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

Security & Privacy 2016

Lucca Hirschi, David Baelde and Stéphanie Delaune

Security & Privacy 2016
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 model)
- for unbounded sessions and users
we need formal verification of crypto protocols covering privacy

Goal:
- checking unlinkability and anonymity
- in the symbolic model (= Dolev-Yao model)
- for unbounded sessions and users

unlinkability (untraceability) [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.
Symbolic Model

Symbolic attacker ( giocardino ) controls all the network:
Symbolic Model

Symbolic attacker (ⓐ) controls all the network:

- eavesdrops messages

$\mathbf{\{n\}_k}$: symmetric encryption

Alice $\mathbf{\{n\}_k}$ Bob
Symbolic Model

Symbolic attacker ( злое лицо ) controls all the network:

- eavesdrops messages

\[ \{n\}_k: \text{symmetric encryption} \]

Alice \[\{n\}_k\] Bob
Symbolic Model

Symbolic attacker ( giocattolo ) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives

\[
\begin{align*}
\text{Symbolic attacker ( giocattolo )} & \quad \text{controls all the network:} \\
\Rightarrow & \quad \text{eavesdrops messages} \\
\Rightarrow & \quad \text{builds new messages, applies crypto primitives} \\
\end{align*}
\]
Symbolic Model

Symbolic attacker ( Wolff) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

\[ \text{Alice} \rightarrow_{\{n\}^k} \text{Bob} \]
Symbolic Model

Symbolic attacker (😈) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages
Symbolic Model

Symbolic attacker (😈) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But 😈 cannot break crypto primitives.
Symbolic Model

Symbolic attacker (😈) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But 😈 cannot break crypto primitives.

Symbolic model, pros & cons:

- less precise than computational model
- allows for automation
Symbolic Model

Symbolic attacker (😈) controls all the network:

▶ eavesdrops messages
▶ builds new messages, applies crypto primitives
▶ injects messages

But 😈 cannot break crypto primitives.

Symbolic model, pros & cons:

⊖ less precise than computational model
⊕ allows for automation

Ingredients for modeling:

▶ messages: term algebra with equational theory
Symbolic Model

Symbolic attacker \(\symbol{\mathcal{A}}\) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But \(\symbol{\mathcal{A}}\) cannot break crypto primitives.

Symbolic model, pros & cons:

- less precise than computational model
- allows for automation

Ingredients for modeling:

- messages: term algebra with equational theory
- protocols & attacker: process algebra (e.g., applied \(\pi\)-calculus)
Symbolic Model

Symbolic attacker ( злоупотребление) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But злоупотребление cannot break crypto primitives.

Symbolic model, pros & cons:
- ⊖ less precise than computational model
- ⊕ allows for automation

Ingredients for modeling:
- messages: term algebra with equational theory
- protocols & attacker: process algebra (e.g., applied π-calculus)
- security properties: reachability & observational equivalence
I : Problem
Unlinkability

Scenario 1 | Scenario 2

"Real" usage of the protocol | "Ideal" usage of the protocol

∀ 🐉, 🐉 cannot observe any difference
Unlinkability

Scenario 1

"Real" usage of the protocol

≈

"Ideal" usage of the protocol

≈: trace equivalence

(observational equivalence between processes)
Unlinkability

Scenario 1

Session 1
Session 2
Session 3

Scenario 2

Session 1
Session 1
Session 1

▶ Infinitely many users
▶ Each playing infinitely many sessions
Unlinkability

Scenario 1

id₁
Session 1
Session 2
Session 3

id₂

id₃

Session 1
Session 2
Session 3

Scenario 2

id₁

id₂

id₃

Session 1
Session 1
Session 1

!ν id !ν Sess. P ≈ !ν id.ν Sess. P

∞ users

∞ sessions

∞ users

(Strong unlinkability [Arapinis, Chothia, Ritter, Ryan CSF’10])
Goal

- automatic verification of

\[ !\, \nu \text{id.} (\!\nu \text{Sess.}P) \approx !\, \nu \text{id.} (\nu \text{Sess.}P) \]

for a large class of 2-party protocols (think of \( P = \text{Tag} \mid \text{Reader} \))
Goal

- automatic verification of

\[ !\, \nu\text{id.}(!\, \nu \text{Sess.}P) \approx !\, \nu\text{id.}(\nu \text{Sess.}P) \]

for a large class of 2-party protocols (think of $P = \text{Tag} \mid \text{Reader}$)

Existing approaches:

- manual: long, difficult, and highly error prone
- automatic (only ProVerif/Maude-NPA/Tamarin):
  - rely on too imprecise approximation of $\approx$
  - $\not\implies$ always fail to prove unlinkability
Contributions

Theory:
- 2 reasonable conditions implying unlinkability (& anonymity)
- for a large class of 2-party protocols
Contributions

Theory:
- 2 reasonable conditions implying unlinkability (& anonymity)
- for a large class of 2-party protocols

Practice:
- our conditions can be checked automatically using existing tools
- we provide tool support for that (UKano)
Contributions

Theory:
- 2 reasonable conditions implying unlinkability (& anonymity)
- for a large class of 2-party protocols

Practice:
- our conditions can be checked automatically using existing tools
- we provide tool support for that (UKano)

Applications:
- new proofs & attacks on RFID protocols
II: Two Generic Classes of Attacks
Two Conditions to Avoid them
1st Class: Leaks through Relations over Messages

Tag

\[ k, \text{id} \]

Reader

\[ \{\text{id}\}_k \]

\[ \cdots \]
1st Class: Leaks through Relations over Messages

\[ \text{Tag}_1 \]
\[ k_1, \text{id}_1 \]
\[ \{\text{id}_1\} \]
\[ k_1 \]

\[ \text{Tag}_2 \]
\[ k_2, \text{id}_2 \]
\[ \{\text{id}_2\} \]
\[ k_2 \]
1st Class: Leaks through Relations over Messages

\[
\tag{k_1, \text{id}_1} = \tag{k_2, \text{id}_2}
\]
$1^{st}$ Class: Leaks through Relations over Messages

\[
\begin{align*}
\text{Tag}_1 \quad & k_1, \text{id}_1 \\
\{\text{id}_1\}_{k_1} \quad & (k_1, \text{id}_1) \neq (k_2, \text{id}_2) \\
\text{Tag}_2 \quad & k_2, \text{id}_2 \\
\{\text{id}_2\}_{k_2} \quad & \equiv 
\end{align*}
\]
1st Class: Leaks through Relations over Messages

![Diagram showing Tag and Reader with relation and encryption]

Tag: $k, id$

Reader: $k$

Relation: $\{n, id\}_k$

Encryption: $n$
1st Class: Leaks through Relations over Messages

Tag\(_1\) \[ k_1, id_1 \]

\[ \{0, id_1 \}_{k_1} \]

Tag\(_2\) \[ k_2, id_2 \]

\[ \{0, id_2 \}_{k_2} \]
Problem

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

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

Main idea to avoid that:

- outputs are indistinguishable from fresh nonces

\[ \langle \text{error}; \{u\}_k \rangle \rightarrow \langle \text{error}; n \rangle \]

\[ \rightsquigarrow 1^{\text{st}} \text{ Condition: Frame Opacity (FO)} \]

... formal definition in the paper
2nd Class: Leaks through Conditionals’ Outcomes

\[
\text{Tag} \quad k \quad \{n\}_k \quad \text{Reader} \quad k
\]

\[
\text{if } \text{dec}(X, k) \neq \bot
\]

\[
\{n'\}_k
\]
2\textsuperscript{nd} Class: Leaks through Conditionals’ Outcomes

\[
\text{Tag}_1 \xrightarrow{k_1} \text{Reader}_1 \xleftarrow{n_1} \xrightarrow{{k_1}}
\]
2nd Class: Leaks through Conditionals’ Outcomes

Tag\(_1\) \(k_1\)

\{n_1\}_{k_1}

Reader\(_1\) \(k_1\)

Tag\(_2\) \(k_2\)

\{n_2\}_{k_2}

Reader\(_2\) \(k_2\)
2\textsuperscript{nd} Class: Leaks through Conditionals’ Outcomes

\[ \text{Tag}_1 \quad \text{Reader}_1 \]

\[ k_1 \quad \downarrow \quad \{ n_1 \}_{k_1} \]

\[ \text{Tag}_2 \quad \text{Reader}_2 \]

\[ k_2 \quad \downarrow \quad \{ n_2 \}_{k_2} \]

\[ \{ n_1 \}_{k_1} \]

\[ \text{if } \text{dec}(X, k_2) \neq \bot \]

\[ ? \]
2nd Class: Leaks through Conditionals’ Outcomes

Tag_1

\[ \{n_1\}_{k_1} \]

Reader_1

\[ \{n_1\}_{k_1} \]

Tag_2

\[ \{n_2\}_{k_2} \]

Reader_2

\[ \{n_1\}_{k_1} \]

\[ \text{if } \text{dec}(X, k_2) \neq \perp \]

\[ \{n'\}_{k_2} \quad k_1 = k_2 \]
2nd Class: Leaks through Conditionals’ Outcomes

\[ \text{Tag}_1 \quad \rightarrow \quad \{ n_1 \}_{k_1} \quad \rightarrow \quad \text{Reader}_1 \]

\[ \text{Tag}_2 \quad \rightarrow \quad \{ n_2 \}_{k_2} \quad \rightarrow \quad \text{Reader}_2 \]

\[ \text{if } \text{dec}(X, k_2) \neq \bot \]

\[ k_1 \neq k_2 \]
2nd Class: Leaks through Conditionals’ Outcomes

Problem
For some malicious behavior, conditionals’ outcomes leak info about involved agents.

Main idea to avoid that:

- conditional evaluates positively $\iff$ attacker did not interfer

$\sim 2^{nd}$ Condition: Well-Authentication (WA)

... formal definition in the paper
Main Result

Theorem

For any protocol in our class:

\[
\begin{align*}
frame \ opacity \ & \ \& \\
& \wedge \\
well\text{-}authentication
\end{align*}
\]

\[\Rightarrow\]

\[
\begin{align*}
Unlinkability \ & \ \& \\
& \wedge \\
Anonymity
\end{align*}
\]

... formal statement and proof in the paper
III : Mechanization & Applications
Both conditions can be automatically verified using ProVerif:

- **Frame Opacity:** $\rightsquigarrow$ equivalence between messages
- **Well Authentication:** $\rightsquigarrow$ just reachability properties
Mechanization

Both conditions can be automatically verified using ProVerif:

- **Frame Opacity:** $\leadsto$ equivalence between messages
- **Well Authentication:** $\leadsto$ just reachability properties

**Tool: UKano**

Built on top of ProVerif that **automatically checks** our conditions.
## Case Studies

<table>
<thead>
<tr>
<th>RFID auth. protocol</th>
<th>Frame opacity</th>
<th>Well-auth.</th>
<th>Unlinkability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feldhofer</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>Hash-Lock</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>LAK (stateless)</td>
<td>–</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Fixed LAK</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ePassport protocol</th>
<th>Frame opacity</th>
<th>Well-auth.</th>
<th>Unlinkability</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAC</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>BAC/PA/AA</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>PACE (faillible dec)</td>
<td>–</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>PACE (missing test)</td>
<td>–</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>PACE</td>
<td>–</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>PACE with tags</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
</tbody>
</table>

- Found automatically new proofs and new attacks using UKano
IV : Conclusion
Conclusion

Contributions

- **Theory**: 2 conditions $\Rightarrow$ **unlinkability** & anonymity
- **Practice**: **UKano** automatically verifies them
- **Applications**: **new proofs** & **attacks** on RFID protocols
Conclusion

Contributions

- **Theory**: 2 conditions $\Rightarrow$ unlinkability & anonymity
- **Practice**: UKano automatically verifies them
- **Applications**: new proofs & attacks on RFID protocols

Future Work

- Improve the method (class of protocols, other back-end)
- Seek other types of protocols (e.g., e-Voting)

More details, sources of UKano, ProVerif files at

http://projects.lsv.ens-cachan.fr/ukano/