# Verification of cryptographic protocols

## From authentication to privacy

Steve Kremer

INRIA Nancy, LORIA

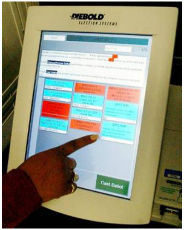
Séminaire Confiance Numérique

# Cryptographic protocols everywhere!

- Distributed programs that
- use cryptographic primitives (encryption, digital signature ,. . . )
- to ensure security properties (confidentiality, authentication, anonymity,. . . )

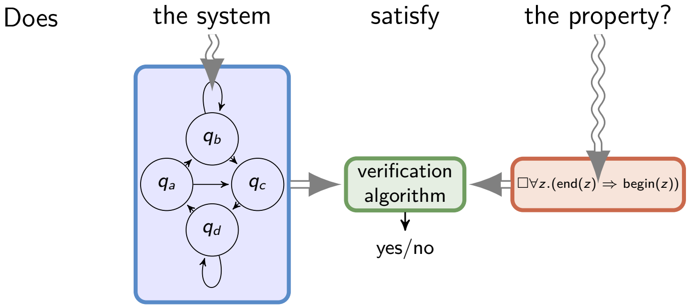

E-commerce

Mobile telephony



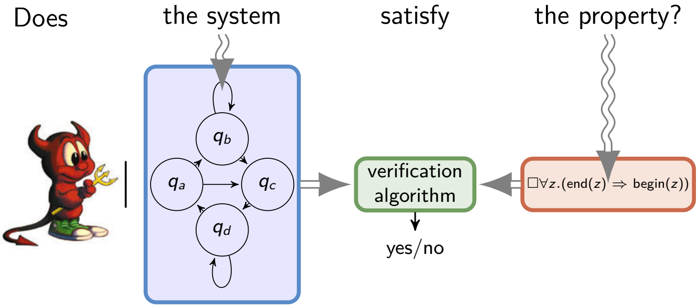

Electronic voting

# Formal verification of critical systems



# Formal verification of critical systems

Applied to security protocols:




Difficulties :

- ⇝ arbitrary attacker controlling the network
- ⇝ infinite state system

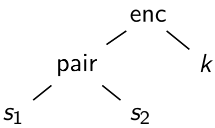
Techniques :

automated deduction, concurrency theory, model-checking, ...

# Symbolic analysis

Symbolic techniques (following [Dolev&Yao'82]):

- messages = terms




- perfect cryptography (equational theories)

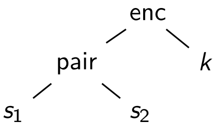
$$\mathrm{dec}(\mathrm{enc}(x, y), y) = x \quad \mathrm{fst}(\mathrm{pair}(x, y)) = x \quad \mathrm{snd}(\mathrm{pair}(x, y)) = y$$

- the network is the attacker

# Symbolic analysis

Symbolic techniques (following [Dolev&Yao'82]):

- messages = terms




- perfect cryptography (equational theories)

$$\mathsf{dec}(\mathsf{enc}(x,y),y) = x \quad \mathsf{fst}(\mathsf{pair}(x,y)) = x \quad \mathsf{snd}(\mathsf{pair}(x,y)) = y$$

- the network is the attacker

Automated tools successfully found flaws in:

- Google's Single Sign-On protocol
- ISO/IEC 9798 standard for entity authentication
- commercial PKCS#11 key-management tokens
- ...

# Automated verification?

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
- correspondence properties

  *If B ended a session with parameter p then A must have started a session with parameters $p'$.*

# Automated verification?

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
- correspondence properties

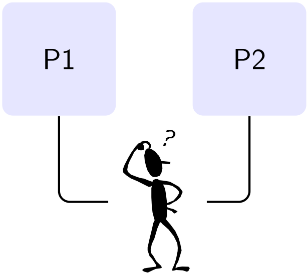
  *If $B$ ended a session with parameter $p$ then $A$ must have started a session with parameters $p'$.*

Not all properties can be expressed on a trace.

$\rightsquigarrow$ 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?




We write $P1 \approx P2$ when the adversary cannot distinguish $P1$ and $P2$

# Indistinguishability in process calculi

Naturally modelled using equivalences from process calculi

e.g. [Spi calculus, Abadi & Gordon'96]
[Applied pi calculus, Abadi & Fournet'01]

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

for all processes $A$, we have that:

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

$\longrightarrow$ $P \Downarrow c$ when $P$ can send a message on the channel $c$.

# Indistinguishability in process calculi

Naturally modelled using equivalences from process calculi

e.g. [Spi calculus, Abadi & Gordon'96]
[Applied pi calculus, Abadi & Fournet'01]

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

for all processes $A$, we have that:

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

$\longrightarrow$ $P \Downarrow c$ when $P$ can send a message on the channel $c$.

**Remarks**

- Process equivalences are well known notions in concurrency theory; much more difficult when adding support for crypto primitives
- A whole zoo of equivalences (with subtle differences)

# A cryptographic process calculus

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

| | | |
|---|---|---|
| $P ::=$ | $0$ | |
| $\mid$ | $\text{in}(c, x).P$ | input |
| $\mid$ | $\text{out}(c, t).P$ | output |
| $\mid$ | $\text{if } t_1 = t_2 \text{ then } P \text{ else } Q$ | conditional |
| $\mid$ | $P \parallel Q$ | parallel |
| $\mid$ | $!P$ | replication |
| $\mid$ | $\text{new } n.P$ | restriction |

# A cryptographic process calculus

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

$$
\begin{array}{rcll}
P ::= & & 0 & \\
& | & \mathrm{in}(c, x).P & \text{input} \\
& | & \mathrm{out}(c, t).P & \text{output} \\
& | & \text{if } t_1 = t_2 \text{ then } P \text{ else } Q & \text{conditional} \\
& | & P \parallel Q & \text{parallel} \\
& | & !P & \text{replication} \\
& | & \text{new } n.P & \text{restriction}
\end{array}
$$

**Specificities:**

- messages are **terms** (not just atomic names as in the pi calculus)
- equality in conditionals interpreted modulo an **equational theory**

# Secrecy in symbolic models

In symbolic analysis secrecy is generally modelled as non-deducibility:

*the attacker cannot compute the value of the secret*

$\rightsquigarrow$ partial leakage is not detected

**Example (Weak secrecy)**

Let $h$ be a one-way hash function. The protocol $P = \nu s.out(c, h(s))$ 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]

$$\mathbf{in}(c, \langle t_1, t_2\rangle).\ P\{^{t_1}/_s\} \approx \mathbf{in}(c, \langle t_1, t_2\rangle).\ P\{^{t_2}/_s\}$$

*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.*

- Resistance against offline guessing attacks (real-or-random): [Corin et al.'05]

$$P; \mathbf{out}(s) \approx P; \nu s'.\mathbf{out}(s')$$

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

# 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

# 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
  $\leadsto$ 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 when we change the voter identity:
  $V_A(v) \approx V_B(v)$

# 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 when we change the voter identity:
  $V_A(v) \approx V_B(v)$
  $\rightsquigarrow$ but identities are 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 when we change the voter identity~~:
  ~~$V_A(v) \approx V_B(v)$~~

- The attacker cannot distinguish when change the vote:
  $V_A(0) \approx V_A(1)$

# 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 when we change the voter identity:~~
  ~~$V_A(v) \approx V_B(v)$~~
- The attacker cannot distinguish when change the vote:
  $V_A(0) \approx V_A(1)$
  $\rightsquigarrow$ 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 when we change the voter identity~~: ~~$V_A(v) \approx V_B(v)$~~
- ~~The attacker cannot distinguish when change the vote~~: ~~$V_A(0) \approx V_A(1)$~~
- The attacker cannot distinguish the situation where two honest voters swap votes:

$$V_A(0) \parallel V_B(1) \approx V_A(1) \parallel V_B(0)$$

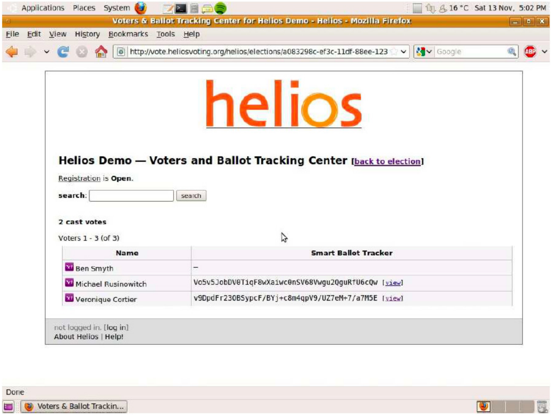
Also avoids the problematic case of unanimity!

[Kremer, Ryan '05]

# The Helios e-voting protocol

Verifiable online elections via the Internet

http://heliosvoting.org/




Already in use:

- Election at Louvain University Princeton
- Election of the IACR board (major association in Cryptography)

# Behavior of Helios (simplified)

Phase 1: voting


**Bulletin Board**

| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |
| Chris | $\{v_C\}_{pk(S)}$ | $v_C = 0$ or $1$ |

$pk(S)$: public key, the private key being shared among trustees.

# Behavior of Helios (simplified)

Phase 1: voting


$\xrightarrow{\{v_D\}_{pk(S)}}$

**Bulletin Board**

| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or 1 |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or 1 |
| Chris | $\{v_C\}_{pk(S)}$ | $v_C = 0$ or 1 |

$pk(S)$: public key, the private key being shared among trustees.

# Behavior of Helios (simplified)

## Phase 1: voting


**Bulletin Board**

| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |
| Chris | $\{v_C\}_{pk(S)}$ | $v_C = 0$ or $1$ |
| David | $\{v_D\}_{pk(S)}$ | $v_D = 0$ or $1$ |

$pk(S)$: public key, the private key being shared among trustees.

# Behavior of Helios (simplified)

Phase 1: voting


**Bulletin Board**

| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |
| Chris | $\{v_C\}_{pk(S)}$ | $v_C = 0$ or $1$ |
| David | $\{v_D\}_{pk(S)}$ | $v_D = 0$ or $1$ |
| ... | ... | |

Phase 2: Tallying using homomorphic encryption (El Gamal)

$$\prod_{i=1}^{n} \{v_i\}_{pk(S)} = \{\sum_{i=1}^{n} v_i\}_{pk(S)}$$

based on $g^a * g^b = g^{a+b}$

$\rightarrow$ Only the final result needs to be decrypted!

$pk(S)$: public key, the private key being shared among trustees.

# This is oversimplified!


**Bulletin Board**

| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |
| Chris | $\{v_C\}_{pk(S)}$ | $v_C = 0$ or $1$ |
| David | $\{v_D\}_{pk(S)}$ | |
| ... | ... | |

**Result**: $\{v_A + v_B + v_C + v_D + \cdots\}_{pk(S)}$

# This is oversimplified!


**Bulletin Board**

| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |
| Chris | $\{v_C\}_{pk(S)}$ | $v_C = 0$ or $1$ |
| David | $\{v_D\}_{pk(S)}$ | $v_D = 100$ |
| ... | ... | |

**Result**: $\{v_A + v_B + v_C + 100 + \cdots\}_{pk(S)}$

A malicious voter can cheat!

# This is oversimplified!


**Bulletin Board**

| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |
| Chris | $\{v_C\}_{pk(S)}$ | $v_C = 0$ or $1$ |
| David | $\{v_D\}_{pk(S)}$ | ~~$v_D = 100$~~ |
| ... | ... | |

**Result**: $\{v_A + v_B + v_C + v_D + \cdots\}_{pk(S)}$

~~A malicious voter can cheat!~~

In Helios: use Zero Knowledge Proof

$$\{v_D\}_{pk(S)}, \mathrm{ZKP}\{v_D = 0 \text{ or } 1\}$$

# A privacy attack on Helios


| Bulletin Board | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |

# A privacy attack on Helios


**Bulletin Board**

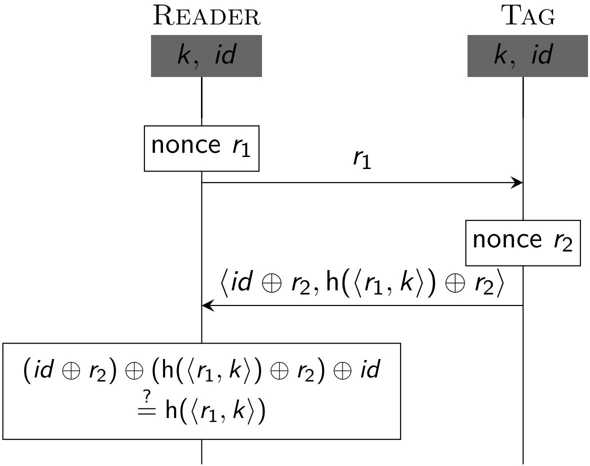
| | | |
|---|---|---|
| Alice | $\{v_A\}_{pk(S)}$ | $v_A = 0$ or $1$ |
| Bob | $\{v_B\}_{pk(S)}$ | $v_B = 0$ or $1$ |
| Chris | $\{v_A\}_{pk(S)}$ | |

Vote-copying attack:
copying Alice's vote introduces a bias in the outcome

Weakness in Helios discovered when trying to prove the previous definition of anonymity

[Cortier, Smyth '11]

# Authentication protocol of a RFID tag




$$P_{\mathrm{tag}} = \mathbf{in}(c, x).\ \mathrm{new}\ r_2.\ \mathbf{out}(c, \langle id \oplus r_2,\ \mathsf{h}(\langle x, k\rangle) \oplus r_2\rangle).\ 0$$

# Untraceability

An attacker must not be able to **link two sessions of a same tag**.

Modelled as an equivalence:

2 sessions of the **same** tag $\approx$ 2 sessions of **different** tags

# Untraceability

An attacker must not be able to **link two sessions of a same tag**.

Modelled as an equivalence:

2 sessions of the **same** tag $\approx$ 2 sessions of **different** tags

**Linkability attack:**

$$\begin{array}{rcl} P_{\mathsf{same}} & = & \mathbf{in}(c,x).\ \mathsf{new}\ r_2.\ \mathbf{out}(c,t).\ \mathbf{in}(c,x).\ \mathsf{new}\ r_2'.\ \mathbf{out}(c,t_{\mathsf{s}}').0 \\ & & \not\approx \\ P_{\mathsf{diff}} & = & \mathbf{in}(c,x).\ \mathsf{new}\ r_2.\ \mathbf{out}(c,t).\ \mathbf{in}(c,x).\ \mathsf{new}\ r_2'.\ \mathbf{out}(c,t_{\mathsf{d}}').0 \end{array}$$

where

$$\begin{array}{rcl} t & = & \langle id \oplus r_2,\ \mathsf{h}(\langle x, k\rangle) \oplus r_2\rangle \\ t_{\mathsf{s}}' & = & \langle id \oplus r_2',\ \mathsf{h}(\langle x, k\rangle) \oplus r_2'\rangle \\ t_{\mathsf{d}}' & = & \langle id' \oplus r_2',\ \mathsf{h}(\langle x, k'\rangle) \oplus r_2'\rangle \end{array}$$

# Untraceability

An attacker must not be able to **link two sessions of a same tag**.

Modelled as an equivalence:

2 sessions of the **same** tag $\approx$ 2 sessions of **different** tags

**Linkability attack:**

$$\begin{array}{rcl} P_{\text{same}} & = & \mathbf{in}(c,x).\ \text{new}\ r_2.\ \mathbf{out}(c,t).\ \mathbf{in}(c,x).\ \text{new}\ r_2'.\ \mathbf{out}(c,t_s').0 \\ & & \not\approx \\ P_{\text{diff}} & = & \mathbf{in}(c,x).\ \text{new}\ r_2.\ \mathbf{out}(c,t).\ \mathbf{in}(c,x).\ \text{new}\ r_2'.\ \mathbf{out}(c,t_d').0 \end{array}$$

where

$$\begin{array}{rcl} t & = & \langle id \oplus r_2,\ \mathsf{h}(\langle x,k\rangle) \oplus r_2\rangle \\ t_s' & = & \langle id \oplus r_2',\ \mathsf{h}(\langle x,k\rangle) \oplus r_2'\rangle \\ t_d' & = & \langle id' \oplus r_2',\ \mathsf{h}(\langle x,k'\rangle) \oplus r_2'\rangle \end{array}$$

distinguished by

$$(\text{proj}_1(t) \oplus \text{proj}_2(t) \stackrel{?}{=} \text{proj}_1(t') \oplus \text{proj}_2(t')$$

# Our goals and approach for verifying equivalence properties

[Chadha, Ciobâcă, K., 2012]

. . . and actively developed since

Decision procedure for **trace equivalence**:

- many equational theories,
- practical implementation

Protocols modelled as **first order Horn clauses** (**bounded number of sessions**, i.e., no replication)

**Resolution based procedure for trace equivalence** for convergent equational theories (in particular **optimally reducing eq. theories**)

# Terms and frames

Messages are modelled as first-order terms equipped with a convergent rewrite system R.

Secret values are modelled as names in a set $\mathcal{N}$.

We write $t =_\mathsf{R} u$ when $t\!\downarrow = u\!\downarrow$

## Example

Signature: senc/3, sdec/2, pair/2, fst/1, snd/1, $\mathbf{0}$/0, $\mathbf{1}$/0

Rewrite system:
$\mathsf{sdec}(\mathsf{senc}(x, y, z), y) \rightarrow_\mathsf{R} x, \mathsf{fst}(\mathsf{pair}(x, y)) \rightarrow_\mathsf{R} x, \mathsf{snd}(pair(x, y)) \rightarrow_\mathsf{R} y$

Terms: $t_1 = \mathsf{senc}(n, k, r)$, $t_2 = \mathsf{sdec}(t_1, k) \quad (n, k, r \in \mathcal{N})$
We have that $t_2 =_\mathsf{R} n$

# Deduction

Sequences of messages are grouped in a frame $\varphi = \{^{t_1}/_{w_1}, \dots, ^{t_n}/_{w_n}\}$

What messages can an attacker compute?

## Definition (Deduction)

A term $t$ is *deducible from frame* $\varphi$ *with a recipe* $r$ ($\varphi \vdash^r t$) if $r\varphi =_{\mathsf{R}} t$ and $r$ does not contain names in $\mathcal{N}$.

## Example

Let $\varphi = \{^{\mathsf{senc}(n_1,k_1,r_1)}/_{w_1}, ^{\mathsf{senc}(n_2,k_2,r_2)}/_{w_2}, ^{k_1}/_{w_3}\}$.

We have that $\varphi \vdash^{\mathsf{sdec}(w_1,w_3)} n_1$, $\varphi \not\vdash n_2$, $\varphi \vdash^{\mathbf{1}} \mathbf{1}$

# Static equivalence

Sequences of messages are grouped in a frame $\varphi = \{^{t_1}/_{w_1}, \ldots, ^{t_n}/_{w_n}\}$

Indistinguishability of sequences of messages

## Definition (Static equivalence)

$(r_1 = r_2)\varphi$ if $\varphi \vdash^{r_1} t$ and $\varphi \vdash^{r_2} t$ for some $t$.

$\varphi_1$ *statically equivalent* to $\varphi_2$ $(\varphi_1 \approx_s \varphi_2)$ iff $(r_1 = r_2)\varphi_1 \Leftrightarrow (r_1 = r_2)\varphi_2$.

## Examples

$$\{^{n_1}/_{w_1}\} \quad \approx_s \quad \{^{n_2}/_{w_1}\}$$

$$\{^{n_1}/_{w_1}, ^{n_2}/_{w_2}\} \quad \not\approx_s \quad \{^{n_1}/_{w_1}, ^{n_1}/_{w_2}\} \qquad (w_1 \stackrel{?}{=} w_2)$$

$$\{^{\mathsf{senc}(\mathbf{0},k,r)}/_{w_1}\} \quad \approx_s \quad \{^{\mathsf{senc}(\mathbf{1},k,r)}/_{w_1}\}$$

$$\{^{\mathsf{senc}(n,k,r)}/_{w_1}, ^{k}/_{w_2}\} \quad \not\approx_s \quad \{^{\mathsf{senc}(\mathbf{0},k,r)}/_{w_1}, ^{k}/_{w_2}\} \qquad (sdec(w_1, w_2) \stackrel{?}{=} \mathbf{0})$$

# A simple crypto process calculus: syntax

Actions : $\mathbf{in}(c, x) \mid \mathbf{out}(c, t) \mid [s \stackrel{?}{=} t]$

Symbolic Trace: sequence of actions

**Example**

$$\begin{aligned} T \;=\; & \mathbf{out}(c, \mathrm{enc}(a, k)).\mathbf{out}(c, \mathrm{enc}(a', k)). \\ & \mathbf{in}(c, x).\mathbf{out}(c, \mathrm{dec}(x, k)). \\ & \mathbf{in}(c, y).[y \stackrel{?}{=} \mathrm{pair}(a, a')].\mathbf{out}(c, s) \end{aligned}$$

Process: set of symbolic traces

Remark: Parallel composition $(P \mid Q)$ can be defined as the set of interleavings

# A simple crypto process calculus: semantics

Operational semantics: $(T, \varphi) \xrightarrow{\ell} (T', \varphi')$

$$\textsc{Receive}\ \frac{\varphi \vdash^r t}{(\mathbf{in}(c,x).T, \varphi) \xrightarrow{\mathbf{in}(c,r)} (T\{x \mapsto t\}, \varphi)}$$

$$\textsc{Test}\ \frac{s =_{\mathsf{R}} t}{([s \stackrel{?}{=} t].T, \varphi) \xrightarrow{\mathbf{test}} (T, \varphi)}$$

$$\textsc{Send}\ \frac{}{(\mathbf{out}(c,t).T, \varphi) \xrightarrow{\mathbf{out}(c)} (T, \varphi \cup \{w_{|dom(\varphi)|+1} \mapsto t\})}$$

$P \xrightarrow{\ell} (T', \varphi)$ if $\exists T \in P.\ (T, \emptyset) \xrightarrow{\ell} (T', \varphi)$

$\stackrel{\ell}{\Longrightarrow}$ if $\xrightarrow{\mathbf{test}^*\ \ell\ \mathbf{test}^*}$: weak semantics hiding silent test actions

# Trace equivalences

**Trace equivalence:** $P \sqsubseteq_t Q$

if $(P, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (P', \varphi)$ then $\exists Q', \varphi'. \ (Q, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi') \wedge \varphi \sim_s \varphi'$

$P \approx Q$ iff $P \sqsubseteq Q \wedge Q \sqsubseteq P$

# Trace equivalences

**Fine grained trace equivalence:** $P \sqsubseteq_{ft} Q$

$\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

**Trace equivalence:** $P \sqsubseteq_t Q$

if $(P, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (P', \varphi)$ then $\exists Q', \varphi'.\ (Q, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi') \wedge \varphi \sim_s \varphi'$

$P \approx Q$ iff $P \sqsubseteq Q \wedge Q \sqsubseteq P$

# Trace equivalences

**Fine grained trace equivalence:** $P \sqsubseteq_{ft} Q$

$\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$$\not\Downarrow\!\cap$$

**Trace equivalence:** $P \sqsubseteq_t Q$

if $(P, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (P', \varphi)$ then $\exists Q', \varphi'.\ (Q, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi') \wedge \varphi \sim_s \varphi'$

$P \approx Q$ iff $P \sqsubseteq Q \wedge Q \sqsubseteq P$

# Trace equivalences

**Fine grained trace equivalence:** $P \sqsubseteq_{ft} Q$

$\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$\not\supseteq \quad \cap$

**Trace equivalence:** $P \sqsubseteq_t Q$

if $(P, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (P', \varphi)$ then $\exists Q', \varphi'.\ (Q, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi') \wedge \varphi \sim_s \varphi'$

**Coarse trace equivalence:** $P \sqsubseteq_{ct} Q$

if $(P, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (P', \varphi) \wedge (r = s)\varphi$ then $\exists Q', \varphi'.\ (Q, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi') \wedge (r = s)\varphi'$

$P \approx Q$ iff $P \sqsubseteq Q \wedge Q \sqsubseteq P$

# Trace equivalences

**Fine grained trace equivalence:** $P \sqsubseteq_{ft} Q$

$\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$$\not\supseteq$$

**Trace equivalence:** $P \sqsubseteq_t Q$

if $(P, \emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (P', \varphi)$ then $\exists Q', \varphi'.\ (Q, \emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (Q', \varphi') \wedge \varphi \sim_s \varphi'$

$$\not\supseteq$$

**Coarse trace equivalence:** $P \sqsubseteq_{ct} Q$

if $(P, \emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (P', \varphi) \wedge (r = s)\varphi$ then $\exists Q', \varphi'.\ (Q, \emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (Q', \varphi') \wedge (r = s)\varphi'$

$P \approx Q$ iff $P \sqsubseteq Q \wedge Q \sqsubseteq P$

# Trace equivalences

**Fine grained trace equivalence:** $P \sqsubseteq_{ft} Q$

$\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$\not\supseteq$

**Trace equivalence:** $P \sqsubseteq_t Q$

if $(P,\emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (P',\varphi)$ then $\exists Q',\varphi'.\ (Q,\emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (Q',\varphi') \wedge \varphi \sim_s \varphi'$

$\not\supseteq$ $\quad ||$ det. proc.

**Coarse trace equivalence:** $P \sqsubseteq_{ct} Q$

if $(P,\emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (P',\varphi) \wedge (r = s)\varphi$ then $\exists Q',\varphi'.\ (Q,\emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (Q',\varphi') \wedge (r = s)\varphi'$

$P \approx Q$ iff $P \sqsubseteq Q \wedge Q \sqsubseteq P$

$P$ is *determinate* if whenever $(P,\emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (T,\varphi)$ and $(P,\emptyset) \xrightarrow{\ell_1,\dots,\ell_n} (T',\varphi')$ then $\varphi \approx_s \varphi'$.

# Our procedure: overview

1. Model protocol and intruder capabilities in Horn clauses
2. Saturate clauses using dedicated resolution procedure
3. Check equivalence

# Our procedure: overview

1. Model protocol and intruder capabilities in Horn clauses
2. Saturate clauses using dedicated resolution procedure
3. Check equivalence

We fail to verify trace equivalence (in general) :-(

- under-approximate trace equivalence ($\approx_{ft}$)
- over-approximate trace equivalence ($\approx_{ct}$)
- verify trace equivalence for determinate processes

# 1. Horn clause modelling: predicates

Predicates: interpreted over ground trace $T$

- Reachability predicate

  $$T \models \mathsf{r}_{\ell_1,\ldots,\ell_n} \quad \text{if } (T,\emptyset) \xrightarrow{L_1} (T_1,\varphi_1) \xrightarrow{L_2} \ldots \xrightarrow{L_n} (T_n,\varphi_n)$$
  $$\text{such that } \ell_i =_{\mathsf{R}} L_i\varphi_{i-1} \text{ for all } 1 \leq i \leq n$$

- intruder Knowledge predicate

  $$T \models \mathsf{k}_{\ell_1,\ldots,\ell_n}(R,t) \quad \text{if } \mathsf{r}_{\ell_1,\ldots,\ell_n} \text{ then } \varphi_n \vdash^{R\sigma} t\sigma$$

- Identity predicate

  $$T \models \mathsf{i}_{\ell_1,\ldots,\ell_n}(R,R') \quad \text{if } \exists t.\ T \models \mathsf{k}_{\ell_1,\ldots,\ell_i}(R,t) \text{ and } T \models \mathsf{k}_{\ell_1,\ldots,\ell_i}(R',t)$$

- Reachable Identity predicate

  $$T \models \mathsf{ri}_{\ell_1,\ldots,\ell_n}(R,R') \quad \text{if } T \models \mathsf{i}_{\ell_1,\ldots,\ell_n}(R,R') \text{ and } T \models \mathsf{r}_{\ell_1,\ldots,\ell_n}$$

# 1. Horn clause modelling: initial clauses

$$T = \mathbf{in}(c, x).[\mathrm{dec}(x, k) \stackrel{?}{=} a].\mathbf{out}(c, s)$$

Compute an initial set for trace $T$: seed($T$)

$$\begin{aligned}
\mathrm{r}_{\mathbf{in}(c,x)} &\Leftarrow \mathrm{k}(X, x) \\
\mathrm{r}_{\mathbf{in}(c,x),\mathbf{test}} &\Leftarrow \mathrm{k}(X, x), \mathrm{dec}(x, k) =_{\mathrm{R}} a \\
\mathrm{r}_{\mathbf{in}(c,x),\mathbf{test},\mathbf{out}(c)} &\Leftarrow \mathrm{k}(X, x), \mathrm{dec}(x, k) =_{\mathrm{R}} a \\
\\
\mathrm{k}_{\mathbf{in}(c,x),\mathbf{test},\mathbf{out}(c)}(w_1, s) &\Leftarrow \mathrm{k}(X, x), \mathrm{dec}(x, k) =_{\mathrm{R}} a \\
\\
\mathrm{k}_w(f(X_1, \ldots X_n), f(x_1, \ldots, x_k)) &\Leftarrow \mathrm{k}_w(X_1, x_1), \ldots, \mathrm{k}_w(X_k, x_k) \\
&\qquad \text{for any function } f
\end{aligned}$$

# 1. Horn clause modelling: getting rid of equations

Use equational unification to remove tests:

$$\Big(H \Leftarrow B_1, \ldots, B_n, u =_{\mathrm{R}} v\Big) \quad \rightsquigarrow \quad \begin{array}{l} \Big((H \Leftarrow B_1, \ldots, B_n)\sigma_1\Big) \\ \ldots \\ \Big((H \Leftarrow B_1, \ldots, B_n)\sigma_k\Big) \end{array}$$

where $\sigma_1, \ldots, \sigma_k$ is a complete set of unifiers for $u =_{\mathrm{R}} v$.

# 1. Horn clause modelling: getting rid of equations

Use equational unification to remove tests:

$$\Big(H \Leftarrow B_1, \ldots, B_n, u =_{\mathsf{R}} v\Big) \quad \rightsquigarrow \quad \begin{array}{l} \Big((H \Leftarrow B_1, \ldots, B_n)\sigma_1\Big) \\ \ldots \\ \Big((H \Leftarrow B_1, \ldots, B_n)\sigma_k\Big) \end{array}$$

where $\sigma_1, \ldots, \sigma_k$ is a complete set of unifiers for $u =_{\mathsf{R}} v$.

## Example

$$\mathsf{r}_{\mathbf{in}(c,x),\mathbf{test},\mathbf{out}(c)} \Leftarrow \mathsf{k}(X, x), \mathsf{dec}(x, k) =_{\mathsf{R}} a$$

$$\rightsquigarrow$$

$$\mathsf{r}_{\mathbf{in}(c,\mathsf{enc}(a,k)),\mathbf{test},\mathbf{out}(c)} \Leftarrow \mathsf{k}(X, \mathsf{enc}(a, k))$$

# 1. Horn clause modelling: getting rid of equations (2)

Use finite variant property ([Comon-Lund, Delaune'05]) to get rid of equational reasoning:

> *Finite variant property: possibility to precompute a finite set of all possible normal forms*

$$\Big(\mathsf{k}_h(R,t) \Leftarrow B_1,\ldots,B_n\Big) \quad \rightsquigarrow \quad \begin{array}{l} \Big((\mathsf{k}_H(R,t))\theta_1{\downarrow} \Leftarrow B_1\theta_1{\downarrow},\ldots,B_n\theta_1{\downarrow}\Big) \\ \ldots \\ \Big((\mathsf{k}_H(R,t))\theta_k{\downarrow} \Leftarrow B_1\theta_k{\downarrow},\ldots,B_n\theta_k{\downarrow}\Big). \end{array}$$

where $\theta_1,\ldots,\theta_k$ is a complete set of variants for $t$.

We can compute finite sets of variants and $\mathsf{mgu}_\mathsf{E}$ for the class of optimally reducing theories (contains subterm convergent, blind sigs, td commitment, ...)

# 2. Saturation: goals of saturation

Saturate seed knowledge base using the following rules

$$\textsc{Resolution}\ \frac{\begin{array}{c} f \in K, g \in K_{\text{solved}}, \qquad f = \left(H \Leftarrow \mathsf{k}_{uv}(X, t), B_1, \dots, B_n\right) \\ g = \left(\mathsf{k}_w(R, t') \Leftarrow B_{n+1}, \dots, B_m\right) \\ \sigma = \mathrm{mgu}(\mathsf{k}_u(X, t), \mathsf{k}_w(R, t')) \qquad t \notin \mathcal{X} \end{array}}{K := K \cup \left((H \Leftarrow B_1, \dots, B_m)\sigma\right)}$$

$$\textsc{Equation}\ \frac{\begin{array}{c} f, g \in K_{\text{solved}}, \qquad f = \left(\mathsf{k}_u(R, t) \Leftarrow B_1, \dots, B_n\right) \\ g = \left(\mathsf{k}_{u'v'}(R', t') \Leftarrow B_{n+1}, \dots, B_m\right) \qquad \sigma = \mathrm{mgu}(\mathsf{k}_u(\_, t), \mathsf{k}_{u'}(\_, t')) \end{array}}{K = K \cup \left((\mathsf{i}_{u'v'}(R, R') \Leftarrow B_1, \dots, B_m)\sigma\right)}$$

$$\textsc{Test}\ \frac{\begin{array}{c} f, g \in K_{\text{solved}}, \\ f = \left(\mathsf{i}_u(R, R') \Leftarrow B_1, \dots, B_n\right) \qquad g = \left(\mathsf{r}_{u'v'} \Leftarrow B_{n+1}, \dots, B_m\right) \qquad \sigma = \mathrm{mgu}(u, u') \end{array}}{K = K \cup \left((\mathsf{ri}_{u'v'}(R, R') \Leftarrow B_1, \dots, B_m)\sigma\right)}$$

# 2. Saturation rules: soundness, completeness, termination

A clause is solved if it is of the form

$$H \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots, \mathsf{k}_{w_n}(X_n, x_n)$$

# 2. Saturation rules: soundness, completeness, termination

A clause is solved if it is of the form

$$H \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots, \mathsf{k}_{w_n}(X_n, x_n)$$

- Sound: If $f \in \mathsf{sat}(\mathsf{seed}(T))$ then $T \models f$

# 2. Saturation rules: soundness, completeness, termination

A clause is solved if it is of the form

$$H \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots, \mathsf{k}_{w_n}(X_n, x_n)$$

- Sound: If $f \in \mathsf{sat}(\mathsf{seed}(T))$ then $T \models f$
- Complete: If $(T, \emptyset) \xrightarrow{L_1,\ldots,L_n} (S, \varphi)$ and $K = \mathsf{sat}(\mathsf{seed}(T))_{\mathsf{solved}}$ then
  1. $\mathsf{r}_{L_1,\ldots,L_n}$ is a consequence of $K$
  2. if $\varphi \vdash^R t$ then $\mathsf{k}_{L_1,\ldots,L_n}(R, t\downarrow)$ is a consequence of $K$
  3. if $\varphi \vdash^R t$ and $\varphi \vdash^{R'} t$, then $\mathsf{i}_{L_1,\ldots,L_n}(R, R')$ is a consequence of $K$

# 2. Saturation rules: soundness, completeness, termination

A clause is solved if it is of the form

$$H \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots, \mathsf{k}_{w_n}(X_n, x_n)$$

- Sound: If $f \in \mathsf{sat}(\mathsf{seed}(T))$ then $T \models f$
- Complete: If $(T, \emptyset) \xrightarrow{L_1,\ldots,L_n} (S, \varphi)$ and $K = \mathsf{sat}(\mathsf{seed}(T))_{\mathsf{solved}}$ then
  1. $\mathsf{r}_{L_1,\ldots,L_n}$ is a consequence of $K$
  2. if $\varphi \vdash^R t$ then $\mathsf{k}_{L_1,\ldots,L_n}(R, t\downarrow)$ is a consequence of $K$
  3. if $\varphi \vdash^R t$ and $\varphi \vdash^{R'} t$, then $\mathsf{i}_{L_1,\ldots,L_n}(R, R')$ is a consequence of $K$
- Termination:
  - guaranteed for subterm convergent equational theories;
  - in practice terminates also on examples outside this class.

# 3. Checking equivalence

To check that $T \sqsubseteq_{ct} Q$

1. saturate: let $K = \mathrm{sat}(\mathrm{seed}(T))_{\mathrm{solved}}$
2. check reachability:
   for each $\mathrm{r}_{L_1,\dots,L_n} \Leftarrow \mathrm{k}_{h_1}(X_1, x_1), \dots \mathrm{k}_{h_k}(X_k, x_k) \in K$
   check that $Q, \emptyset \xrightarrow{L_1,\dots,L_n} Q', \varphi$
3. check equalities:
   for each $\mathrm{ri}_{L_1,\dots,L_n}(R_1, R_2) \Leftarrow \mathrm{k}_{h_1}(X_1, x_1), \dots \mathrm{k}_{h_k}(X_k, x_k) \in K$
   check that $Q, \emptyset \xrightarrow{L_1,\dots,L_n} Q', \varphi$ and $(R_1 = R_2)\varphi$

# The AKISS tool

## AKISS

(Active Knowledge In Security protocolS)

`https://github.com/akiss`

Examples:

- Strong secrecy
  NSL protocol and Blanchet's variant's of Denning-Sacco (det. processes)
- Resistance to offline guessing attacks
  EKE (det. process)
- (Everlasting) Vote privacy: FOO, Okamoto, Helios and Moran-Naor electronic voting protocols
- **New**: support for $\oplus$ (RFID protocols)

# Conclusions

- Process equivalences are the **main tool to model security properties** (except authentication)
- **Theoretical understanding** still rather poor: decidability for which equational theory? Complexity?
- **Tool support** not yet mature enough
  - AKiSs: no else branches, approximates trace equivalence
  - APTE: only fixed equational theory (encryption, signature, hash)
  - ProVerif: unbounded number of sessions, but false attacks may occur
- WIP: a new procedure that takes **the best of both AKiSs and APTE**:

  real trace equivalence + else branches + many equational theories