle$$

$$P_B \triangleq c(x).\text{if } \mathsf{3rd}(x) = \mathsf{mac}(s, (\!|\mathsf{1st}(x), \mathsf{2nd}(x)|\!)) \text{ then } \overline{c}\langle\mathsf{resp}\rangle$$

P is vulnerable to guessing attacks on s. To see this, we consider the transition

$$P \xrightarrow{\nu x.\overline{c}\langle x\rangle} \nu s.(\nu n.\{(\!|\text{comm},\ n,\ \mathsf{mac}(s, (\!|\text{comm}, n|\!))|\!)/_x\} \mid !P_A \mid !P_B)$$

The frame of this latter process is vulnerable to guessing attacks on s, since we have

$$\begin{aligned}&\nu s.\nu n.(\{(\!|\text{comm},\ n,\ \mathsf{mac}(s, (\!|\text{comm}, n|\!))|\!)/_x\} \mid \{^s/_z\})\\ &\qquad \not\approx_s \nu s'.\nu s.\nu n.(\{(\!|\text{comm},\ n,\ \mathsf{mac}(s, (\!|\text{comm}, n|\!))|\!)/_x\} \mid \{^{s'}/_z\})\end{aligned}$$

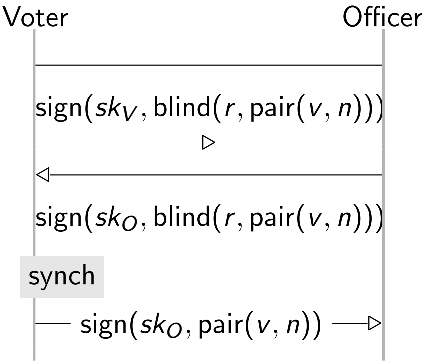
as witnessed by the test

$$\mathsf{3rd}(x) = \mathsf{mac}(z, (\!|\mathsf{1st}(x), \mathsf{2nd}(x)|\!)).$$

Ballot secrecy in electronic voting I

The protocol relies on *blind signatures*; with this cryptographic primitive, an agent can sign a text without having seen it. Another agent first blinds the text, then the signing agent signs it, and then the other agent unblinds it again. We do not need to consider how this cryptography actually works; we can encode the effect using the equation

$$\mathrm{unblind}(x, \mathrm{sign}(y, \mathrm{blind}(x, z))) = \mathrm{sign}(y, z)$$



Ballot secrecy in electronic voting II

$$
\begin{aligned}
P_V \quad \triangleq \quad & \nu n.\nu r.\mathrm{let}\ bvn = \mathsf{blind}(r, \mathsf{pair}(v, n))\ \mathrm{in} \\
& \overline{c}\langle \mathsf{pair}(\mathsf{pk}sk_V, \mathsf{sign}(sk_V, bvn))\rangle. \\
& c(x).\mathrm{if}\ \mathsf{checksign}(pk_O, x) = \mathsf{true}\ \mathrm{then} \\
& \mathrm{if}\ \mathsf{getmsg}(x) = bvn\ \mathrm{then} \\
& synch. \\
& \overline{c}\langle \mathsf{unblind}(r, x)\rangle
\end{aligned}
$$

$$
\begin{aligned}
P_O \quad \triangleq \quad & c(y).\mathrm{if}\ \mathsf{checksign}(\mathsf{fst}(y), \mathsf{snd}(y)) = \mathsf{true}\ \mathrm{then} \\
& \mathrm{if}\ \mathsf{Eligible}(\mathsf{fst}(y)) = \mathsf{true}\ \mathrm{then} \\
& \overline{c}\langle \mathsf{sign}(sk_O, \mathsf{getmsg}(\mathsf{snd}(y)))\rangle. \\
& c(w). \\
& \mathrm{if}\ \mathsf{checksign}(sk_O, w) = \mathsf{true}\ \mathrm{then} \\
& \mathrm{if}\ \mathsf{NotSeen}(w) = \mathsf{true}\ \mathrm{then} \\
& \overline{vote}\langle \mathsf{fst}(\mathsf{getmsg}(w))\rangle
\end{aligned}
$$

Ballot secrecy in electronic voting III

$$
\begin{array}{rcl}
P & \triangleq & \nu\, sk_1 \,\ldots\, \nu\, sk_n .\, \nu\, sk_O . \\
 & & \mathrm{let}\ pk_1 = \mathsf{pk} sk_1\ \mathrm{in}\ \ldots\ \mathrm{let}\ pk_n = \mathsf{pk} sk_V\ \mathrm{in} \\
 & & \mathrm{let}\ pk_O = \mathsf{pk} sk_O\ \mathrm{in} \\
 & & (\overline{c}\langle pk_1\rangle \mid \cdots \mid \overline{c}\langle pk_n\rangle \mid \overline{c}\langle pk_O\rangle \mid P_V\{^{sk_1}/_{sk_V}, {}^{v_1}/_{v}\} \\
 & & \qquad \mid \cdots \mid P_V\{^{sk_n}/_{sk_V}, {}^{v_n}/_{v}\} \mid !P_O \mid S)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{synch} & \triangleq & \overline{syn}\langle * \rangle . syn'(o) \\
S & \triangleq & syn(x_1) \ldots syn(x_n) . \overline{syn'}\langle * \rangle \ldots \overline{syn'}\langle * \rangle
\end{array}
$$

Ballot secrecy in electronic voting IV

The ballot secrecy property is written as the equivalence

$$\nu\, syn.\nu\, syn'.(P_V\{^{sk_A}/_{sk_V}, {}^{v_a}/_v\} \mid P_V\{^{sk_A}/_{sk_V}, {}^{v_b}/_v\} \mid S)$$
$$\approx_l \nu\, syn.\nu\, syn'.(P_V\{^{sk_A}/_{sk_V}, {}^{v_b}/_v\} \mid P_V\{^{sk_B}/_{sk_V}, {}^{v_a}/_v\} \mid S)$$

Proof: We define the relation $\mathcal{R}$ as follows. Given closed extended processes X and Y, $X \mathcal{R} Y$ and $Y \mathcal{R} X$ both hold if

- there exist integers i, j, variables w, z and terms M, N with $1 \leq i, j \leq 6$ and

  $$X \equiv P_i\{^{sk_A}/_{sk_V}, {}^{v_a}/_v, {}^w/_y, {}^M/_m\} \mid P_j\{^{sk_B}/_{sk_V}, {}^{v_b}/_v, {}^z/_y, {}^N/_m\} \mid S_{i,j},$$
  $$Y \equiv P_i\{^{sk_A}/_{sk_V}, {}^{v_b}/_v, {}^w/_y, {}^M/_m\} \mid P_j\{^{sk_B}/_{sk_V}, {}^{v_a}/_v, {}^z/_y, {}^N/_m\} \mid S_{i,j};$$

  and if $i = 4$ then $\mathrm{checksign}(pk_O, M) = \mathrm{true}$, and if $j = 4$ then $\mathrm{checksign}(pk_O, N) = \mathrm{true}$, or
- there exist integers i, j, and variables s, t, w, z with $6 \leq i, j \leq 8$ and

  $$X \equiv P_i\{^{sk_A}/_{sk_V}, {}^{v_a}/_v, {}^w/_y, {}^s/_u\} \mid P_j\{^{sk_B}/_{sk_V}, {}^{v_b}/_v, {}^z/_y, {}^t/_u\} \mid S_{i,j},$$
  $$Y \equiv P_j\{^{sk_A}/_{sk_V}, {}^{v_b}/_v, {}^w/_y, {}^t/_u\} \mid P_i\{^{sk_B}/_{sk_V}, {}^{v_a}/_v, {}^z/_y, {}^s/_u\} \mid S_{i,j}.$$

Summary

- Applied pi calculus provides a practical approach to cryptographic protocol verification
- Reasoning by hand, or in many cases by ProVerif, permitting analysis of:
  1. Reachability
  2. Correspondence assertions
  3. Observational equivalence
- Practical real world applications:

  TPM authorisation [ChenRyan'08], Direct Anonymous Attestation [SmythRyanChen'07], electronic voting [DelauneKremerRyan'08, SmythRyanKremerKourjieh'09], zero knowledge protocols [BackesMaffeiUnruh'07], certified email [AbadiBlanchet'05], JFK [AbadiBlanchetFournet'04], web services [CorinFournetGordon'04], untrusted sortage [BlanchetChaudhuri'08], ...