Symbolic verification of security protocols
Modelling and verifying unlinkability

Stéphanie DELAUNE, Univ Rennes, CNRS, IRISA
GT Méthodes Formelles pour la Sécurité
March 29th, 2023

joint work with D. Baelde, A. Debant, L. Hirschi, and S. Moreau
Cryptographic protocols everywhere!

- small programs designed to secure communication (*e.g.* secrecy, authentication, anonymity, ...)
- use cryptographic primitives (*e.g.* encryption, signature, .......)
Cryptographic protocols everywhere!

- small programs designed to secure communication (*e.g.* secrecy, authentication, anonymity, ...)
- use **cryptographic primitives** (*e.g.* encryption, signature, ....)

It becomes more and more important to protect our privacy.
An e-passport is a passport with an RFID tag embedded in it.

The RFID tag stores:

- the information printed on your passport;
- a JPEG copy of your picture;
- ...
Electronic passport

An e-passport is a passport with an **RFID tag** embedded in it.

The **RFID tag** stores:
- the information printed on your passport;
- a JPEG copy of your picture;
- ... 

The Basic Access Control (BAC) protocol is a key establishment protocol that has been designed to **protect our personal data**, and to ensure **unlinkability**.

**Unlinkability** aims to ensure that a user may make multiple uses of a service or resource without others being able to link these uses together. [ISO/IEC standard 15408]
How cryptographic protocols can be attacked?

Logical attacks
- can be mounted even assuming perfect cryptography, e.g. replay attack, man-in-the-middle attack, ...
- subtle and hard to detect by "eyeballing" the protocol

Example: A traceability attack on the BAC protocol
How cryptographic protocols can be attacked?

Logical attacks

- can be mounted even assuming perfect cryptography, *e.g.* replay attack, man-in-the-middle attack, . . .
- subtle and hard to detect by “eyeballing” the protocol
Logical attacks

- can be mounted even assuming perfect cryptography, e.g. replay attack, man-in-the middle attack, ... 
- subtle and hard to detect by “eyeballing” the protocol

Example: A traceability attack on the BAC protocol

Privacy issue

Security

Defects in e-passports allow real-time tracking
This threat brought to you by RFID

The register - Jan. 2010
How to verify the absence of logical flaws?

- dissect the protocol and test their resilience against well-known attacks;
  → this is not sufficient!
How to verify the absence of logical flaws?

- dissect the protocol and test their resilience against well-known attacks; → this is not sufficient!

- perform a manual security analysis → this is error-prone!
How to verify the absence of logical flaws?

- dissect the protocol and test their resilience against well-known attacks;  
  → this is not sufficient!

- perform a manual security analysis  
  → this is error-prone!

Our approach

formal symbolic verification using automatic/interactive tools
Formal (symbolic) verification in a nutshell

Does the protocol satisfy a security property?

Modelling

Two main tasks:

1. Modelling: protocols, security properties, and the attacker;
2. Verifying: designing verification algorithms and tools.

→ this talk: a focus on unlinkability
Some success stories (mostly related to reachability properties)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Authors</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProVerif</td>
<td>Blanchet, 01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Meier et al., 13</td>
<td></td>
</tr>
<tr>
<td>Sapić+</td>
<td>Cheval et al., 22</td>
<td></td>
</tr>
</tbody>
</table>
Some success stories (mostly related to reachability properties)

**ProVerif**
[Blanchet, 01]

**Sapic**
[Cheval *et al.*, 22]

[Meier *et al.*, 13]

**Verified models** and reference implementations for TLS 1.3
[Blhargavan *et al.*, 17]

A formal security analysis of the EMV Standard using Tamarin (Break, Fix, and Verify)
[Basin *et al.*, 2020]

A comprehensive, formal and automated analysis of the EDHOC protocol
[Jacomme *et al.*, 23]
Some success stories (mostly related to reachability properties)

**ProVerif**

[Blanchet, 01]  [Meier et al., 13]  [Cheval et al., 22]

Verified models and reference implementations for TLS 1.3  
[Bhargavan et al., 17]

A formal security analysis of the EMV Standard using Tamarin (Break, Fix, and Verify)  
[Basin et al., 2020]

A comprehensive, formal and automated analysis of the EDHOC protocol  
[Jacomme et al, 23]
What about unlinkability (in the symbolic setting)?

Actually, existing tools like ProVerif and Tamarin are not suitable to analyse unlinkability, and therefore few formal proofs exist in the unbounded setting.

- [Chatzikokolakis et al., 2010]: sufficient conditions checkable using ProVerif that allows one to establish unlinkability for a simple class of protocols (single-step protocols). → their notion of unlinkability is rather weak

- [Arapinis et al., 2010]: a formal definition of unlinkability, and a manual proof of unlinkability for a fixed version of the e-passport protocol. → this result is wrong

- [Bhargavan et al., 2022]: a symbolic analysis of privacy for TLS 1.3 with Encrypted Client Hello → several encodings tricks are used.
Part I

Modelling: protocols, the attacker, and unlinkability
Running example: Basic Hash protocol

- $k$ is a long-term secret key shared between the tag and the reader;
- each tag has its own key $k$. 
Protocols as processes

→ a programming language with constructs for concurrency and communication (applied-pi calculus [Abadi & Fournet, 01])

\[
P, Q := 0 \quad \text{null process}
\]
\[
\quad \mid \text{in}(c, x); P \quad \text{input}
\]
\[
\quad \mid \text{out}(c, M); P \quad \text{output}
\]
\[
\quad \mid \text{new } n; P \quad \text{name generation}
\]
\[
\quad \mid \text{if } M = N \text{ then } P \text{ else } Q \quad \text{conditional}
\]
\[
\quad \mid !P \quad \text{replication}
\]
\[
\quad \mid (P \mid Q) \quad \text{parallel composition}
\]
\[
\quad \mid \ldots
\]
Terms are built over a set of names $\mathcal{N}$ (private), and function symbols $\Sigma$ (public) equipped with an equational theory $E$.

Example:

$$\Sigma = \{senc, sdec\} \text{ with } E = \{sdec(senc(x, y), y) = x\}.$$  

Let $\Phi = \{w_1 \mapsto senc(s, k); w_2 \mapsto k\}$. $R = sdec(w_1, w_2)$ is a recipe to compute $s$. Indeed, we have that $R\Phi \equiv_E s$. 
Going back to Basic Hash

Mesages/Computations as terms

- \( \Sigma = \{h, \langle \rangle, \text{proj}_1, \text{proj}_2\} \);
- \( E = \{\text{proj}_1(\langle x_1, x_2 \rangle) = x_1, \text{proj}_2(\langle x_1, x_2 \rangle) = x_2\} \).

Protocol as a process

- \( T(k) = \text{new } n; \text{out}(c, \langle n, h(n, k) \rangle) \).
- \( R(k) = \text{in}(c, y); \text{if } h(\text{proj}_1(y), k) = \text{proj}_2(y) \)
  \[ \text{then out}(c, \text{ok}) \]
  \[ \text{else out}(c, \text{ko}). \]

Then, the whole system can be written as follows:

\[ !\text{new } k; ( !R(k) \mid !T(k) ) \]
Semantics (some selected rules)

Labelled transition system over configurations:

\[(\mathcal{P}; \Phi)\]

- multiset of processes
- frame = knowledge of the attacker
Semantics (some selected rules)

Labelled transition system over configurations:

\[(\mathcal{P}; \Phi)\]

- multiset of processes
- \textbf{frame} = knowledge of the attacker

\[\text{Out} (\{\text{out}(c, M); P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\text{out}(c, w_i)} (\{P\} \uplus \mathcal{P}; \Phi \uplus \{w_i \mapsto M\})\]

\[\text{with } i = |\Phi|\]

\[\text{Then} (\{\text{if } M_1 = M_2 \text{ then } P \text{ else } Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{T} (\{P\} \uplus \mathcal{P}; \Phi)\]

\[\text{when } M_1 =_E M_2\]

\[\text{In} (\{\text{in}(c, x); P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\text{in}(c, R)} (\{P\{x \mapsto R\Phi}\} \uplus \mathcal{P}; \Phi)\]

\[\ldots\]
Trace equivalence

Trace equivalence between configurations: \( K \approx_t K' \).
For any execution trace \( K \xrightarrow{\text{tr}} (\mathcal{P}; \Phi) \) there exists an execution \( K' \xrightarrow{\text{tr}} (\mathcal{P}'; \Phi') \) such that \( \Phi \sim_s \Phi' \) (and conversely)

\[
\begin{align*}
\{w_1 \mapsto \left< n_1, h(n_1, k) \right> ; w_2 \mapsto \left< n_2, h(n_2, k) \right> \} & \not\sim_s \\
\{w_1 \mapsto \left< n'_1, h(n'_1, k') \right> ; w_2 \mapsto \left< n'_2, h(n'_2, k') \right> \} & \rightarrow
\end{align*}
\]

\([w_2 \mapsto proj_2(w_2), w_1] = proj_1(w_2)\)
Trace equivalence between configurations: $K \approx_t K'$.
For any execution trace $K \xrightarrow{\text{tr}} (P; \Phi)$ there exists an execution $K' \xrightarrow{\text{tr}} (P'; \Phi')$ such that $\Phi \sim_s \Phi'$ (and conversely).

Static equivalence between frames: $\Phi \sim_s \Phi'$.
Any test that holds in $\Phi$ also holds in $\Phi'$ (and conversely).

Example:
$\{w_1 \mapsto k; w_2 \mapsto \langle n, h(n, k) \rangle\} \not\sim_s \{w_1 \mapsto k; w_2 \mapsto \langle n', h(n', k') \rangle\}$
$\rightarrow$ with the test $h(\text{proj}_1(w_2), w_1) \neq \text{proj}_2(w_2)$. 
Trace equivalence

Trace equivalence between configurations: \( K \cong_t K' \).
For any execution trace \( K \xrightarrow{\text{tr}} (P; \Phi) \) there exists an execution \( K' \xrightarrow{\text{tr}} (P'; \Phi') \) such that \( \Phi \sim_s \Phi' \) (and conversely).

Static equivalence between frames: \( \Phi \sim_s \Phi' \).
Any test that holds in \( \Phi \) also holds in \( \Phi' \) (and conversely).

Example:
\[
\{ w_1 \mapsto k; w_2 \mapsto \langle n, h(n, k) \rangle \} \not\sim_s \{ w_1 \mapsto k; w_2 \mapsto \langle n', h(n', k') \rangle \}
\]
\[
\xrightarrow{\ ? } \text{ with the test } h(\text{proj}_1(w_2), w_1) = \text{proj}_2(w_2).
\]

\[
\{ w_1 \mapsto \langle n_1, h(n_1, k) \rangle; w_2 \mapsto \langle n_2, h(n_2, k) \rangle \}
\sim_s
\{ w_1 \mapsto \langle n'_1, h(n'_1, k) \rangle; w_2 \mapsto \langle n'_2, h(n'_2, k') \rangle \}
\]
Unlinkability aims to ensure that a user may make multiple uses of a service or resource without others being able to link these uses together. [ISO/IEC standard 15408]

→ the real system should be equivalent to the ideal one (from the point of view of the attacker).
Unlinkability aims to ensure that a user may make multiple uses of a service or resource without others being able to link these uses together. [ISO/IEC standard 15408]

→ the real system should be equivalent to the ideal one (from the point of view of the attacker).

many sessions for each identity

only one session for each identity
For single-step protocols, we may consider the following equivalence:

\[ !\text{new } k; !T(k) \approx_t !\text{new } k; T(k) \]

This approach was used in [Chatzikokolakis et al., 2010] to establish unlinkability for BH and OSK protocols.
Modelling unlinkability (1\textsuperscript{th} attempt)

For single-step protocols, we may consider the following equivalence:

\[ ! \text{new } k; \; ! T(k) \approx_t ! \text{new } k; \; T(k) \]

This approach was used in [Chatzikokolakis et al., 2010] to establish unlinkability for BH and OSK protocols.

Example: OSK protocol

- \( h \) and \( g \) are two hash functions;
- \( k \) is updated - with \( h(k) \) - after a successful execution on both sides.
Tags are proved unlinkable in [Chatzikokolakis et al., 2010] but there is an attack!

Keypoint #1: modelling the reader is important.
Modelling unlinkability (2\textsuperscript{th} attempt)

→ definition first proposed by [Arapinis et al., CSF’10] (but for another notion of equivalence)

\[ !\text{new } k; ( ! R(k) \mid ! T(k)) \approx_t !\text{new } k; ( R(k) \mid T(k) ) \]

This definition is:

- suitable to analyse e.g. e-passport protocols, and many other stateless protocols;
- the one we used in our work [Hirschi, Baelde & D., SP’16 & JCS’19].
Going back to Basic Hash protocol (a stateful protocol)

→ linkable according to our previous definition (specific readers).

\[
\langle n, h(n, k) \rangle \quad \text{new } n
\]

\[
\langle n, h(n, k) \rangle \quad \text{ok}
\]

\[
\langle n, h(n, k) \rangle \quad \text{ok}
\]

\[
\langle n, h(n, k) \rangle \quad \text{error}
\]
Basic Hash protocol

→ with a generic reader, no linkability attack.

Keypoint #2: The way the reader is modelled is important.
Modelling unlinkability for stateful protocols (3rd attempt)

→ definition proposed in [Baelde, D., Moreau, CSF’20]

We consider a generic reader having an access to a database $DB$

$$
!R \mid (!\text{new } k; \text{insert } DB(k); !T(k))
$$

$$
\approx_t
$$

$$
!R \mid (!\text{new } k; \text{insert } DB(k); T(k))
$$
Modelling unlinkability for stateful protocols (3rd attempt)

→ definition proposed in [Baelde, D., Moreau, CSF’20]

We consider a generic reader having an access to a database DB

\[ !R \mid (\!\text{new } k; \text{insert } DB(k); !T(k)) \approx_t !R \mid (\!\text{new } k; \text{insert } DB(k); T(k)) \]

Basic Hash Example

- \( R = \text{in}(c, y); \text{get } z_k \in DB \text{ such that } h(\text{proj}_1(y), z_k) = \text{proj}_2(y) \)
  \( \text{in out}(c, \text{ok}) \)
  \( \text{else out}(c, \text{ko}). \)

→ Modelling tables in ProVerif (or Tamarin) is not an issue.
Part II

How can we establishunlinkability?
Exciting tools able to establish trace equivalence

The problem is undecidable in general.
Exsiting tools able to establish trace equivalence

The problem is undecidable in general.

Approach 1: Limiting the number of sessions

- the problem becomes decidable (under some assumptions);
- decision procedures and tools have been developed, e.g. Deepsec, Spec, Sat-Equiv, …
Existing tools able to establish trace equivalence

The problem is undecidable in general.

**Approach 1:** Limiting the number of sessions

- the problem becomes decidable (under some assumptions);
- decision procedures and tools have been developed, e.g. Deepsec, Spec, Sat-Equiv, ...

**Approach 2:** Trying to solve the general case

- **ProVerif:** over-approximations are performed, termination is not guaranteed [Blanchet et al., 2005]
- **Tamarin:** an interactive tool [Basin et al., 2015]

→ they are based on diff-equivalence (too strong)
Diff-equivalence

How does it work (or not)?

- form a bi-process $B$ using the operator $\text{diff}[M_L, M_R]$;
- both sides of the bi-process $B$ have to evolve simultaneously (+ static equivalence) to be declared in diff-equivalence

$\rightarrow$ In such a case, we have that $\text{fst}(B) \approx_t \text{snd}(B)$. 
Diff-equivalence

How does it work (or not)?

- form a bi-process $B$ using the operator $\text{diff}[M_L, M_R]$;
- both sides of the bi-process $B$ have to evolve simulatenously (+ static equivalence) to be declared in diff-equivalence

$\rightarrow$ In such a case, we have that $\text{fst}(B) \approx_t \text{snd}(B)$.

Formally, the semantics is given by a labelled transition system over bi-configurations $(\mathcal{P}; \Phi)$ where messages and computations may contain the $\text{diff}$ operator.

**Example 1:** $\text{out}(a) \; | \; \text{out}(b) \; ? \; \approx \; \text{out}(b) \; | \; \text{out}(a)$

$\rightarrow \; B = \text{out}(\text{diff}[a, b]) \; | \; \text{out}(\text{diff}[b, a])$ (* not in diff-equivalence *)
Diff-equivalence

How does it work (or not)?

- form a bi-process $B$ using the operator $\text{diff}[M_L, M_R]$;
- both sides of the bi-process $B$ have to evolve simulatenously (+ static equivalence) to be declared in diff-equivalence

\[ \rightarrow \] In such a case, we have that $\text{fst}(B) \approx_t \text{snd}(B)$.

Formally, the semantics is given by a labelled transition system over bi-configurations $(\mathcal{P}; \Phi)$ where messages and computations may contain the $\text{diff}$ operator.

Example 2

$B = \text{insert } DB(\text{diff}[a, b]); \text{insert } DB(\text{diff}[b, a])$

get $x$ such that $x = a$ then out$(c, \text{ok})$ else out$(c, \text{ko})$

(* not in diff-equivalence *)
$B = !R \mid (\text{!new } k; \text{!new } kk; \text{insert } DB(\text{diff}[k, kk]); \text{T(} \text{diff}[k, kk] ))$

Let’s consider a scenario with:

- 1 reader;
- 2 tags: $T(\text{diff}[k, kk_1])$,
  and $T(\text{diff}[k, kk_2])$.

<table>
<thead>
<tr>
<th>DB</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
<td>$k$</td>
<td>$kk_1$</td>
</tr>
<tr>
<td>line 2</td>
<td>$k$</td>
<td>$kk_2$</td>
</tr>
</tbody>
</table>
Diff-equivalence does not hold on Basic Hash

\[ B = !R \mid (!\text{new } k; !\text{new } kk; \text{insert DB} \langle \text{diff}[k, kk] \rangle; T(\text{diff}[k, kk])) \]

Let’s consider a scenario with:

- 1 reader;
- 2 tags: \( T(\text{diff}[k, kk_1]) \), and \( T(\text{diff}[k, kk_2]) \).

\[
\begin{array}{|c|c|c|}
\hline
\text{DB} & \text{left} & \text{right} \\
\hline
\text{line 1} & k & kk_1 \\
\hline
\text{line 2} & k & kk_2 \\
\hline
\end{array}
\]

1. The tag outputs \( w_1 = \langle n_1, h(n_1, \text{diff}[k, kk_1]) \rangle \);
2. The reader \( R \) will diverge on this input:

\[
R = \text{in}(c, y); \text{ get } DB(z_k) \text{ st. } \text{eq}(h(\text{proj}_1(y), z_k), \text{proj}_2(y)) \text{ in out}(c, \text{ok}) \text{ else out}(c, \text{ko})
\]
Diff-equivalence does not hold on Basic Hash

\[ B = !R \mid (!\text{new } k; \text{!new } kk; \text{insert } \text{DB}(\text{diff}[k, kk]); \text{T}(\text{diff}[k, kk])) \]

Let’s consider a scenario with:

- 1 reader;
- 2 tags: \( T(\text{diff}[k, kk_1]) \),
  and \( T(\text{diff}[k, kk_2]) \).

<table>
<thead>
<tr>
<th></th>
<th>DB</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
<td>line 1</td>
<td>k</td>
<td>kk_1</td>
</tr>
<tr>
<td>line 2</td>
<td>line 2</td>
<td>k</td>
<td>kk_2</td>
</tr>
</tbody>
</table>

1. The tag outputs \( w_1 = \langle n_1, h(n_1, \text{diff}[k, kk_1]) \rangle \);
2. The reader \( R \) will diverge on this input:

\[ R = \text{in}(c, y); \]
\[ \text{get } \text{DB}(z_k) \text{ st. } \text{eq}(h(\text{proj}_1(y), z_k), \text{proj}_2(y)) \text{ in out}(c, \text{ok}) \text{ else out}(c, \text{ko}) \]

\( \rightarrow \) Proverif returns cannot be proved.
Our result for stateless 2-party protocols


**Theorem**

If a protocol ensures both **well-authentication** and **frame opacity**
then it ensures unlinkability, i.e.:

\[ \neg \text{new } k; (\neg R(k) \mid \neg T(k)) \approx_t \neg \text{new } k; (R(k) \mid T(k)) \]

\[ \rightarrow \] These 2 conditions are easier to check by existing tools
Intuition behind the sufficient conditions

Well-Authentication

- Goal = avoid leaks through outcomes of conditionals.
- "Whenever a conditional is positively evaluated, the agents involved are having so far an honest interaction."

→ This is a reachability property.
Intuition behind the sufficient conditions

Well-Authentication

- Goal = avoid leaks through outcomes of conditionals.
- "Whenever a conditional is positively evaluated, the agents involved are having so far an honest interaction."

→ This is a reachability property.

Frame Opacity

- Goal = avoid leaks through relations over messages.
- "Any reachable frame must be statically equivalent to an idealised frame that only depends on data already observed during the execution."

→ This can be verified with (an extension of) diff-equivalence.
## Summary of our case studies using ProVerif

<table>
<thead>
<tr>
<th>Protocol</th>
<th>WA</th>
<th>FO</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>attack</td>
</tr>
<tr>
<td>Fixed LAK</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<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>attack</td>
</tr>
<tr>
<td>PACE (as in [Bender et al, 09])</td>
<td>✗</td>
<td>✓</td>
<td>attack</td>
</tr>
<tr>
<td>PACE</td>
<td>✗</td>
<td>✓</td>
<td>attack</td>
</tr>
<tr>
<td>PACE with tags</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>DAA sign</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>DAA join</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>abcdh (irma)</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
</tbody>
</table>
Our result for stateful 2-party protocols

[Baelde, D., Moreau, CSF’20]

**Theorem**

If a protocol ensures well-authentication, frame opacity and no desynchronisation then it ensures unlinkability, i.e.:

\[
\neg R \mid (\neg \text{new } k; \text{insert } DB(k); \neg T(k)) \approx_t \neg R \mid (\neg \text{new } k; \text{insert } DB(k); T(k))
\]

**No desynchronisation**

- Goal = avoid leaks through desynchronisations between agents.
- "An honest interaction between a tag and a reader cannot fail."

→ This is also a reachability property! (But a little more tricky...)
### Summary of our case studies using Tamarin

<table>
<thead>
<tr>
<th>Protocol</th>
<th>WA</th>
<th>FO</th>
<th>ND</th>
<th>unlinkability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Hash</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>Hash Lock</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>Feldhofer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>OSK v1</td>
<td>✓</td>
<td>X</td>
<td></td>
<td>attack</td>
</tr>
<tr>
<td>OSK v2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>LAK (pairs)</td>
<td>✓</td>
<td>X</td>
<td></td>
<td>attack</td>
</tr>
<tr>
<td>LAK (pairs, fixed)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>LAK (pairs, no update)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
<tr>
<td>5G-AKA (simplified)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>safe</td>
</tr>
</tbody>
</table>

→ simple conditions in the theory but **not so easily checkable** in practice
A recent result for stateful 2-party protocols

[Baelde, Debant, D., CSF’23]

Main Goal

Transform a ProVerif model $\mathcal{M}$ into another model $\mathcal{M}'$ such that:

- diff-equivalence on $\mathcal{M}' \Rightarrow$ trace equivalence on $\mathcal{M}$; and
- diff-equivalence is verified with ProVerif on $\mathcal{M}'$.

Our transformation:

1. duplicate the get instructions in $\mathcal{M}$ to dissociate the two parts of the bi-process; (possible using the allowDiffPatterns option)
2. add some axioms (proved correct manually) to help ProVerif to reason on our new model.
let T(k) = new nT; out(c,(nT,h(nT,k))).

let R = in(c,y);
get db(k) suchthat snd(y) = h(fst(y),k) in out(c,ok)
else out(c,ko).

process
(! new k; ! new kk; insert db(diff[k,kk]);
 phase 1; T(diff[k,kk]))
| (phase 1; ! R)
Step 1: Basic Hash example

→ duplicate the get instructions to dissociate the two parts of the bi-process.

let R =
    in(c,diff[y1L,y1R]);
    get db(diff[kL,wR]) suchthat snd(y1L) = h(fst(y1L),kL) in
        (get db(diff[wL,kR]) suchthat snd(y1R) = h(fst(y1R),kR) in
            out(c,diff[ok,ok])
        else
            out(c,diff[ok,ko]))
    else
        out(c,diff[ok,ko]))
else
    (get db(diff[wL,kR]) suchthat snd(y1R) = h(fst(y1R),kR) in
        out(c,diff[ko,ok])
    else
        out(cR,diff[ko,k0])).
Step 2: Refining the analysis in the failure branches

We illustrate this on a very simple example.

Before, ... 

\[B = \text{insert } \text{tbl}(\text{ok});\]
\[\text{get } \text{tbl}(x) \text{ st. true in out}(c, \text{ok})\]
\[\text{else out}(c, \text{diff}[\text{ok}_L, \text{ok}_R])\]

... and ProVerif cannot prove equivalence (whereas it holds).
We illustrate this on a very simple example.

After, ...

\[ B = \text{event}(\text{Inserted}(\text{ok})); \text{insert} \; \text{tbl}(\text{ok}); \]

\[ \text{get} \; \text{tbl}(x) \text{ st. true in out}(c, \text{ok}) \]

\[ \text{else event}(\text{Fail}()); \text{out}(c, \text{diff}[\text{ok}_L, \text{ok}_R]) \]

...together with the following axiom:

\[ \text{event}(\text{Fail}()) \land \text{event}(\text{Inserted}(\text{diff}[y^L, y^R]))) \Rightarrow \text{false.} \]

\[ \rightarrow \text{Now, Proverif is able to conclude that equivalence holds.} \]
Step 2: Refining the analysis in the failure branches

We illustrate this on a very simple example.

After, ...

\[
B = \text{event(Insert}(\text{ok})) \land \text{insert } \text{tbl(}\text{ok)}; \text{get } \text{tbl(}x\text{) st. true in } \text{out(}c, \text{ok)}
\]

\[
\text{else event(}\text{Fail(})\land \text{out(}c, \text{diff[}ok_L, \text{ok}_R])\text{)}
\]

...together with the following axiom:

\[
\text{event(}\text{Fail(})\land \text{event(}\text{Inserted(}\text{diff[}y_L, y_R\text{]})\Rightarrow \text{false.}
\]

\[\rightarrow \text{ Now, Proverif is able to conclude that equivalence holds.}\]

Going back to the Basic Hash protocol

\[
\text{event(}\text{FailL}(x^L)) \land \text{event(}\text{Inserted(}\text{diff[}y_L, y_R\text{]})\Rightarrow \text{proj}_2(x^L) \neq h(\text{proj}_1(x^L), y_L)
\]

\[
\text{event(}\text{FailR}(x^R)) \land \text{event(}\text{Inserted(}\text{diff[}y_L, y_R\text{]})\Rightarrow \text{proj}_2(x^R) \neq h(\text{proj}_1(x^R), y_R)
\]
Case studies

Implementation
The two steps of the transformation have been implemented ($\approx 2k$ Ocaml LoC).

Case studies
Basic Hash, Hash-Lock, Feldhofer, a variant of LAK, OSK.

$\rightarrow$ ProVerif is able to conclude on all these examples!
Case studies

Implementation
The two steps of the transformation have been implemented (≈ 2k Ocaml LoC).

Case studies
Basic Hash, Hash-Lock, Feldhofer, a variant of LAK, OSK.

→ ProVerif is able to conclude on all these examples!

(during a break if someone is interested)
Conclusion
• modelling unlinkability is rather subtle:
  —→ importance of modelling the reader, and how it is modelled;
  —→ states can introduce observables, especially in the case of a desynchronisation.
• verifying unlinkability properties is not an easy task but a lot of progress has been done.
Summary

- modelling unlinkability is rather subtle:
  - importance of **modelling the reader, and how it is modelled**;
  - **states can introduce observables**, especially in the case of a desynchronisation.
- verifying unlinkability properties is not an easy task but a lot of progress has been done.

Going a step further:

- stateful protocols (with updates) using ProVerif/GSVerif;
- from diff-equivalence to session equivalence;
- A nice way to encore unlinkability in Tamarin is to rely on (asymmetric) restrictions but currently the tool does not support them.
PEPR Cybersecurity (2022-2028)

Job offers:
- PhDs
- Post-docs
- Engineers

→ contact me: stephanie.delaune@irisa.fr