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
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**
- **VM₁ and VMM**
- **VM₁ and VM₂**
- **VoterAgent and Collector**
- **Alice and Bob**
- **n parties agreeing a contract signature**
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**

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 $C$ and $S$.
2. **Authentication of S**: if $C$ reaches the end of the protocol with session key $k$, then $S$ proposed $k$ for use by $C$.
3. **Authentication of C**: if $S$ reaches the end of the protocol and she believes she has session key $k$ with $C$, then $C$ was indeed her interlocutor and she has session $k$. 

$\textbf{Handshake protocol}$

\[
\begin{align*}
S & \quad \text{new } k \quad C \\
& \quad \text{enc}_{pkC}(\text{sign}_{skS}(k)) \\
& \quad \text{senc}_k(s)
\end{align*}
\]
Handshake protocol attack

\[ S \xrightarrow{\text{new } k} M \xrightarrow{\text{encrypt with } \text{pk}_M \big( \text{sign with } \text{sk}_S(k) \big)} \xrightarrow{\text{decrypt with } \text{pk}_C \big( \text{sign with } \text{sk}_S(k) \big)} C \xleftarrow{\text{encrypt with } \text{senc}_k(s)} \xleftarrow{\text{decrypt with } \text{senc}_k(s)} \]

Intended properties:

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

Our aim is to be able to automatically establish these facts.
Example: Needham-Schroeder public key protocol

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.
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.

The three properties hold of the revised protocol, but not for the original one.
### “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
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)
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
Terms

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

Equational theory

Suppose we have defined nullary function $\text{ok}$, unary function $\text{pk}$, binary functions $\text{enc}$, $\text{dec}$, $\text{senc}$, $\text{sdec}$, $\text{sign}$, and ternary function $\text{checksign}$.

\[
\text{sdec}(x, \text{senc}(x, y)) = y \\
\text{dec}(x, \text{enc}(\text{pk}(x), y)) = y \\
\text{checksign}(\text{pk}(x), y, \text{sign}(x, y)) = \text{ok}
\]
Applied pi calculus: Grammar

Processes

\[
P, Q, R ::= \text{processes} \quad \quad A, B, C ::= \text{extended processes}
\]

\[
\begin{align*}
0 & \quad \text{null process} \\
P \mid Q & \quad \text{parallel comp.} \\
!P & \quad \text{replication} \\
\nu n.P & \quad \text{name restriction} \\
u(x).P & \quad \text{message input} \\
\overline{u}(M).P & \quad \text{message output} \\
\text{if } M = N \text{ then } P \text{ else } Q & \quad \text{cond’nl}
\end{align*}
\]

Example

\[
\nu k. (\overline{c}(\text{senc}(k, a)). \overline{c}(\text{senc}(k, b)) \mid \{h(k)/x\})
\]
<table>
<thead>
<tr>
<th>Math. syntax</th>
<th>Machine syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P \mid Q$</td>
<td>$P \mid Q$</td>
</tr>
<tr>
<td>$!P$</td>
<td>$!P$</td>
</tr>
<tr>
<td>$\nu, n\cdot P$</td>
<td>new n ; P</td>
</tr>
<tr>
<td>$u(x)\cdot P$</td>
<td>in(u,x) ; P</td>
</tr>
<tr>
<td>$\overline{u}\langle M\rangle \cdot P$</td>
<td>out(u,M) ; P</td>
</tr>
<tr>
<td>if $M = N$ then $P$ else $Q$</td>
<td>if M=N then P else Q</td>
</tr>
<tr>
<td>$\nu, x.({M/x} \mid P)$</td>
<td>let x=M in P</td>
</tr>
</tbody>
</table>
**Applied pi calculus: Operational semantics I**

\[
\begin{align*}
\text{Par-0} & \quad A & \equiv & & A \mid 0 \\
\text{Par-A} & \quad A \mid (B \mid C) & \equiv & & (A \mid B) \mid C \\
\text{Par-C} & \quad A \mid B & \equiv & & B \mid A \\
\text{Repl} & \quad !P & \equiv & & P \mid !P \\
\text{New-0} & \quad \nu n.0 & \equiv & & 0 \\
\text{New-C} & \quad \nu u.\nu w. A & \equiv & & \nu w.\nu u. A \\
\text{New-Par} & \quad A \mid \nu u. B & \equiv & & \nu u.(A \mid B) \\
& & & & \text{where } u \notin \text{fv}(A) \cup \text{fn}(A) \\
\text{Alias} & \quad \nu x.\{M/x\} & \equiv & & 0 \\
\text{Subst} & \quad \{M/x\} \mid A & \equiv & & \{M/x\} \mid A\{M/x\} \\
\text{Rewrite} & \quad \{M/x\} & \equiv & & \{N/x\} \\
& & & & \text{where } M =_E N
\end{align*}
\]
\text{COMM} \quad \overline{c}(x).P \mid c(x).Q \rightarrow P \mid Q

\text{THEN} \quad \text{if } N = N \text{ then } P \text{ else } Q \rightarrow P

\text{ELSE} \quad \text{if } L = M \text{ then } P \text{ else } Q \rightarrow Q

\text{for ground terms } L, M \text{ where } L \not\equiv E M
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 $B$.

- $A \xrightarrow{\nu u.\overline{c}(u)} B$ means that the process $A$ outputs $u$ that is restricted in $A$, and becomes free in $B$. Again, $u$ is a channel name or a variable representing a term.
\[
\text{IN} \quad c(x).P \xrightarrow{c(M)} P\{M/x\}
\]

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

\text{OPEN-ATOM} \quad \begin{align*}
A & \xrightarrow{\overline{c}\langle u \rangle} A' \quad u \neq c \\
\nu u.A & \xrightarrow{\nu u.\overline{c}\langle u \rangle} A'
\end{align*}

\text{SCOPE} \quad \begin{align*}
A & \xrightarrow{\alpha} A' \quad u \text{ does not occur in } \alpha \\
\nu u.A & \xrightarrow{\alpha} \nu u.A'
\end{align*}

\text{PAR} \quad \begin{align*}
A & \xrightarrow{\alpha} A' \quad \text{bv}(\alpha) \cap \text{fv}(B) = \text{bn}(\alpha) \cap \text{fn}(B) = \emptyset \\
A \mid B & \xrightarrow{\alpha} A' \mid B
\end{align*}

\text{STRUCT} \quad \begin{align*}
A & \equiv B \\
B & \xrightarrow{\alpha} B' \\
B' & \equiv A' \\
A & \xrightarrow{\alpha} A'
\end{align*}
Operational semantics: example

\[
\begin{align*}
\text{OUT-ATOM} & : \quad \text{c} \langle x \rangle . P \xrightarrow{\text{c}\langle x \rangle} P \\
\text{PAR} & : \quad \text{c} \langle x \rangle . P \mid \{M/x\} \xrightarrow{\text{c}\langle x \rangle} P \mid \{M/x\} \\
\text{OPEN-ATOM} & : \quad \nu x. (\text{c}\langle x \rangle . P \mid \{M/x\}) \xrightarrow{\nu x. \text{c}\langle x \rangle} P \mid \{M/x\} \equiv P \mid \{M/x\} \\
\text{STRUCT} & : \quad \text{c}\langle M \rangle . P \xrightarrow{\nu x. \text{c}\langle x \rangle} P \mid \{M/x\}
\end{align*}
\]
The process

\[ A \equiv \nu s. (c(x). \text{if } x = s \text{ then } \bar{c}(i\_got\_s)) \]

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

\[ A \rightarrow^* \alpha \rightarrow^* \cdots \rightarrow^* \alpha \rightarrow^* B \mid \{ i\_got\_s / y \} \]

for some process \( B \) and variable \( y \).
Consider $A''$:

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

This test can succeed; the process can output $\text{i\_got\_s}$, as shown by the following execution:

$$
\begin{align*}
A'' & \xrightarrow{\nu y. \overline{c}(y)} \nu s. (c(x). \text{if } x = s \text{ then } \overline{c \langle \text{i\_got\_s} \rangle} | \{\text{senc}(k, s)/y\}) \\
& \xrightarrow{c(\text{sdec}(k, y))} \nu s. (\text{if } \text{sdec}(k, y) = s \text{ then } \overline{c \langle \text{i\_got\_s} \rangle} | \{\text{senc}(k, s)/y\}) \\
& \equiv \nu s. (\text{if } s = \text{senc}(k, s) \text{ then } \overline{c \langle \text{i\_got\_s} \rangle} | \{\text{senc}(k, s)/y\}) \\
& \equiv \nu s. (\text{if } s = s \text{ then } \overline{c \langle \text{i\_got\_s} \rangle} | \{\text{senc}(k, s)/y\}) \\
& \rightarrow \nu s. (\overline{c \langle \text{i\_got\_s} \rangle} | \{\text{senc}(k, s)/y\}) \\
& \xrightarrow{\nu z. \overline{c}(z)} \nu s. (\{\text{senc}(k, s)/y\} | \{\text{i\_got\_s}/z\}) \\
& \equiv \nu s. (\{\text{senc}(k, s)/y\} | \{\text{i\_got\_s}/z\})
\end{align*}
$$
Five steps to verification

1. Write equations to capture cryptographic primitives
   - For example: $\text{dec}(x, \text{enc}(x, y)) = y$

2. Decide which participants are honest/dishonest

3. Model the honest parties as processes. Example:

   ```
   \text{processA =}
   \begin{align*}
   &\text{new } k; \\
   &\text{in}(c, m); \\
   &\text{out}(c, \text{sign}(k, m));
   \end{align*}
   ```

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
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
1. **Encryption and signatures**

   \[
   sdec(x, senc(x, y)) = y \\
   dec(x, enc(pk(x), y)) = y \\
   \text{checksign}(pk(x), sign(x, y)) = \text{ok}
   \]

2. **Blind signatures**

   \[
   \text{unblind}(r, \text{sign}(x, \text{blind}(r, y))) = \text{sign}(x, y)
   \]

3. **Designated verifier proof of re-encryption**

   The term \(dvp(x, \text{rencrypt}(r, x), r, pkv)\) represents a proof designated for the owner of \(pkv\) that \(x\) and \(\text{rencrypt}(x, r)\) have the same plaintext.

   \[
   \text{checkdvp}(dvp(x, \text{rencrypt}(r, x), r, pkv), x, \text{rencrypt}(r, x), pkv) = \text{ok}
   \]

   \[
   \text{checkdvp}(dvp(x, y, z, skv), x, y, pk(skv)) = \text{ok}.
   \]

4. **Zero knowledge proofs of knowledge...**
Original handshake protocol:

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

in (ch, m);
0.
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.
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)).
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 $\text{fn}(I) \cap \text{bn}(P) = \emptyset$, there is no evaluation context $C[\_]$ with channel $c \notin \text{bn}(C)$ 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 \( \text{sign}_{skS}(k) \) and \( k \)
- **I** replays \( \text{sign}_{skS}(k) \) in a session with **S**
- **I** is able to output secret \( s \)

**Adversary process \( I \)**

\[
\text{in (c, } xPK) ; \\
\text{out (c, } pkM) ; \\
\text{in (c, } y) ; \\
\text{let } \text{sig} = \text{dec}_{skM}(y) \text{ in} \\
\text{out (c, } \text{enc}_{xPK}(\text{sig})) ; \\
\text{in (c, } z) ; \\
\text{out (c, } \text{sdec}_{\text{getmsg}(\text{sig})}(z)) \\
\]
By annotating processes with events $\tilde{\mathcal{f}}(M)$, relationships between the order of events and their parametrisation $M$ can be studied.

**Annotated server process**

```plaintext
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 $\text{compS}(k)$ 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:

\[ f\langle M \rangle \rightsquigarrow g\langle N \rangle. \]

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* \( f\langle M \rangle \rightsquigarrow g\langle N \rangle \) if for all execution paths

\[
A_0 \rightarrow^* \alpha_1 \rightarrow^* A_1 \rightarrow^* \alpha_2 \rightarrow^* \cdots \rightarrow^* \alpha_n \rightarrow^* A_n,
\]

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

\[
\text{compC} \langle \text{pair}(x, y) \rangle \rightsquigarrow \text{startedS} \langle \text{pair}(x, y) \rangle.
\]

Server wants authentication of client that started session:

\[
\text{compS} \langle y \rangle \rightsquigarrow \text{startedC} \langle y \rangle.
\]
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

Consider two honest voters and swap their votes

**Privacy in electronic voting**

A voting protocol respects privacy if

$$S[V_A\{a/v\} | V_B\{b/v\}] \approx S[V_A\{b/v\} | V_B\{a/v\}].$$
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 $A \mathcal{R} B$ 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 $A' \mathcal{R} B'$;

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 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 $\text{dom}(\varphi) = \text{dom}(\psi)$ and for all terms $M, N$ such that $(\tilde{m} \cup \tilde{n}) \cap (\text{fn}(M) \cup \text{fn}(N)) = \emptyset$, 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.
- $\nu m.\{m/x\} \approx_s \nu n.\{\text{hash}(n)/x\}$.
- $\{m/x\} \not\approx_s \{\text{hash}(m)/x\}$. LHS satisfies $x = m$.
- $\nu s.\{\text{pair}(s, s)/x\} \not\approx_s \nu s.\{s/x\}$. LHS satisfies $\text{pair}(\text{fst}(x), \text{snd}(x)) = x$. 
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 $R$ on closed extended processes such that $A R B$ implies:

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

Abadi & Fournet state that observational equivalence and labelled bisimilarity coincide.
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!
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 $\varphi$ 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 $x \notin \text{dom}(\varphi)$.

**Guessing attacks on processes**

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

\[ P \overset{\triangle}{=} \nu s. (!P_A \mid !P_B) \]

\[ P_A \overset{\triangle}{=} \nu n. \overline{c}(\langle \text{comm, } n, \text{mac}(s, \langle \text{comm, } n \rangle) \rangle) \]

\[ P_B \overset{\triangle}{=} c(x). \text{if } 3rd(x) = \text{mac}(s, \langle 1st(x), 2nd(x) \rangle) \text{ then } \overline{c}\langle \text{resp} \rangle \]

where

\[ (M_1, \ldots, M_n) = \text{pair}(M_1, \text{pair}(M_2, \text{pair}(\ldots, \text{pair}(M_n, \ast) \ldots))) \]

\[ 1st(M) = \text{fst}(M) \]

\[ 2nd(M) = \text{fst}(\text{snd}(M)) \]

\[ 3rd(M) = \text{fst}(\text{snd}(\text{snd}(M))) \]
\( P \triangleq \nu s.(!P_A | !P_B) \)

\( P_A \triangleq \nu n.\overline{c}\langle (\text{comm, } n, \text{mac}(s, (\text{comm}, n))) \rangle \)

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

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

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

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

\[
    \nu s.\nu n.(\{(\text{comm, } n, \text{mac}(s, (\text{comm}, n))) \rangle / x \mid \{s/ z\}) \\
    \not\approx_s \nu s'.\nu s.\nu n.(\{(\text{comm, } n, \text{mac}(s, (\text{comm}, n))) \rangle / x \mid \{s'/ z\})
\]

as witnessed by the test

\[
    3\text{rd}(x) = \text{mac}(z, (1\text{st}(x), 2\text{nd}(x))).
\]
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

\[
\text{unblind}(x, \text{sign}(y, \text{blind}(x, z))) = \text{sign}(y, z)
\]
$P_V \triangleq \nu n. \nu r. \text{let } bvn = \text{blind}(r, \text{pair}(v, n)) \text{ in }$
\[\overline{c}\langle \text{pair}(\text{pk}_{V}, \text{sign}(\text{sk}_{V}, bvn))\rangle.\]
\[c(x).\text{if checksign}(pk_{O}, x) = \text{true} \text{ then if getmsg}(x) = bvn \text{ then synch. } \overline{c}\langle \text{unblind}(r, x)\rangle\]

$P_O \triangleq c(y).\text{if checksign}(\text{fst}(y), \text{snd}(y)) = \text{true} \text{ then if Eligible(\text{fst}(y)) = true then }$
\[\overline{c}\langle \text{sign}(\text{sk}_{O}, \text{getmsg}(\text{snd}(y)))\rangle.\]
\[c(w).\text{if checksign}(\text{sk}_{O}, w) = \text{true} \text{ then if NotSeen(w) = true then } \overline{vote}\langle \text{fst}(\text{getmsg}(w))\rangle\]
\[ P \triangleq \nu sk_1 \ldots \nu sk_n. \nu sk_O. \]
\[
\text{let } pk_1 = pksk_1 \text{ in } \ldots \text{ let } pk_n = pksk_V \text{ in }
\text{let } pk_O = pksk_O \text{ in }
(\overline{c}\langle pk_1 \rangle | \cdots | \overline{c}\langle pk_n \rangle | \overline{c}\langle pk_O \rangle | P_V\{sk_1/sk_V, v_1/v\} | \cdots | P_V\{sk_n/sk_V, v_n/v\} | !P_O | S)
\]
\[
\text{synch } \triangleq \overline{\text{syn}}\langle \ast \rangle.\text{syn}'(o)
\]
\[
S \triangleq \text{syn}(x_1) \ldots \text{syn}(x_n).\overline{\text{syn}}\langle \ast \rangle \ldots \overline{\text{syn}}\langle \ast \rangle
\]
The ballot secrecy property is written as the equivalence

$$\nu \text{syn.}\nu \text{syn'} (P \{ s^A_{sk} / sk, v_a / v \} \parallel P \{ s^B_{sk} / sk, v_b / v \} \parallel S)$$

$$\approx_1 \nu \text{syn.}\nu \text{syn'} (P \{ s^A_{sk} / sk, v_b / v \} \parallel P \{ s^B_{sk} / sk, v_a / v \} \parallel S)$$

**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
  
  \[ X \equiv P_i \{ s^A_{sk} / sk, v_a / v, w / y, M / m \} \parallel P_j \{ s^B_{sk} / sk, v_b / v, z / y, N / m \} \parallel S_{i,j}, \]
  \[ Y \equiv P_i \{ s^A_{sk} / sk, v_b / v, w / y, M / m \} \parallel P_j \{ s^B_{sk} / sk, v_a / v, z / y, N / m \} \parallel S_{i,j}; \]

  and if $i = 4$ then $\text{checksign}(pk_O, M) = \text{true}$, and if $j = 4$ then $\text{checksign}(pk_O, N) = \text{true}$, or

- there exist integers $i, j$, and variables $s, t, w, z$ with $6 \leq i, j \leq 8$ and
  
  \[ X \equiv P_i \{ s^A_{sk} / sk, v_a / v, w / y, s / u \} \parallel P_j \{ s^B_{sk} / sk, v_b / v, z / y, t / u \} \parallel S_{i,j}, \]
  \[ Y \equiv P_j \{ s^A_{sk} / sk, v_b / v, w / y, t / u \} \parallel P_i \{ s^B_{sk} / sk, v_a / v, z / y, s / u \} \parallel S_{i,j}. \]
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], . . .