A Method for Verifying Privacy-Type Properties: The Unbounded Case

HotSpot 2016

Lucca Hirschi

April 3rd, 2016

joint work with

David Baelde
LSV

and

Stéphanie Delaune
LSV
we need formal verification of crypto protocols covering privacy
we need formal verification of crypto protocols covering privacy

Goal:

- checking *unlinkability* and *anonymity*
- in the *symbolic model* (Dolev-Yao)
- for *unbounded sessions*
Introduction

we need formal verification of crypto protocols covering privacy

Goal:

- checking **unlinkability** and anonymity
- in the **symbolic model** (Dolev-Yao)
- for **unbounded sessions**

- **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.

- **Anonymity** [ISO/IEC 15408]:
  
  Ensuring that a user may use a service or resource without disclosing the user’s identity. [...]

[ISO/IEC 15408]
Strong unlinkability [Arapinis, Chothia, Ritter, Ryan CSF’10]:

$$
\not{\nu} T_R \not{\nu} R \not{\nu} T \not{\nu} R \equiv \not{\nu} T_R \not{\nu} S
$$

- $\mathcal{M}$: $\infty$ many different $T - R$ playing $\infty$ many sessions
- $S$: $\infty$ many different $T - R$ playing at most one session
- $\equiv$: observational equivalence (trace equivalence)
Strong unlinkability [Arapinis, Chothia, Ritter, Ryan CSF’10]:

\[
! \nu \overrightarrow{k} (！\nu \overrightarrow{n}(T | R)) \approx ! \nu \overrightarrow{k}.\nu \overrightarrow{n}(T | R)
\]

- \(\mathcal{M}\): \(\infty\) many different \(T - R\) playing \(\infty\) many sessions
- \(\mathcal{S}\): \(\infty\) many different \(T - R\) playing at most one session
- \(\approx\): observational equivalence (trace equivalence)

Checking this is undecidable (because of replication)

Existing approaches:
- **manual**: need to exhibit huge bisimulations
- **automatic** (ProVerif/Maude-NPA/Tamarin): rely on abstraction (diff-equivalence) not precise enough
  \(\Rightarrow\) always fail to prove unlinkability
Context

Strong unlinkability [Arapinis, Chothia, Ritter, Ryan CSF’10]:

\[ \nu \vec{k} (\nu \vec{n} (T | R)) \approx \nu \vec{k} . \nu \vec{n} (T | R) \]

- \( \mathcal{M} \): \( \infty \) many different \( T - R \) playing \( \infty \) many sessions
- \( S \): \( \infty \) many different \( T - R \) playing at most one session
- \( \approx \): observational equivalence (trace equivalence)

- Checking this is undecidable (because of replication)

Existing approaches:

- manual: need to exhibit huge bisimulations
- automatic (ProVerif/Maude-NPA/Tamarin): rely on abstraction (diff-equivalence) not precise enough
  \( \Rightarrow \) always fail to prove unlinkability

\( \Rightarrow \) there is a need for dedicated abstraction targeting unlinkability
Contribution

We identify:
- 2 conditions implying unlinkability and anonymity
- for a class of 2-agents protocols including our target case studies

We make sure:
- our conditions can be checked automatically using ProVerif
- they correspond to good design practices

(sound approach to check automatically privacy properties working well in practice)
I : What could go wrong 🙄?
R1: Messing up with messages

For some malicious behavior, relations over messages leak information about involved agents.

Main idea to avoid that:
- Outputs are (statically) indistinguishable from nonces
  \[ \text{Condition 1: Frame Opacity (FO)} \]

Diagram:
- Tag with \( k, id \)
- Reader with \( k \)
- \( \text{enc}(id, k) \)
R1: Messing up with messages

Practical examples (RFID protocols): HB⁺, DM, KCL, LBV, LD, …
R1: Messing up with messages

Problem
For some malicious behavior, relations over messages leak info about involved agents.

Main idea to avoid that:
- outputs are (statically) indistinguishable from nonces

~ Condition 1: Frame Opacity (FO)
R2: Messing up with conditionals

```
new n_T
y ← enc(n_T, k)
```

```
if y = enc(Y, k)
then x ← enc(⟨Y, n_R⟩, k)
else x ← fail
```

Practical examples: BAC (ePassport), some versions of PACE (new version of ePassport), LAK, CH
R2: Messing up with conditionals

Problem

For some malicious behavior, outcome of conditionals leak info about involved agents

Main idea to avoid that:

- conditional true $\iff$ attacker did not interfere

$\Rightarrow$ Condition 2: Well-Authentication (WA)
II : Big picture
<table>
<thead>
<tr>
<th>Equivalence?</th>
<th>Active Attacker?</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK/ANO</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Equivalence? □✓ □✓  Active Attacker? □✓

⇑  Theorem: implies  ⇑

⇑ FO: automatic check of diff-equivalence using Proverif  ⇑
⇑ WA: automatic check of correspondence prop. using Proverif  ⇑

⇑ Tight enough to conclude on our case studies: (BAC, LAK, Hash-Lock, EKE, SPKE)  ⇑

⇑ FO  ⇑
“Messages are without relations”

⇑ WA  ⇑
“Conditionals hold only for honest interactions”
<table>
<thead>
<tr>
<th>Location</th>
<th>Equivalence?</th>
<th>Active Attacker?</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK/ANO</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FO</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>WA</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Theorem: implies
<table>
<thead>
<tr>
<th>UK/ANO</th>
<th>Equivalence?</th>
<th>Active Attacker?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

\[
\uparrow \text{ Theorem: implies } \uparrow
\]

<table>
<thead>
<tr>
<th>FO</th>
<th>Equivalence?</th>
<th>Active Attacker?</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WA</th>
<th>Equivalence?</th>
<th>Active Attacker?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

\[
\uparrow \text{ can be checked } \uparrow
\]

- FO: automatic check of diff-equivalence using Proverif
- WA: automatic check of correspondence prop. using Proverif
Theorem: implies

can be checked

- FO: automatic check of diff-equivalence using Proverif
- WA: automatic check of correspondence prop. using Proverif

Tight enough to conclude on our case studies:
(BAC, LAK, Hash-Lock, EKE, SPKE)
III : Model and Problem
Applied-$\pi$ - Terms

Any $\Sigma$-algebra + equational theory $E$ + reduction rules (à la Proverif)

Example

- $\Sigma_c = \{\text{dh}/2, \langle_,_\rangle/2, \text{enc}/2, \text{ok}/0, \text{no}/0\}$
- $\Sigma_d = \{\pi_1/1, \pi_2/1, \text{dec}/2\}$
- $E = \{(\text{dh}(\text{dh}(x, y), z) = \text{dh}(\text{dh}(x, z), y))\}$
- $\text{def}_\Sigma(\text{dec}) = \{\text{dec}(\text{enc}(x, y), y) \rightarrow x\}$
- $\text{def}_\Sigma(\pi_i) = \{\pi_i(\langle x_1, x_2 \rangle) \rightarrow x_i\}$

induce

- a congruence $=_E$
  - e.g., $g^{xyz} =_E g^{zyx}$
- a “computation” relation $\downarrow$
  - e.g., $\text{dec}(\text{enc}(n, g^{ab}), g^{ba}) \downarrow n$
Applied-\(\pi\) - Terms

Any \(\Sigma\)-algebra + equational theory \(E\) + reduction rules (à la Proverif)

Example

\[\Sigma_c = \{\text{dh}/2, \langle \_, \_ \rangle/2, \text{enc}/2, \text{ok}/0, \text{no}/0\}\]
\[\Sigma_d = \{\pi_1/1, \pi_2/1, \text{dec}/2\}\]
\[E = \{(\text{dh}(\text{dh}(x, y), z) = \text{dh}(\text{dh}(x, z), y))\}\]
\[\text{def}_\Sigma(\text{dec}) = \{\text{dec}(\text{enc}(x, y), y) \rightarrow x\}\]
\[\text{def}_\Sigma(\pi_i) = \{\pi_i(\langle x_1, x_2 \rangle) \rightarrow x_i\}\]

\[\text{induce}\]

\[\text{a congruence} =_E\]
\[\text{a “computation” relation} \downarrow\]

\[\text{e.g., } g^{xyz} =_E g^{zyx}\]
\[\text{e.g., } \text{dec}(\text{enc}(n, g^{ab}), g^{ba}) \downarrow n\]

\(\leadarrow\) We deal with an arbitrary theory.
Applied-$\pi$ - Syntax

Process:

$$P, Q ::= \begin{array}{ll}
0 & \text{null} \\
\text{in}(c, x).P & \text{input} \\
\text{out}(c, u).P & \text{output} \\
\text{if Test then } P \text{ else } Q & \text{conditional} \\
P \mid Q & \text{parallel} \\
!P & \text{replication} \\
\nu n.P & \text{restriction}
\end{array}$$
Applied-$\pi$ - Syntax

- **Process:**

$$P, Q \ := \ \begin{cases} 0 & \text{null} \\ \in(c, x).P & \text{input} \\ \out(c, u).P & \text{output} \\ \text{if Test then } P \ \text{else } Q & \text{conditional} \\ P \ | \ Q & \text{parallel} \\ \!P & \text{replication} \\ \nu n.P & \text{restriction} \end{cases}$$

- **Frame** ($\phi$): the set of messages revealed to the network
  
  intuition: intruder’s knowledge

$$\phi = \{ w_1 \mapsto \text{enc}(m, k); w_2 \mapsto k \}$$

- **Frame** ($\phi$): the set of messages revealed to the network
  
  intuition: intruder’s knowledge

$$\phi = \{ w_1 \mapsto \text{enc}(m, k); w_2 \mapsto k \}$$
Applied-$\pi$ - Syntax

- **Process:**

  \[
  P, Q := \begin{cases} 
  0 & \text{null} \\
  \text{in}(c, x).P & \text{input} \\
  \text{out}(c, u).P & \text{output} \\
  \text{if Test then } P \text{ else } Q & \text{conditional} \\
  P \mid Q & \text{parallel} \\
  \nu n.P & \text{restriction} \\
  \end{cases}
  \]

- **Frame** ($\phi$): the set of messages revealed to the network
  
  $\mapsto$ intuition: intruder’s knowledge

  \[
  \phi = \{ \begin{array}{l}
  w_1 \mapsto \text{enc}(m, k); \\
  w_2 \mapsto k
  \end{array} \}
  \]

- **Configuration:** $A = (\mathcal{P}; \phi)$
Applied-$\pi$ - Semantics

- **Recipes**: are terms built using handles

  \[ R = \text{dec}(w_1, w_2) \]

  for \( \phi = \{ w_1 \mapsto \text{enc}(m, k); w_2 \mapsto k \} \)

  \( R\phi \downarrow m \)

  \( \leadsto \) intuition: *how* the environment builds messages from its knowledge
Applied-$\pi$ - Semantics

- **Recipes**: are terms built using handles

  \[ R = \text{dec}(w_1, w_2) \]
  \[ R\phi \Downarrow m \]
  \[ \text{for } \phi = \{ w_1 \mapsto \text{enc}(m, k); w_2 \mapsto k \} \]

  \[ \rightsquigarrow \text{intuition: how the environment builds messages from its knowledge} \]

- **Semantics** of configurations:

  \[
  \begin{align*}
  (\text{in}(c, x).P \cup \mathcal{P}; \phi) & \xrightarrow{\text{in}(c,R)} (P\{x \mapsto u\} \cup \mathcal{P}; \phi) & \text{if } R\phi \Downarrow u \\
  (\text{out}(c, u).P \cup \mathcal{P}; \phi) & \xrightarrow{\text{out}(c,w)} (P \cup \mathcal{P}; \phi \cup \{w \mapsto u\}) & \text{if } w \text{ fresh}
  \end{align*}
  \]
Applied-$\pi$ - Semantics

- **Recipes**: are terms built using handles

  $$R = \text{dec}(w_1, w_2)$$

  for $\phi = \{w_1 \mapsto \text{enc}(m, k); w_2 \mapsto k\}$

  $R_\phi \downarrow m$

  → intuition: how the environment builds messages from its knowledge

- **Semantics** of configurations:

  $$(\text{in}(c, x).P \cup \mathcal{P}; \phi) \xrightarrow{\text{in}(c,R)} (P\{x \mapsto u\} \cup \mathcal{P}; \phi) \quad \text{if } R_\phi \downarrow u$$

  $$(\text{out}(c, u).P \cup \mathcal{P}; \phi) \xrightarrow{\text{out}(c,w)} (P \cup \mathcal{P}; \phi \cup \{w \mapsto u\}) \quad \text{if } w \text{ fresh}$$

  + expected rules for conditional and other constructs
Static Equivalence (intuitively)
\[ \phi \sim \psi \text{ when } \]
\[ \quad \text{dom}(\phi) = \text{dom}(\psi) \text{ and } \]
\[ \quad \text{for all tests, it holds on } \phi \iff \text{it holds on } \psi \]

Trace Equivalence
\[ A \sqsubseteq B \text{ when, for any } A \xrightarrow{\text{tr}} A' \text{ there exists } B \xrightarrow{\text{tr}} B' \text{ such that } \]
\[ \phi(A') \sim \phi(B') . \]

\[ A \approx B, \text{ when } A \sqsubseteq B \text{ and } B \sqsubseteq A. \]
### Our class of protocols & our problem

#### Our class

- Intuitively, a party $P$ is a process of the form:

$$
P ::= 0 \mid \text{in}(c, y). \text{if } \text{Test} \then \text{out}(c, u). P_R \else P_{\text{else}}
$$

$$
P_{\text{else}} ::= 0 \mid \text{out}(c', u')
$$
Our class of protocols & our problem

Our class

- Intuitively, a party $P$ is a process of the form:

$$P ::= 0 \mid \text{in}(c, y). \text{if Test then out}(c, u). P_R \text{ else } P_{\text{else}}$$

$$P_{\text{else}} ::= 0 \mid \text{out}(c', u')$$

- A protocol $\Pi$ is a tuple $(\vec{k}, \vec{n}_T, \vec{n}_R, T, R)$ where:
  - $T$ and $R$ are parties
  - $\vec{k}$: identity names and $\vec{n}_T/\vec{n}_R$: session names
  - $fn(T) \subseteq \vec{k} \cup \vec{n}_T$ (resp. for $R$)
Our class of protocols & our problem

Our class

- Intuitively, a party $P$ is a process of the form:

  $$
P ::= 0 \mid \text{in}(c, y). \text{if Test then out}(c, u). P_R \text{ else } P_{\text{else}}
  
P_{\text{else}} ::= 0 \mid \text{out}(c', u')
  $$

- A protocol $\Pi$ is a tuple $(\vec{k}, \vec{n}_T, \vec{n}_R, T, R)$ where:
  - $T$ and $R$ are parties
  - $\vec{k}$: identity names and $\vec{n}_T/\vec{n}_R$: session names
  - $\text{fn}(T) \subseteq \vec{k} \cup \vec{n}_T$ (resp. for $R$)

Unlinkability

$$
\begin{align*}
! \nu \vec{k} (\! (\nu \vec{n}_T T \mid \nu \vec{n}_R R) \! ) & \approx \\
\mathcal{M} & \equiv \\
! \nu \vec{k} . (\nu \vec{n}_T T \mid \nu \vec{n}_R R) & \equiv \\
\mathcal{S} & \equiv
\end{align*}
$$
IV : Sufficient conditions
For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^\text{nonce}$.
For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^{\text{nonce}}$.

Require that all outputs are $\sim$ from nonces is too strong:

- $\Phi = \{ w \mapsto \langle \text{enc}(n_1, k), \text{enc}(n_2, k) \rangle \}$
- If $[\Phi]^{\text{nonce}} = \{ w \mapsto n \}$ then $\Phi \not\sim [\Phi]^{\text{nonce}}$
- If $[\Phi]^{\text{nonce}} = \{ w \mapsto \langle n, n' \rangle \}$ then $\Phi \sim [\Phi]^{\text{nonce}}$
Frame opacity

For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^{\text{nonce}}$.

Require that all outputs are $\sim$ from nonces is too strong:

- $\Phi = \{ w \mapsto \langle \text{enc}(n_1, k), \text{enc}(n_2, k) \rangle \}$
- if $[\Phi]^{\text{nonce}} = \{ w \mapsto n \}$ then $\Phi \not\sim [\Phi]^{\text{nonce}}$
- if $[\Phi]^{\text{nonce}} = \{ w \mapsto \langle n, n' \rangle \}$ then $\Phi \sim [\Phi]^{\text{nonce}}$

Transparent function symbols

$f \in \Sigma_c$ is transparent if:

- attacker can extract its arguments and
- does not appear in E.
For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^{\text{nonce}}$.

- $\Phi = \{ w \mapsto \langle \text{enc}(n_1, k), \text{enc}(n_2, k) \rangle \}$
- $[\Phi]^{\text{nonce}} = \{ w \mapsto \langle n, n' \rangle \}$ and $\Phi \sim [\Phi]^{\text{nonce}}$

**Transparent function symbols**

$f \in \Sigma_c$ is **transparent** if:

- attacker can extract its arguments and
- does not appear in $E$.

**Idealization**

There exists a function $[\cdot]^{\text{ideal}} : \mathcal{T}(\Sigma_c, \mathcal{N}) \rightarrow \mathcal{T}(\Sigma_t, \{\Box\})$ such that:

- $[u]^{\text{ideal}} = f([u_1]^{\text{ideal}}, \ldots)$ if $u =_E f(u_1, \ldots)$ for some $f \in \Sigma_t$,
- and $[u]^{\text{ideal}} = \Box$ otherwise.
We assume additional annotations to actions:

\[ \text{e.g., } \{ T\{ \vec{k} \mapsto \vec{k}_0; \vec{n}_T \mapsto \vec{n}_0 \}; \phi \} \xrightarrow{\text{in}(c,x)[T(\vec{k}_0, \vec{n}_0)].\text{then}[T(\vec{k}_0, \vec{n}_0)]} . \]
We assume additional annotations to actions:

\[ \{ \{ T(\vec{k} \mapsto \vec{k}_0; \vec{n}_T \mapsto \vec{n}_0) \}; \phi \} \xrightarrow{\text{in}(c,x)[T(\vec{k}_0, \vec{n}_0)].\text{then}[T(\vec{k}_0, \vec{n}_0)]} \]

**Well-Authentication**

\( \Pi = (\vec{k}, \vec{n}_T, \vec{n}_R, T, R) \) is *well-authenticating* if, for any execution

\[ (\mathcal{M}; \emptyset) \xrightarrow{t.\text{then}[T(\vec{k}, \vec{n}_1)]]} (\mathcal{P}; \Phi) \]

there must be a \( R(\vec{k}, \vec{n}_2) \) such that \( T(\vec{k}, \vec{n}_1) \) and \( R(\vec{k}, \vec{n}_2) \) were having an **honest execution** in \( (t, \Phi) \).

**A trace** \( t \) is **honest** for a frame \( \Phi \) if

- \( \text{else} \notin t \) and
- \( \text{obs}(t) = \text{out}(\cdot, w_0).\text{in}(\cdot, M_0).\text{out}(\cdot, w_1) \ldots \) with \( M_i \Phi \Downarrow= E w_i \Phi \).
Main Theorem

If $\Pi = (\vec{k}, \vec{n}_T, \vec{n}_R, T, R)$ is well-authenticating and $M$ ensures frame opacity, then $\Pi$ ensures unlinkability.

A similar theorem for both unlinkability and anonymity.
V : Applications
**Tool: UKano**

We wrote UKano: a tool built on top of ProVerif that **automatically checks** our two sufficient conditions.
Tool: UKano

We wrote UKano: a tool built on top of ProVerif that automatically checks our two sufficient conditions.

New proofs of Unlinkability & Anonymity for:

- Feldhofer, Hash-Lock and (fixed) LAK (RFID auth.);
- BAC+PA+AA, (fixed) PACE (ePassport);
Tool: UKano

We wrote UKano: a tool built on top of ProVerif that automatically checks our two sufficient conditions.

New proofs of Unlinkability & Anonymity for:

- Feldhofer, Hash-Lock and (fixed) LAK (RFID auth.);
- BAC+PA+AA, (fixed) PACE (ePassport);

When conditions fail to hold: no direct attacks but still...

Flaws/attacks discovered:

- PACE (¬ UK);
- LAK (¬ UK).

Paper, sources of UKano, ProVerif files at
http://projects.lsv.ens-cachan.fr/ukano/
VI : Conclusion
Equivalence?  Active Attacker?

UK/ANO  ✓  ✓

↑ Theorem: implies ↑

FO  “Messages are without relations”

WA  “Conditionals hold only for honest interactions”

↑ can be checked ↑

- FO: automatic check of diff-equivalence using Proverif
- WA: automatic check of correspondence prop. using Proverif

Tight enough to conclude on our case studies:
(BAC, LAK, Hash-Lock, EKE, SPKE)
Future Work

Improve the method

- tackle memory (often used in RFID)
- move to other tools as backends (Tamarin, Maude-NPA)
- allow more flexibility for idealization
Future Work

**Improve the method**
- tackle memory (often used in RFID)
- move to other tools as backends (Tamarin, Maude-NPA)
- allow more flexibility for idealization

**Reusing core ideas**
- exploit our conditions to obtain other properties
- extract guidelines from our conditions

Paper, sources of UKano, ProVerif files at http://projects.lsv.ens-cachan.fr/ukano/

Thank you!