Models and Techniques for Symbolic Analysis of Security Protocols
Episode II: Equivalence properties

Steve Kremer

Summer School: Models and Tools for Cryptographic Proofs
Episode I

Part 1 Protocols
Part 2 Model: the applied-pi calculus
   \(\rightarrow\) the ProVerif tool
Part 3 Analysis: protocols as Horn clauses

Episode II

Part 4 Indistinguishability properties in the applied pi calculus
Part 5 Applications: modelling security protocols
Part 6 Automated analysis: ProVerif & DEEPSEC
Part I

Indistinguishability properties in the applied pi calculus
Symbolic models for protocol verification

Main ingredient of symbolic models

- messages = terms

- perfect cryptography (equational theories)

\[
\begin{align*}
\text{dec}(\text{enc}(x, y), y) &= x \\
\text{fst}(\text{pair}(x, y)) &= x \\
\text{snd}(\text{pair}(x, y)) &= y
\end{align*}
\]

- the network is the attacker
  - messages can be eavesdropped
  - messages can be intercepted
  - messages can be injected

Dolev, Yao: On the Security of Public Key Protocols. FOCS'81
Modelling the protocol

Protocols modelled in a process calculus, e.g. the applied pi calculus

\[
P ::= 0 \\
| c(x).P & \text{input} \\
| \overline{c}\langle t\rangle.P & \text{output} \\
| \text{if } t_1 = t_2 \text{ then } P \text{ else } Q & \text{conditional} \\
| P \parallel Q & \text{parallel} \\
| !P & \text{replication} \\
| \text{new } n.P & \text{restriction}
\]
Modelling the protocol

Protocols modelled in a process calculus, e.g. the applied pi calculus

\[
P ::= 0 \mid c(x).P \mid \overline{c}\langle t \rangle .P \mid \text{input} \mid \text{output} \mid \text{conditional} \mid \text{parallel} \mid \text{replication} \mid \text{restriction}
\]

Specificities:

- messages are terms (not just names as in the pi calculus)
- equality in conditionals interpreted modulo an equational theory
Reasoning about attacker knowledge

Terms output by a process are organised in a frame:

$$\phi = \text{new } \bar{n}. \{ t_1/x_1, \ldots, t_n/x_n \}$$
Reasoning about attacker knowledge

Terms output by a process are organised in a frame:

\[ \phi = \text{new } \vec{n}. \{ t_1/x_1, \ldots, t_n/x_n \} \]

Deducibility:
\[ \phi \vdash^R t \text{ if } R \text{ is a public term and } R\phi =_E t \]

Example

\[ \varphi = \text{new } n_1, n_2, k_1, k_2. \{ \text{enc}(n_1,k_1)/x_1, \text{enc}(n_2,k_2)/x_2, k_1/x_3 \} \]

\[ \varphi \vdash^{\text{dec}(x_1,x_3)} n_1 \quad \varphi \not\vdash n_2 \quad \varphi \vdash^1 1 \]
Deduction may not be sufficient!

Some properties not captured by the terms an attacker can deduce.

**Example**

Consider 2 observations by an attacker

\[ \varphi_1 = \{a/x_1, 0/x_2, 1/x_3, \langle a, 0 \rangle/x_4 \} \]

\[ \varphi_2 = \{a/x_1, 0/x_2, 1/x_3, \langle a, 1 \rangle/x_4 \} \]

**Note:** set of terms deducible from both frames are identical.
Deduction may not be sufficient!

Some properties not captured by the terms an attacker can deduce.

Example

Consider 2 observations by an attacker

$$\varphi_1 = \{ a/x_1, 0/x_2, 1/x_3, \langle a,0 \rangle/x_4 \}$$

$$\varphi_2 = \{ a/x_1, 0/x_2, 1/x_3, \langle a,1 \rangle/x_4 \}$$

Note: set of terms deducible from both frames are identical.

But the attacker may learn the link between $a$ and either $0$ or $1$. 
Deduction may not be sufficient!

Some properties not captured by the terms an attacker can deduce.

**Example**

Consider 2 observations by an attacker

\[
\varphi_1 = \{ a/x_1, 0/x_2, 1/x_3, \langle a,0 \rangle/x_4 \}
\]

\[
\varphi_2 = \{ a/x_1, 0/x_2, 1/x_3, \langle a,1 \rangle/x_4 \}
\]

**Note:** set of terms deducible from both frames are identical.

But the attacker may learn the link between \( a \) and either 0 or 1.

Such properties are captured by the notion of **indistinguishability**: an attacker is unable to distinguish two frames.
From authentication to privacy

Many good tools:

**AVISPA, Casper, Maude-NPA, ProVerif, Scyther, Tamarin, ...**

Good at verifying **trace properties** (predicates on system behavior), e.g.,

- (weak) secrecy of a key
- authentication (correspondence properties)

\[ If \ B \ ended \ a \ session \ with \ A \ (and \ parameters \ p) \ then \ A \ must \ have \ started \ a \ session \ with \ B \ (and \ parameters \ p'). \]
From authentication to privacy

Many good tools:

**AVISPA, Casper, Maude-NPA, ProVerif, Scyther, Tamarin, ...**

Good at verifying *trace properties* (predicates on system behavior), e.g.,

- (weak) secrecy of a key
- authentication (correspondence properties)
  
  *If B ended a session with A (and parameters p) then A must have started a session with B (and parameters p').*

Not all properties can be expressed on a trace.

⇝ recent interest in **indistinguishability properties**.
Indistinguishability (informally)

Can the adversary **distinguish two situations**, i.e. decide whether it is interacting with protocol P1 or protocol P2?
Distinguishing messages

The notion of indistinguishability of message sequences is formalised by **static equivalence** of frames.

**Idea**: any test an attacker can perform on one frame should also hold in the other frame.

**Definition (static equivalence)**

\[ \phi_1 \sim_s \phi_2 \text{ if } \forall \text{ public terms } R, R'. \]

\[ R\phi_1 = R'\phi_1 \iff R\phi_2 = R'\phi_2 \]
Static equivalence: examples

Example

\[ \varphi_1 = \{0/x, 1/y\} \text{ and } \varphi_2 = \{1/x, 0/y\} \]
Static equivalence: examples

Example

\[ \varphi_1 = \{0/x, 1/y\} \text{ and } \varphi_2 = \{1/x, 0/y\} \]

\[ \varphi_1 \neq_s \varphi_2 \text{ as } (x = 0) \varphi_1 \text{ while } (x \neq 0) \varphi_2. \]
Static equivalence: examples

Example

\[ \varphi_1 = \{ \begin{array}{c} 0 / x, 1 / y \end{array} \} \text{ and } \varphi_2 = \{ \begin{array}{c} 1 / x, 0 / y \end{array} \} \]

\[ \varphi_1 \not\sim_s \varphi_2 \text{ as } (x = 0) \varphi_1 \text{ while } (x \neq 0) \varphi_2. \]

Example

\[ \varphi_1 = \nu k \{ \text{aenc}(0, \text{pk}(k)) / x, \text{pk}(k) / y \} \]
\[ \varphi_2 = \nu k \{ \text{aenc}(1, \text{pk}(k)) / x, \text{pk}(k) / y \} \]
Static equivalence: examples

Example

\( \varphi_1 = \{0/x, 1/y\} \) and \( \varphi_2 = \{1/x, 0/y\} \)

\( \varphi_1 \not\sim_s \varphi_2 \) as \((x = 0)\varphi_1\) while \((x \neq 0)\varphi_2\).

Example

\( \varphi_1 = \nu k\{aenc(0, pk(k)) / x, pk(k) / y\} \quad \varphi_2 = \nu k\{aenc(1, pk(k)) / x, pk(k) / y\} \)

\( \varphi_1 \not\sim_s \varphi_2 \) as \((aenc(0, y) = x)\varphi_1\) while \((aenc(0, y) \neq x)\varphi_2\)
Static equivalence: examples

Example

\[ \phi_1 = \{0/x, 1/y\} \text{ and } \phi_2 = \{1/x, 0/y\} \]

\[ \phi_1 \not\sim_s \phi_2 \text{ as } (x = 0)\phi_1 \text{ while } (x \neq 0)\phi_2. \]

Example

\[ \phi_1 = \nu k\{\text{aenc}(0, pk(k))/x, pk(k)/y\} \]
\[ \phi_2 = \nu k\{\text{aenc}(1, pk(k))/x, pk(k)/y\} \]

\[ \phi_1 \not\sim_s \phi_2 \text{ as } (\text{aenc}(0, y) = x)\phi_1 \text{ while } (\text{aenc}(0, y) \neq x)\phi_2 \]

Need to model randomisation of encryption.

\[ \phi_1 = \nu k, r\{\text{aenc}(0, r, pk(k))/x, pk(k)/y\} \]
\[ \phi_2 = \nu k, r\{\text{aenc}(1, r, pk(k))/x, pk(k)/y\} \]

Then \( \phi_1 \sim_s \phi_2 \).
Before defining indistinguishability of processes, we need a precise semantics!

A configuration is a triple:

$$(\mathcal{E}, \mathcal{P}, \varphi)$$

- $\mathcal{E}$ is the set of restricted names;
- $\mathcal{P}$ is the multiset of processes executed in parallel;
- $\varphi$ is the frame of output messages (ignored in internal reduction)
Before defining indistinguishability of processes, we need a precise semantics!

A configuration is a triple:

$$(\mathcal{E}, \mathcal{P}, \varphi)$$

- $\mathcal{E}$ is the set of restricted names;
- $\mathcal{P}$ is the multiset of processes executed in parallel;
- $\varphi$ is the frame of output messages (ignored in internal reduction)

Initial configuration for process $P$: $(\emptyset, \{P\}, \emptyset)$
Operational semantics: internal reduction

Internal reduction → is defined by rules (selection):

\[(\mathcal{E}, \mathcal{P} \cup \{\{0\}\}) \xrightarrow{\varepsilon} (\mathcal{E}, \mathcal{P})\] (NULL)

\[(\mathcal{E}, \mathcal{P} \cup \{\{P \mid Q\}\}) \xrightarrow{\varepsilon} (\mathcal{E}, \mathcal{P} \cup \{\{P, Q\}\})\] (PAR)

\[(\mathcal{E}, \mathcal{P} \cup \{\{\text{new } n.P\}\}) \xrightarrow{\varepsilon} (\mathcal{E} \cup \{n'\}, \mathcal{P}\{n' / n\})\] (NEW)

\[\text{if } n' \text{ fresh}\]

\[(\mathcal{E}, \mathcal{P} \cup \{\{\text{if } u = v \text{ then } P \text{ else } Q\}\}) \xrightarrow{\varepsilon} (\mathcal{E}, \mathcal{P} \cup \{\{P\}\})\] (THEN)

\[\text{if } u =_E v\]

\[(\mathcal{E}, \mathcal{P} \cup \{\{\overline{u}(t).P, v(x).Q\}\}) \xrightarrow{\varepsilon} (\mathcal{E}, \mathcal{P} \cup \{\{P, Q\{t / x\}\}\})\] (COMM)

\[u =_E v\]
Indistinguishability as a process equivalence

Naturally modelled using *equivalences* from process calculi

**Testing equivalence** \((P \approx Q)\)
for all processes \(A\), we have that:

\[
A \mid P \downarrow c \text{ if, and only if, } A \mid Q \downarrow c
\]

\[
\mapsto P \downarrow c \text{ when } P \text{ can send a message on the channel } c.
\]
Indistinguishability as a process equivalence

Naturally modelled using **equivalences** from process calculi

**Testing equivalence** \((P \approx Q)\)

for all processes \(A\), we have that:

\[
A \mid P \Downarrow c \text{ if, and only if, } A \mid Q \Downarrow c
\]

\[
\longrightarrow P \Downarrow c \text{ when } P \text{ can send a message on the channel } c.
\]

**Example**

\[
P = \text{new } k.c(x).c(\langle \text{enc}(x, k) \rangle).c(\langle k \rangle)
\]

\[
Q = \text{new } k.c(x).c(\langle \text{enc}(0, k) \rangle).c(\langle k \rangle)
\]

\(P \not\approx Q\) as \(A \mid P \Downarrow d\), but \(A \mid Q \not\Downarrow d\) for

\[
A = c(1).c(y).c(z). \text{ if dec}(y, z) = 1 \text{ then } \overline{d}(1)
\]
Labelled semantics

Reasoning about all processes A not convenient.

Extend $\epsilon \rightarrow$ to directly interact with a (non specified) adversary.

$$(\mathcal{E}, \mathcal{P} \cup \{u(x).P\}, \Phi) \xrightarrow{\xi(\zeta)} (\mathcal{E}, \mathcal{P} \cup \{P\{\zeta\Phi/x\}\}, \Phi)$$

(IN) if $\nu\mathcal{E}.\Phi \vdash^\xi u$

$$(\mathcal{E}, \mathcal{P} \cup \{\overline{u(t)}.P\}, \Phi) \xrightarrow{\overline{\xi}(\mathbf{ax}_n)} (\mathcal{E}, \mathcal{P} \cup \{P\}, \Phi \cup \{t/\mathbf{ax}_n\})$$

(OUT) if $\nu\mathcal{E}.\Phi \vdash^\xi u$ and $n = |\Phi| + 1$
Labelled semantics

Reasoning about all processes $A$ not convenient.

Extend $\epsilon$ to directly interact with a (non specified) adversary.

$$(\mathcal{E}, \mathcal{P} \cup \{u(x).\mathcal{P}\}, \Phi) \xrightarrow{\xi(\zeta)} (\mathcal{E}, \mathcal{P} \cup \{\mathcal{P}\{\zeta^{\Phi} / x\}\}, \Phi)$$  \hspace{1cm} \text{(IN)} \hspace{1cm} \text{if } \nu \mathcal{E}.\Phi \vdash \xi \ u$$

$$(\mathcal{E}, \mathcal{P} \cup \{u(t).\mathcal{P}\}, \Phi) \xrightarrow{\xi^{\langle ax_n \rangle}} (\mathcal{E}, \mathcal{P} \cup \{\mathcal{P}\}, \Phi \cup \{t / ax_n\})$$  \hspace{1cm} \text{(OUT)} \hspace{1cm} \text{if } \nu \mathcal{E}.\Phi \vdash \xi \ u \text{ and } n = |\Phi| + 1$$

Example

$$P = \text{new } k.c(x).\overline{c}\langle \text{enc}(x, k) \rangle.\overline{c}\langle k \rangle$$

$$(\emptyset, P, \emptyset) \xrightarrow{c(1)} \overline{c}\langle ax_1 \rangle \xrightarrow{\overline{c}\langle ax_2 \rangle} (\{k'\}, \emptyset, \{\text{enc}(1,k') / ax_1, k' / ax_2\})$$

where $\Rightarrow = \epsilon^* \xrightarrow{\epsilon} \epsilon^*$. 
Indistinguishability using labelled semantics

Trace equivalence

\[ P \approx_t Q \]

iff

if \( P \models^{\text{tr}} (E, P, \varphi) \) then \( Q \models^{\text{tr}} (E', Q, \varphi') \land \varphi \sim_s \varphi' \) for some \( E', Q, \varphi' \) (and vice-versa)

Intuition:
Same adversary behaviour (tr) yields indistinguishable frames (\( \sim_s \))
Indistinguishability using labelled semantics

Trace equivalence

\[ P \approx_t Q \]

iff

if \( P \xrightarrow{\text{tr}} (E, P, \varphi) \) then \( Q \xrightarrow{\text{tr}} (E', Q, \varphi') \) \( \land \varphi \sim_s \varphi' \) for some \( E', Q, \varphi' \)

(and vice-versa)

Intuition:
Same adversary behaviour (tr) yields indistinguishable frames (\( \sim_s \))

Example

\[ P = \text{new } k.c(x).\overline{c}\langle\text{enc}(x, k)\rangle.\overline{c}\langle k\rangle \]
\[ Q = \text{new } k.c(x).\overline{c}\langle\text{enc}(0, k)\rangle.\overline{c}\langle k\rangle \]

\( P \not\approx_t Q \) as

\[ P \xrightarrow{c(1)\overline{c}\langle ax_1 \rangle \overline{c}\langle ax_2 \rangle} (\{k'\}, \emptyset, \{\text{enc}(1, k')/ax_1, k' / ax_2 \}) \]

\( \not\sim_s \)

\[ Q \xrightarrow{c(1)\overline{c}\langle ax_1 \rangle \overline{c}\langle ax_2 \rangle} (\{k'\}, \emptyset, \{\text{enc}(0, k')/ax_1, k' / ax_2 \}) \]
Indistinguishability using labelled semantics

Trace equivalence

\[ P \approx_t Q \]

iff

if \( P \xrightarrow{\text{tr}} (E, P, \varphi) \) then \( Q \xrightarrow{\text{tr}} (E', Q, \varphi') \) \( \land \varphi \sim_s \varphi' \) for some \( E', Q, \varphi' \)

(and vice-versa)

Intuition:
Same adversary behaviour (tr) yields indistinguishable frames (\( \sim_s \))

Theorem

\[ P \approx_t Q \implies P \approx Q \]
A tour to the (equivalence) zoo

Abadi, Fournet. Mobile values, new names, and secure communication. POPL’01
A tour to the (equivalence) zoo

Abadi, Fournet. Mobile values, new names, and secure communication. POPL'01
A tour to the (equivalence) zoo

Blanchet et al.: Automated Verification of Selected Equivalences for Security Protocols. LICS'05
A tour to the (equivalence) zoo

Diff equivalence **too fine grained** for several properties.

Blanchet et al.: Automated Verification of Selected Equivalences for Security Protocols. LICS'05
A tour to the (equivalence) zoo

- testing equiv.
- obs. equiv.
- labelled bisim.
- symbolic bisim.
- diff equiv.

Delaune et al. Symbolic bisimulation for the applied pi calculus. JCS’10
A tour to the (equivalence) zoo

- Testing equiv.
- Observation equiv.
- Labelled bisim.
- Symbolic bisim.

Liu, Lin. A complete symbolic bisimulation for full applied pi calculus. TCS'12
A tour to the (equivalence) zoo

1. testing equiv.
2. obs. equiv.
3. trace equiv.
4. labelled bisim.
5. symbolic bisim.
6. diff equiv.

Cheval et al.: Deciding equivalence-based properties using constraint solving. TCS'13
A tour to the (equivalence) zoo

For a **bounded number of sessions** (no replication).

Cheval et al.: Deciding equivalence-based properties using constraint solving. TCS'13
A tour to the (equivalence) zoo

For a class of **determinate processes**.

Cheval et al.: Deciding equivalence-based properties using constraint solving. TCS'13
Part II

Applications: modelling security protocols
Secrecy in symbolic models

Symbolic analysis: secrecy generally modelled as non-deducibility: 
*the attacker cannot compute the value of the secret*

⇝ **partial leakage** not detected

**Example**
Let \( h \) be a one-way hash function. The protocol

\[
P = \text{new } s.\text{out}(c, h(s))
\]

would be considered to enforce the secrecy of \( s \).
Secrecy as indistinguishability

Stronger notions of secrecy can be defined using indistinguishability

▶ **Strong secrecy** of $s$: [Blanchet'04]

$$\text{in}(c, \langle t_1, t_2 \rangle). P\{^{t_1}s\} \approx \text{in}(c, \langle t_1, t_2 \rangle). P\{^{t_2}s\}$$

*Even if the attacker chooses values $t_1$ or $t_2$ he cannot distinguish whether $t_1$ or $t_2$ was used as the secret.*

▶ **Real-or-random**

$$P; \text{out}(s) \approx P; \text{new } s'. \text{out}(s')$$

*The attacker cannot distinguish whether at the end of the protocol he is given the real secret or a random value.*

⇝ Resistance against **offline guessing attacks**
Modelling resistance against offline guessing attacks

\( \nu w. \phi \) is resistant to guessing attacks against \( w \) iff

\[
\nu w. (\phi \cup \{w/x\}) \sim_s \nu w, w'. (\phi \cup \{w'/x\})
\]

**Intuition:** an attacker cannot distinguish the right guess from a wrong guess

A process \( P \) is resistant against guessing attacks on \( w \) if whenever

\[
(\{w\}, \{\{P\}\}, \emptyset) \xrightarrow{\ell}^* (\mathcal{E}, \mathcal{P}, \varphi)
\]

then \( \varphi \) is resistant to guessing attacks.
Example: EKE protocol [BellovinMerritt92]

\[ A \rightarrow B : \text{enc}(\text{pk}(k), w) \quad (\text{EKE.1}) \]
\[ B \rightarrow A : \text{enc}(\text{aenc}(r, \text{pk}(k)), w) \quad (\text{EKE.2}) \]
\[ A \rightarrow B : \text{enc}(na, r) \quad (\text{EKE.3}) \]
\[ B \rightarrow A : \text{enc}(\langle na, nb \rangle, r) \quad (\text{EKE.4}) \]
\[ A \rightarrow B : \text{enc}(nb, r) \quad (\text{EKE.5}) \]

\[ \phi = \nu k, r, na, nb. \quad \{ \text{enc}(\text{pk}(k), w) / x_1, \text{enc}(\text{aenc}(r, \text{pk}(k)), w) / x_2, \text{enc}(na, r) / x_3, \text{enc}(\langle na, nb \rangle, r) / x_4, \text{enc}(nb, r) / x_5 \} \]

\[ \nu w. (\phi \cup \{ w / x \}) \overset{?}{\sim} s \nu w, w'. (\phi \cup \{ w' / x \}) \]
Example: EKE protocol [BellovinMerritt92]

\[
\text{A } \rightarrow \text{B} : \quad \text{enc}(\text{pk}(k), w) \quad \text{(EKE.1)} \\
\text{B } \rightarrow \text{A} : \quad \text{enc}(\text{aenc}(r, \text{pk}(k)), w) \quad \text{(EKE.2)} \\
\text{A } \rightarrow \text{B} : \quad \text{enc}(na, r) \quad \text{(EKE.3)} \\
\text{B } \rightarrow \text{A} : \quad \text{enc}(\langle na, nb \rangle, r) \quad \text{(EKE.4)} \\
\text{A } \rightarrow \text{B} : \quad \text{enc}(nb, r) \quad \text{(EKE.5)}
\]

\[
\phi = \nu k, r, na, nb. \quad \{\text{enc}(\text{pk}(k), w) / x_1, \text{enc}(\text{aenc}(r, \text{pk}(k)), w) / x_2, \text{enc}(na, r) / x_3, \text{enc}(\langle na, nb \rangle, r) / x_4, \text{enc}(nb, r) / x_5\}
\]

\[
\nu w . (\phi \cup \{w / x\}) \sim_s \nu w, w'. (\phi \cup \{w' / x\})
\]

- holds if we suppose the equation \(\text{enc}(\text{dec}(x, y), y) = x\)
- otherwise the test \(\text{enc}(\text{dec}(x_1, x)) \not=_{\varepsilon} x_1\) distinguishes
Example: EKE protocol [BellovinMerritt92]

\[
\begin{align*}
A \rightarrow B : & \quad \text{enc}(pk(k), w) \quad \text{(EKE.1)} \\
B \rightarrow A : & \quad \text{enc}(\text{aenc}(r, pk(k)), w) \quad \text{(EKE.2)} \\
A \rightarrow B : & \quad \text{enc}(na, r) \quad \text{(EKE.3)} \\
B \rightarrow A : & \quad \text{enc}(\langle na, nb \rangle, r) \quad \text{(EKE.4)} \\
A \rightarrow B : & \quad \text{enc}(nb, r) \quad \text{(EKE.5)}
\end{align*}
\]

\[
\phi = \nu k, r, na, nb. \quad \{ \text{enc}(pk(k), w) / x_1, \text{enc}(\text{aenc}(r, pk(k)), w) / x_2, \text{enc}(na, r) / x_3, \\
\text{enc}(\langle na, nb \rangle, r) / x_4, \text{enc}(nb, r) / x_5 \} 
\]

\[
\nu w. (\phi \cup \{ w / x \}) \quad \sim_s \quad \nu w, w'. (\phi \cup \{ w' / x \}) 
\]

- holds if we suppose the equation \(\text{enc}(\text{dec}(x, y), y) = x\)
- otherwise the test \(\text{enc}(\text{dec}(x_1, x)) \overset{?}{=} \varepsilon \) \(x_1\) distinguishes
- if we add equation \(\text{ispubkey}(pk(x)) = \text{ok}\) we distinguish frames by \(\text{ispubkey}(\text{dec}(x_1, x)) \overset{?}{=} \varepsilon \) \(\text{ok}\)
How to model vote privacy?

How can we model

“the attacker does not learn my vote (0 or 1)”?
How to model vote privacy?

How can we model “the attacker does not learn my vote (0 or 1)”?

- The attacker cannot learn the value of my vote

- The attacker cannot distinguish A votes and B votes:
  \[ V_A(v) \approx V_B(v) \]
  but identities are revealed

- The attacker cannot distinguish A votes 0 and A votes 1:
  \[ V_A(0) \approx V_A(1) \]
  but election outcome is revealed

- The attacker cannot distinguish the situation where two honest voters swap votes:
  \[ V_A(0) || V_B(1) \approx V_A(1) || V_B(0) \]
How to model vote privacy?

How can we model

“the attacker does not learn my vote (0 or 1)”?

- The attacker cannot learn the value of my vote
  - but the attacker knows values 0 and 1
How to model vote privacy?

How can we model “the attacker does not learn my vote (0 or 1)”?

- The attacker cannot learn the value of my vote.

- The attacker cannot distinguish A votes and B votes: $V_A(v) \approx V_B(v)$.

Kremer, Ryan: Analysis of an E-Voting Protocol in the Applied Pi Calculus. ESOP’05
How to model vote privacy?

How can we model “the attacker does not learn my vote (0 or 1)”?

▶ The attacker cannot learn the value of my vote

▶ The attacker cannot distinguish A votes and B votes:
\[ V_A(v) \approx V_B(v) \]

\[ \sim \] but identities are revealed

Kremer, Ryan: Analysis of an E-Voting Protocol in the Applied Pi Calculus. ESOP’05
How to model vote privacy?

How can we model “the attacker does not learn my vote (0 or 1)”?

- The attacker cannot learn the value of my vote

- The attacker cannot distinguish A votes and B votes:
  \[ V_A(v) \approx V_B(v) \]

- The attacker cannot distinguish A votes 0 and A votes 1:
  \[ V_A(0) \approx V_A(1) \]
How to model vote privacy?

How can we model

“the attacker does not learn my vote (0 or 1)”?

▶ The attacker cannot learn the value of my vote

▶ The attacker cannot distinguish A votes and B votes:
  \[ V_A(v) \approx V_B(v) \]

▶ The attacker cannot distinguish A votes 0 and A votes 1:
  \[ V_A(0) \approx V_A(1) \]
  \[ \Rightarrow \text{but election outcome is revealed} \]
How to model vote privacy?

How can we model “the attacker does not learn my vote (0 or 1)”?

- The attacker cannot learn the value of my vote

- The attacker cannot distinguish A votes and B votes:
  \[ V_A(v) \approx V_B(v) \]

- The attacker cannot distinguish A votes 0 and A votes 1:
  \[ V_A(0) \approx V_A(1) \]

- The attacker cannot distinguish the situation where two honest voters swap votes:
  \[ V_A(0) \parallel V_B(1) \approx V_A(1) \parallel V_B(0) \]
The Helios e-voting protocol (MixNet version)

$V_1 \xrightarrow{\text{authenticated channel}} \langle id_1, \text{aenc}(pk_E, r_1, v_1) \rangle$

$\langle id_1, \text{aenc}(pk_E, r_1, v_1) \rangle$$\rightarrow$$\text{BB}$

where $pk_E$ is the election public key and MIX a verifiable mixnet.
The Helios e-voting protocol (MixNet version)

where \( pk_E \) is the election public key and MIX a verifiable mixnet.
The Helios e-voting protocol (MixNet version)

\[ V_1 \xrightarrow{\langle id_1, aenc(pk_E, r_1, v_1) \rangle} \langle id_2, aenc(pk_E, r_2, v_2) \rangle \]
\[ \vdots \]
\[ V_n \xrightarrow{\langle id_n, aenc(pk_E, r_n, v_n) \rangle} \]

where \( pk_E \) is the election public key and MIX a verifiable mixnet.
The Helios e-voting protocol (MixNet version)

\[ \langle \text{id}_1, \text{aenc}(\text{pk}_E, r_1, v_1) \rangle \]
\[ \langle \text{id}_2, \text{aenc}(\text{pk}_E, r_2, v_2) \rangle \]
\[ \vdots \]
\[ \langle \text{id}_n, \text{aenc}(\text{pk}_E, r_n, v_n) \rangle \]

\[ \langle \text{id}_1, \text{aenc}(\text{pk}_E, r_1, v_1) \rangle \]
\[ \langle \text{id}_2, \text{aenc}(\text{pk}_E, r_2, v_2) \rangle \]
\[ \vdots \]
\[ \langle \text{id}_n, \text{aenc}(\text{pk}_E, r_n, v_n) \rangle \]

where \( \text{pk}_E \) is the election public key and MIX a verifiable mixnet.
The Helios e-voting protocol (MixNet version)

where \( pk_E \) is the election public key and MIX a verifiable mixnet.

**Privacy:** \( \text{Helios}(v_1, v_2) \approx_t \text{Helios}(v_2, v_1) \)
The Helios e-voting protocol (MixNet version)

\[
\begin{align*}
V_1 & \xrightarrow{\langle id_1, aenc(pk_E, r_1, v_1) \rangle} \langle id_1, aenc(pk_E, r_1, v_1) \rangle \\
V_2 & \xrightarrow{\langle id_2, aenc(pk_E, r_2, v_2) \rangle} \langle id_2, aenc(pk_E, r_2, v_2) \rangle \\
& \vdots \\
V_n & \xrightarrow{\langle id_n, aenc(pk_E, r_n, v_n) \rangle} \langle id_n, aenc(pk_E, r_n, v_n) \rangle
\end{align*}
\]

where pk_E is the election public key and MIX a verifiable mixnet.

**Privacy:** Helios(\(v_1, v_2\)) \(\not\approx_t\) Helios(\(v_2, v_1\)) \(\rightsquigarrow\) **replay attack!**

*Cortier, Smyth: Attacking and Fixing Helios: An Analysis of Ballot Secrecy. CSF’11*
The Helios e-voting protocol (MixNet version)

where \( \text{pk}_E \) is the election public key and MIX a verifiable mixnet.

**Privacy:** \( \text{Helios}(v_1, v_2) \approx_t \text{Helios}(v_2, v_1) \Rightarrow \text{replay attack!} \)

**Fix:** either use weeding, or zkp that voter knows encryption randomness
Everlasting privacy

Does verifiability decrease vote privacy?
Publishing encrypted votes on the bulletin board may be a threat for vote privacy.

- Future technology and scientific advances may break encryptions
- How long must a vote remain private?
  1 year? 10 years? 100 years? $10^{10}$ years?
- Impossible to predict the necessary key length with certainty:
  typical recommendations for less than 10 years
  (cf www.keylength.com)
Everlasting privacy

Does verifiability decrease vote privacy?
Publishing encrypted votes on the bulletin board may be a threat for vote privacy.

▷ Future technology and scientific advances may break encryptions

▷ How long must a vote remain private?
  1 year? 10 years? 100 years? $10^{10}$ years?

▷ Impossible to predict the necessary key length with certainty:
  typical recommendations for less than 10 years
  (cf www.keylength.com)

〜〜 everlasting privacy: guarantee privacy even if crypto is broken
Modelling everlasting privacy

- Information available in the future: everlasting channels
- Define future attacker capabilities (crypto assumption broken)
  \[ \sim \text{equational theory } E^+ \]
  **Example:** \( \text{break(aenc(pk(x), y, z))} \rightarrow z \)
- Check in **two phases**:
  1. check trace equivalence with \( E \)
  2. check static equivalence with \( E^+ \) on future information

\( \sim \) implemented in AKiSs and ProVerif
Modelling everlasting privacy

- Information available in the future: **everlasting channels**
- Define future attacker capabilities (crypto assumption broken)
  \[ \rightsquigarrow \text{equational theory } E^+ \]
  **Example:** \( \text{break(aenc(pk(x), y, z)) \rightarrow z} \)
- Check in **two phases**:
  1. check trace equivalence with \( E \)
  2. check static equivalence with \( E^+ \) on future information

\[ \rightsquigarrow \text{implemented in AKiSs and ProVerif} \]

**Achieving everlasting privacy:**

- Do not publish encryption on the BB, but only a **perfectly hiding commitment**
- Replace identities by **anonymous credentials** \[ \rightsquigarrow \text{Belenios} \]
How to model unlinkability

Unlinkability [ISO/IEC 15408]:

Ensuring that a user may make multiple uses of a service or resource without others being able to link these uses together.

Applications: e-Passport, mobile phones, RFID tags, . . .
How to model unlinkability

Unlinkability [ISO/IEC 15408]:

*Ensuring that a user may make multiple uses of a service or resource without others being able to link these uses together.*

Applications: e-Passport, mobile phones, RFID tags, ... 

Can be modelled as an equivalence property:

2 sessions of the *same* device ≈ 2 sessions of *different* devices

Arapinis et al. Analysing Unlinkability and Anonymity Using the Applied Pi Calculus. CSF’10

Brusò et al. Formal Verification of Privacy for RFID Systems. CSF’10
Authentication protocol of a RFID tag (KCL)

\[ \text{Is unlinkability satisfied?} \]

\[ \tag{id, k} \mid \tag{id, k} \overset{?}{=} \tag{id, k} \mid \tag{id', k'} \]
Linkability attack

1 Tag
\[ k, id \]

new \( r_2 \)

\[ \langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle \]

ATT

\[ \langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle \]

new \( r_2 \)

\[ (id \oplus r_2) \oplus (h(\langle r_1, k \rangle) \oplus r_2) \overset{?}{=} (\text{id}/\text{id'}) \oplus r_2' \oplus (h(\langle r_1, [k/\text{k'}] \rangle) \oplus r_2') \]

2 Tags
\[ k, id \]

new \( r_2' \)

\[ \langle id \oplus r_2', h(\langle r_1, k \rangle) \oplus r_2' \rangle \]

\[ \langle id' \oplus r_2', h(\langle r_1, k' \rangle) \oplus r_2' \rangle \]

new \( r_2' \)
Part III

Automated analysis : ProVerif
Bi-processes

We want to prove $P_1 \approx P_2$.

In ProVerif $P_1$ and $P_2$ are jointly specified using a bi-process. using the \texttt{choice}[t_1, t_2] operator.

When $P$ contains \texttt{choice}[t_1, t_2]:

- $P_1$ is defined by replacing \texttt{choice}[t_1, t_2] by $t_1$;
- $P_2$ is defined by replacing \texttt{choice}[t_1, t_2] by $t_2$.
Bi-processes

We want to prove $P_1 \approx P_2$.

In ProVerif $P_1$ and $P_2$ are jointly specified using a bi-process using the $\texttt{choice}[t_1, t_2]$ operator.

When $P$ contains $\texttt{choice}[t_1, t_2]$:

- $P_1$ is defined by replacing $\texttt{choice}[t_1, t_2]$ by $t_1$;
- $P_2$ is defined by replacing $\texttt{choice}[t_1, t_2]$ by $t_2$.

**Remark:** Any process can be defined as a bi-process:

if $\texttt{choice}[0,1] = 0$ then $P_1$ else $P_2$

but ProVerif does not succeed on general processes
Diff-equivalence

**Diff equivalence** is a fine-grained equivalence that implies trace equivalence

\[ P \approx_{\text{diff}} Q : \text{taking the same branches in } P \text{ and } Q \text{ implies static equivalence (} \sim \text{ reachability + static equivalence).} \]
Diff-equivalence

**Diff equivalence** is a fine-grained equivalence that implies trace equivalence

\[ P \approx_{\text{diff}} Q \]: taking the same branches in \( P \) and \( Q \) implies static equivalence (\( \sim \) reachability + static equivalence).

Often too fine-grained:

\[ \overline{c}\langle \text{choice}[a, b]\rangle | \overline{c}\langle \text{choice}[b, a]\rangle \] not considered equivalent!
Diff-equivalence

Diff equivalence is a fine-grained equivalence that implies trace equivalence

\( P \approx_{\text{diff}} Q \): taking the same branches in \( P \) and \( Q \) implies static equivalence (\( \sim \) reachability + static equivalence).

Often too fine-grained:

\[ \overline{c}\langle \text{choice}[a, b]\rangle | \overline{c}\langle \text{choice}[b, a]\rangle \] not considered equivalent!

Recent versions include a command equivalence, constructing the bi-process automatically:

\[
\begin{align*}
P &= \overline{c}\langle a\rangle.\overline{c}\langle b\rangle \parallel \overline{c}\langle b\rangle.\overline{c}\langle a\rangle \\
Q &= \overline{c}\langle b\rangle.\overline{c}\langle a\rangle \parallel \overline{c}\langle a\rangle.\overline{c}\langle b\rangle
\end{align*}
\]
equivalence \( P \) \( Q \) fails
Strong flavors of secrecy

**Strong secrecy (non-interference)** of $x$

$$c(x_1).c(x_2).P\{x \mapsto \text{choice}[x_1, x_2]\}$$

(or direct query noninterf $x$)

**Resistance to guessing attacks** of $w$

$$\text{new } w.P.\text{new } w'.\overline{c}\langle\text{choice}[w, w']\rangle$$

(or direct query weaksecret $w$)
Modelling equivalence in Horn clauses

**Reachability properties:** $\text{att}(t)$ models attacker knows $t$

**Equivalence properties:**
$\text{att}'(t_1, t_2)$ models attacker knows $t_1$ in $P_1$ and $t_2$ in $P_2$

$\overline{c}\langle\text{choice}[t_1, t_2]\rangle$ is translated into $\text{att}'(t_1, t_2)$

**Special clauses** for equivalence

\[
\begin{align*}
\text{att}'(x, y) \land \text{att}'(x, y') \land \text{nounif}(y, y') & \rightarrow \text{bad} \\
\text{att}'(x, y) \land \text{att}'(x', y) \land \text{nounif}(x, x') & \rightarrow \text{bad}
\end{align*}
\]

where $\text{nounif}(t, t')$ holds when $t, t'$ cannot be unified.

Equivalence holds when $\text{bad}$ cannot be derived.
Part IV

Automated analysis: DEEPSEC
DEEPSEC: DECIDING EQUIVALENCE PROPERTIES IN SECURITY PROTOCOLS

- **Decision procedure** for trace equivalence (no approximation, but high complexity coNEXP!)
- **Bounded number of sessions** (no replication; otherwise full applied πi)
- Crypto primitives specified by destructor subterm convergent rewrite systems

- Tool implemented in OCaml: https://github.com/DeepSec-prover/deepsec
- Input language similar to (untyped) ProVerif
- Possibility to distribute the verification (on multiple cores and multiple machines)
Destructor subterm convergent rewrite systems

Rewrite rules orient equational theories: \( \ell \rightarrow r \) rather than \( \ell = r \).

- Partition function symbols into constructors and destructors
- Messages do not contain destructors
- Each destructor \( g \) defined by rules \( g(t_1, \ldots, t_n) \rightarrow u \)
- For any rule \( \ell \rightarrow r \) \( r \) is a subterm of \( \ell \) (or constant)

Example

\[
\mathcal{F}_c = \{ \text{enc, pair} \} \quad \mathcal{F}_d = \{ \text{dec, fst, snd} \}
\]

\[
\mathcal{R} = \{ \text{dec(\text{enc}(x, y), y) } \rightarrow x, \text{fst(pair}(x, y)) \rightarrow x, \text{snd(pair}(x, y)) \rightarrow y \}
\]

dec(pair(t_1, t_2)) not a valid message!
Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?
Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still infinite: unbounded number of attacker inputs!
Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still **infinite**: unbounded number of attacker inputs!

**Idea:** represent infinite number of possible inputs symbolically in a constraint system
Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still infinite: unbounded number of attacker inputs!

Idea: represent infinite number of possible inputs symbolically in a constraint system

Example

\( c(x).P \) transitions to \( P \) but keeps a deduction constraint \( X \vdash ? x \)
Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still **infinite**: unbounded number of attacker inputs!

**Idea:** represent infinite number of possible inputs **symbolically** in a **constraint system**

**Example**

\[ c(x).P \] transitions to \( P \) but keeps a deduction constraint \( X \vdash ? x \)

if \( t_1 = t_2 \) then \( P \) else \( Q \) : 2 transitions

- to \( P \) with constraint \( t_1 = ? R \ t_2 \)
- to \( Q \) with constraint \( t_1 \neq ? R \ t_2 \)
Constraint systems

A **constraint system** is a tuple \( \mathcal{C} = (\Phi, D, E^1) \) where:

- \( \Phi = \{ a x_1 \mapsto t_1, \ldots, a x_n \mapsto t_n \} \) is a frame;
- \( D \) is a conjunction of deduction facts \( X \vdash ? x \);
- \( E^1 \) is a conjunction of formulas \( u \equiv_R v \) or \( u \not\equiv_R v \).

A **solution** is a pair of substitutions \( \Sigma, \sigma \) such that

- \( \Phi\sigma \vdash^{X\Sigma} x\sigma \) for all \( X \vdash ? x \in D \)
- \( u\sigma \bowtie v\sigma \) for all \( u \bowtie v \in E^1 \)

**Note:** \( \Sigma \) represents attacker inputs and constraints are such that it completely defines \( \sigma \).
Symbolic semantics

**Symbolic semantics:** associate a constraint system to the process (sample rules)

\[(\mathcal{P} \cup \{\text{if } u = v \text{ then } Q\}, (\Phi, D, E^1)) \xrightarrow{\xi_s} (\mathcal{P} \cup \{Q\}, (\Phi, D, E^1 \land u = \?_R v))\]

\[(\mathcal{P} \cup \{c(x).Q\}, (\Phi, D, E^1)) \xrightarrow{c(X)} (\mathcal{P} \cup \{Q\}, (\Phi, D \land X \vdash \? x, E^1))\]

\[(\mathcal{P} \cup \{\overline{c}(t).Q\}, (\Phi, D, E^1)) \xrightarrow{\overline{c}(\text{ax})} (\mathcal{P} \cup \{Q\}, (\Phi \cup \{\text{ax} \mapsto t\}, D, E^1))\]
Symbolic semantics

**Symbolic semantics**: associate a constraint system to the process (sample rules)

\[(P \cup \{\text{if } u = v \text{ then } Q\}, (\Phi, D, E^1)) \xrightarrow{\varepsilon} (P \cup \{Q\}, (\Phi, D, E^1 \land u = ?_R v))\]

\[(P \cup \{c(x).Q\}, (\Phi, D, E^1)) \xrightarrow{c(X)} (P \cup \{Q\}, (\Phi, D \land X \vdash ?_R x, E^1))\]

\[(P \cup \{\overline{c}(t).Q\}, (\Phi, D, E^1)) \xrightarrow{\overline{c}(\text{ax})} (P \cup \{Q\}, (\Phi \cup \{\text{ax} \mapsto t\}, D, E^1))\]

**Sound**: if \((A, C) \xrightarrow{\ell} (A', C')\) then for any \((\Sigma, \sigma) \in \text{Sol}(C)\) we have that \(A\sigma \xrightarrow{\ell\Sigma} A'\sigma\)

**Complete**: if \((\Sigma, \sigma) \in \text{Sol}(C)\) and \(A\sigma \xrightarrow{\ell\Sigma} A'\) then \((A, C) \xrightarrow{\ell} (A', C')\) and \(\Sigma', \sigma' \in \text{Sol}(C')\) and \(A''\sigma' = A'\)
A simple example

\[ P^b \triangleq c(x). \text{if } x = b \text{ then } \overline{c}(0) \text{ else } \overline{c}(x) \quad b \in \{0, 1\} \]
\[ Q \triangleq c(x). \overline{c}(x) \]
A simple example

\[ P^b \triangleq c(x). \text{if } x = b \text{ then } \overline{c}\langle 0 \rangle \text{ else } \overline{c}\langle x \rangle \quad b \in \{0, 1\} \]

\[ Q \triangleq c(x).\overline{c}\langle x \rangle \]

\[ P^0 \approx_t Q \text{ but } P^1 \not\approx_t Q \text{ (different behavior on input 1)} \]
A simple example

\[ P^b \triangleq c(x). \text{if } x = b \text{ then } \overline{c}(0) \text{ else } \overline{c}(x) \quad b \in \{0, 1\} \]

\[ Q \triangleq c(x).\overline{c}(x) \]

\[ P^0 \approx_t Q \text{ but } P^1 \not\approx_t Q \text{ (different behavior on input 1)} \]

Symbolic transitions tree:

\[ (P^b_0, C_\emptyset) \xrightarrow{s} (P^b_1, C_1) \xrightarrow{s} (P^b_2, C_2) \xrightarrow{s} (P^b_4, C_4) \]
\[ (Q_0, C_\emptyset) \xrightarrow{s} (Q_1, C_1) \xrightarrow{s} (Q_2, C_2) \]

\[ C_2 \triangleq (\{ax_1 \rightarrow x\}, X \not\vdash x, \emptyset) \]
\[ C^b_4 \triangleq (\{ax_1 \rightarrow 0\}, X \not\vdash x, x =_R b) \]
\[ C^b_4 \triangleq (\{ax_1 \rightarrow x\}, X \not\vdash x, x \neq_R b) \]
Partition Tree

Build a **joint** symbolic execution tree

**Partition** solutions (split nodes): ensure static equivalences of all solutions in a same node

⇝ done by **constraint solving algorithm**

\[
(Q_0, C_0)
\]
\[
(P_0^0, C_0)
\]
Partition Tree

Build a **joint** symbolic execution tree

**Partition** solutions (split nodes): ensure static equivalences of all solutions in a same node

\[ \sim \] done by *constraint solving algorithm*

\[
\begin{align*}
(Q_0, C_0) & \xrightarrow{c(X)} (Q_1, C_1), (P_1^0, C_1^0) \\
(P_0^0, C_0) & \xrightarrow{s} (P_2^0, C_2^0), (P_3^0, C_3^0)
\end{align*}
\]
**Partition Tree**

Build a *joint* symbolic execution tree

**Partition** solutions (split nodes): ensure static equivalences of all solutions in a same node  
⇝ done by *constraint solving algorithm*

\[
\begin{align*}
(Q_0, C_0) & \xrightarrow{c(X)} s (Q_1, C_1), (P_0^0, C_0^0), (P_2^0, C_2^0), (P_3^0, C_3^0) \xrightarrow{\overline{c(\langle ax_1 \rangle)}} s (Q_2, C_2), (P_4^0, C_4^0), (P_5^0, C_5^0)
\end{align*}
\]

Need to **partition**: \(C_4^0\) enforces \(X = 0\) and \(C_5^0\) enforces \(X \neq 0\).
Partition Tree

Build a **joint** symbolic execution tree

**Partition** solutions (split nodes): ensure static equivalences of all solutions in a same node

⇒ done by **constraint solving algorithm**

\[
\begin{align*}
(Q_0, C_0) \quad & \overset{c(X)}{\longrightarrow} \quad (Q_1, C_1), \ (P_1^0, C_1^0) \\
(P_0^0, C_0) \quad & \overset{s}{\longrightarrow} \quad (Q_2, C_2), \ (P_2^0, C_2^0), \ (P_3^0, C_3^0) \\
\bar{c}(ax_1) \quad & \overset{s}{\longrightarrow} \quad (Q_2, C_2), \ (P_4^0, C_4^0) \\
\quad & \overset{X = 0}{\longrightarrow} \\
\bar{c}(ax_1) \quad & \overset{s}{\longrightarrow} \quad (Q_2, C_2), \ (P_5^0, C_5^0) \\
\quad & \overset{X \neq 0}{\longrightarrow}
\end{align*}
\]

Need to **partition**: \(C_4^0\) enforces \(X = 0\) and \(C_5^0\) enforces \(X \neq 0\).
Partition Tree

Build a **joint** symbolic execution tree

**Partition** solutions (split nodes): ensure static equivalences of all solutions in a same node

⇝ done by **constraint solving algorithm**

\[
\begin{align*}
(Q_0, C_0) & \quad c(X) \quad (Q_1, C_1), (P_0^0, C_1^0) \\
(P_0^0, C_0) & \quad \rightarrow_s (P_2^0, C_2^0), (P_3^0, C_3^0)
\end{align*}
\]

Need to **partition**: \(C_4^0\) enforces \(X = 0\) and \(C_5^0\) enforces \(X \neq 0\). \(P_0^0 \approx_t Q\): each leaf contains processes derived from \(P_0^0\) and \(Q\).
Partition Tree

Build a **joint** symbolic execution tree

**Partition** solutions (split nodes): ensure static equivalences of all solutions in a same node

$\rightsquigarrow$ done by *constraint solving algorithm*

\[
\begin{align*}
(Q_0, C_0) \quad (P_0^1, C_0) & \xrightarrow{c(X)} (Q_1, C_1), (P_1^1, C_1^1) \quad (P_2^1, C_2^1), (P_3^1, C_3^1) \quad (Q_2, C_2), (P_4^1, C_4^1) \\
\end{align*}
\]

Need to **partition more** to ensure static equivalence inside nodes.

$P^1 \not\equiv_t Q$: leaves with processes only from $P^1$. 

Overview of tools

**Unbounded number of sessions** (no termination guarantees)

<table>
<thead>
<tr>
<th></th>
<th><strong>ProVerif</strong></th>
<th><strong>Tamarin</strong></th>
<th><strong>Maude NPA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>equivalence</td>
<td>diff</td>
<td>diff</td>
<td>diff</td>
</tr>
<tr>
<td></td>
<td>(+ extensions)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>protocol</td>
<td>applied pi</td>
<td>MSR (state, loops, ...)</td>
<td>strands</td>
</tr>
<tr>
<td>model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eq. theories</td>
<td>finite variant (?)</td>
<td>subterm conv. + DH</td>
<td>finite variant + algebraic prop.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overview of tools

**Unbounded number of sessions** (no termination guarantees)

<table>
<thead>
<tr>
<th></th>
<th><strong>ProVerif</strong></th>
<th><strong>Tamarin</strong></th>
<th><strong>Maude NPA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>equivalence</td>
<td>diff (+ extensions)</td>
<td>diff</td>
<td>diff</td>
</tr>
<tr>
<td>protocol model</td>
<td>applied pi</td>
<td>MSR (state, loops, . . .)</td>
<td>strands</td>
</tr>
<tr>
<td>eq. theories</td>
<td>finite variant (?)</td>
<td>subterm conv. + DH</td>
<td>finite variant + algebraic prop.</td>
</tr>
</tbody>
</table>

**Bounded number of sessions**

<table>
<thead>
<tr>
<th></th>
<th><strong>SPEC</strong></th>
<th><strong>APTE</strong></th>
<th><strong>AKiSs</strong></th>
<th><strong>DEEPSEC</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>equivalence</td>
<td>symb. bisimulations</td>
<td>trace equiv</td>
<td>~ trace equiv</td>
<td>trace equiv</td>
</tr>
<tr>
<td>protocol model</td>
<td>spi (no else)</td>
<td>applied pi</td>
<td>applied pi</td>
<td>applied pi</td>
</tr>
<tr>
<td>eq. theories</td>
<td>fixed</td>
<td>fixed</td>
<td>finite variant + xor</td>
<td>destructor subterm conv.</td>
</tr>
</tbody>
</table>
## Overview of tools

### Unbounded number of sessions (no termination guarantees)

<table>
<thead>
<tr>
<th></th>
<th>ProVerif</th>
<th>Tamarin</th>
<th>Maude NPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>equivalence</td>
<td>diff (+ extensions)</td>
<td>diff</td>
<td>diff</td>
</tr>
<tr>
<td>protocol model</td>
<td>applied pi</td>
<td>MSR (state, loops, ...)</td>
<td>strands</td>
</tr>
<tr>
<td>eq. theories</td>
<td>finite variant (?)</td>
<td>subterm conv. + DH</td>
<td>finite variant + algebraic prop.</td>
</tr>
</tbody>
</table>

### Bounded number of sessions

<table>
<thead>
<tr>
<th></th>
<th>SPEC</th>
<th>APTE</th>
<th>AKiSs</th>
<th>DEEPSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>equivalence</td>
<td>symb. bisimulations</td>
<td>trace equiv</td>
<td>~ trace equiv</td>
<td>trace equiv</td>
</tr>
<tr>
<td>protocol model</td>
<td>spi (no else)</td>
<td>applied pi</td>
<td>applied pi</td>
<td>applied pi</td>
</tr>
<tr>
<td>eq. theories</td>
<td>fixed</td>
<td>fixed</td>
<td>finite variant + xor</td>
<td>destructor subterm conv.</td>
</tr>
</tbody>
</table>

No swiss knife for equivalence properties